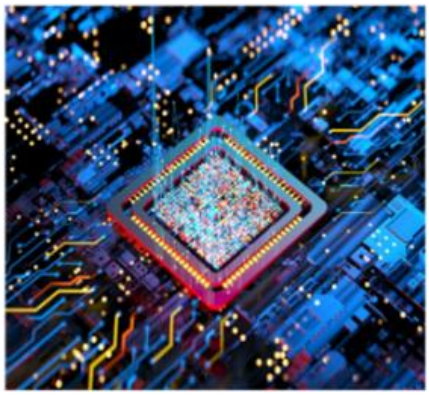


RAWABI VLSI DESIGN CENTER

ORION

VLSI Technologies

PROVIDING SERVICES FOR
THE COMPLETE VLSI DESIGN CYCLE

Final Project

BMU Verification Plan

Context

List Of Abbreviations

Abbreviation	Definition
BMU	Bit Manipulation Unit
UVM	Universal Verification Methodology
DUT	Design Under Test
RTL	Register Transfer Level
OP	Operation
RISC-V	Reduced Instruction Set Computer Version 5
Zbb	Basic Bit Manipulation Subset
Zba	Address Calculation Bit Manipulation Subset
Zbs	Single Bit Manipulation Subset
Zbp	Permutation Bit Manipulation Subset
SRL	Shift Right Logical
SRA	Shift Right Arithmetic
MSB	Most Significant Bit
CSR	Control Status Register
FF	Flip-Flop
LZD	Leading Zero Detector
CLZ	Count Leading Zeros
CTZ	Count Trailing Zeros
CPOP	Count Population (number of 1 bits)
SEXT	Sign Extend
MSB	Most Significant Bit
LSB	Least Significant Bit
ORC	OR-Combine (OR-Compression)
REV8	Reverse Bytes (8-bit chunks)
ROR	Rotate Right
ROL	Rotate Left
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetic
SH2ADD	Shift-Left-And-Add instructions
TB	Testbench
vif	Virtual Interface
env	Environment
seq	Sequence
sb	Scoreboard
sbc	Subscriber

Introduction

Goals (Purpose) of this document

The purpose of this document is to define a complete and structured verification plan for the Bit Manipulation Unit (BMU). It identifies all functional features, corner cases, error conditions, and interface behaviors that must be validated to ensure that the design fully complies with the RISC-V BitManip specification and the requirements described in the BMU RTL document.

Scope of verification

The scope of this verification effort covers all functional, structural, and error-handling behaviors of the Bit Manipulation Unit (BMU) as defined in the official specification document.

The goal is to ensure that the BMU behaves correctly for every supported instruction, under all valid and invalid operating scenarios, and across all legal combinations of operands and control signals.

The verification scope includes the following:

1. Functional correctness of all supported BitManip instructions

    - Zbb instructions
    - Zbs instructions
    - Zbp instructions
    - Zba instructions
    - CSR write/imm behavior

2. Validation of all control fields (ap. signals)*

    - Correct activation of each instruction
    - Correct disabling of non-selected instruction paths
    - Proper enforcement of guard conditions

3. Error detection logic

    - Conflicts between CSR and BitManip operations
    - Multi-active instruction bits
    - Invalid Zba usage
    - Misuse of SHxADD

4. Interface protocol behavior

    - Reset behavior
    - Valid/invalid instruction timing
    - Handling of valid_in
    - Output register behavior (result_ff)

Assumptions and constraints

Assumptions

The following assumptions are made during the verification of the Bit Manipulation Unit (BMU):

- Single active instruction per cycle
- Synchronous design behavior
- valid_in protocol
    - When valid_in = 1, the BMU accepts the instruction during that cycle.
- CSR behavior complies with the specification
    - ap.csr_write = 1 writes either a_in or b_in depending on ap.csr_imm
- Zba enable requirements
    - SH1ADD / SH2ADD / SH3ADD are valid only when ap.zba = 1, otherwise an error is expected.
- GREV mode requirement
    - GREV is only valid when b_in[4:0] == 24 as per the specification; other values must trigger an error.
- Stable inputs during a valid cycle
    - When valid_in = 1, all BMU inputs (ap, a_in, b_in, CSR signals) remain stable until sampled on the rising clock.
- Reset behavior
    - With rst_l = 0, all internal states and result_ff output are assumed to be cleared (result = 0, error = 0).

Constraints

The following constraints apply to the verification environment and design behavior:

- Operand width constraint
    - All input operands (a_in, b_in) are strictly 32-bit values.
- Control field exclusivity
    - All operations rely on mutually exclusive ap.* control bits.
- Overlapping operations must propagate error = 1.
    - Valid instruction domain
    - Any undefined combinations of ap.* and CSR signals must be flagged as errors.
- CSR access constraints
    - Only one CSR operation may occur per instruction
    - CSR read (csr_ren_in) cannot overlap with bit manipulation operations
- Clocking constraint
    - The BMU does not support multi-cycle handshake or back-pressure.
    - The verification must assume single-cycle acceptance of each valid instruction.
- Result timing constraint

- Since result_ff is a registered output, it is expected that results are visible one clock cycle after valid_in = 1.

- Parameter constraints
  - Compile-time parameters controlling enable/disable of instruction groups (e.g., pt.BITMANIP_Zbb) are assumed to be static for all tests.

- Error propagation constraint
  - When error = 1:
    - result_ff is expected to be 0
    - For verification, we will assume result = 0 for consistency and check accordingly

Design Overview

The Bit Manipulation Unit (BMU) is a synthesizable RTL block responsible for executing all RISC-V BitManip Extension instructions. It operates as part of the processor's integer execution and provides 32-bit bit-level operations, arithmetic shift-add instructions, rotation, counting, packing, and CSR-related functionality.

Internally, the BMU contains several combinational functional units (adder, shifter, counter, reverser, packer), whose outputs are combined through a final result multiplexing stage controlled by the decoded instruction signals. The BMU enforces strict guard conditions, ensuring that only one instruction path is active at a time, and includes robust error-detection logic to flag invalid or conflicting operations. The unit is clock-synchronous and uses a registered output for stable timing within the processor pipeline.

Key features

1. Full support for RISC-V BitManip instruction subsets

   - Zbb (Basic Bit Manipulation)
   - Zbs (Single-bit manipulation)
   - Zbp (Packed and permutation operations)
   - Zba (Shift-and-add instructions)

2. Integrated Combinational Functional Units

   - Arithmetic adder for SHxADD
   - Logical shifter supporting SLL, SRL, SRA
   - Rotate unit supporting ROR/ROL
   - Leading-zero detection logic (LZD) for CLZ/CTZ
   - Population counter logic for CPOP
   - Byte reversal and OR-compression logic
   - Packing and merging logic for PACK instructions

3. Registered Output Path

- Final result is captured in result_ff
- Ensures synchronous timing and integration in processor pipeline

4. Configurable Instruction Enablement

- Compile-time parameters (pt.BITMANIP_*) allow enabling or disabling of instruction groups
- Design must maintain correct behavior when some instructions are disabled

5. Robust Error-Detection System

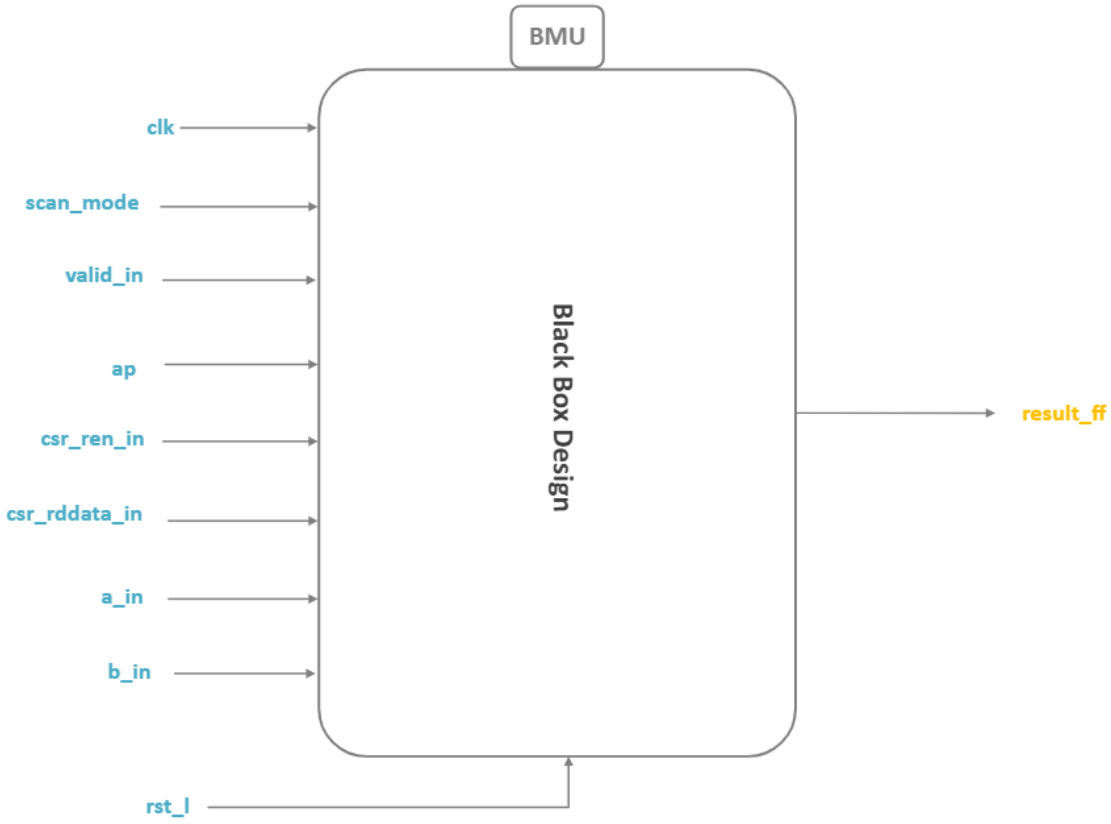
6. Strict Guard Conditions

- Only one instruction path allowed at a time
- All unrelated ap.* fields must be zero
- csr_ren_in must be zero unless executing CSR read
- Violations produce error = 1

7. Final Result Muxing

- All functional unit outputs are OR-combined
- Only the enabled unit contributes a non-zero value
- Ensures correct final result selection based on control signals

Block diagram and interface



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

Interface

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 bits	input	Represents the first operand of BMU module
b_in	32 bits	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 bits	input	CSR read data
result_ff	32 bits	output	Final computed result
error	1 bit	output	Error Indicator

Verification Strategy

Methodology

Methodology using the Universal Verification Methodology (UVM).
This approach ensures that the design is validated thoroughly, systematically, and in a reusable manner across all instruction types and error scenarios defined in the BMU specification. UVM-Based Layered Testbench Architecture.
The verification environment is built using a standard UVM structure consisting of: Interface, Driver,

Sequencer, Monitor, Scoreboard, Subscriber (Coverage Collector), Environment (env), Test Layer

Selection of Verification Tests

Tests will cover all aspects of the BMU functionalities, including:

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.**

A. 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

B. Standard Bitwise Invertor OR Operation

- A OR (~B)
- B = 0, 1, alternating bits
- Mode bit (ap.zbb) ON vs OFF
- Guard conditions + error scenarios

C. Standard Bitwise XOR Operation

- A ^ B
- Identical operands → result = 0
- Alternating patterns
- MSB = 1
- Error when overlapping operation

D. 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.**

A. 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

B. 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

C. Rotate Right Operation (ROR)

- Rotation by 0, small values, 31
- Rotation by >31 (masked)
- All patterns (zeros, ones, alternating)
- Illegal activation tests

D. 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

E. 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

A. 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

A. Set on Less Than (Unsigned SLT)

- Negative < Positive
- Positive < Negative

- Equal values
- INT_MIN vs INT_MAX
- Random signed comparisons

B. **Set on Less Than (SLT signed == default)**
- 0 < 1
- Large unsigned values
- Unsigned wrap-around
- A = B
- Edge cases (0xFFFFFFFF vs small numbers)

C. Count Trailing Zero Bits (CTZ)
- Input = 0
- Inputs with single 1-bit at positions: 0, 5, 31
- Alternating patterns
- Random values

D. Count Set Bits / Population Count (CPOP)
- All 0 → count = 0
- All 1 → count = 32
- Random patterns
- Large mixed values

E. Sign Extend Byte (siext_b)
- Sign bit of byte = 0 → zero-extend
- Sign bit of byte = 1 → extend ones
- Extreme values (0x7F, 0x80)

F. Maximum (max)
- Signed max
- A=B
- Positive vs Negative
- Large magnitudes

G. Pack (pack)
- Combine A[15:0] with B[15:0]
- A or B = 0
- Random values
- Verify byte ordering

H. Byte-Reverse Register (grev)
- Valid mode: b_in[4:0] = 24
- Invalid mode → error
- Patterns:
  - 0x00000000
  - 0xFFFFFFFF
  - Alternating bytes
  - Random

5. CSR operations

1. CSR Write

- Immediate mode (csr_imm = 1)
- Register mode (csr_imm = 0)
- Correct output of result

2.CSR Read Enable Conflict

- csr_ren_in=1 + any BitManip instruction → error
- csr_write + ap.* instruction active → error

6. Unsupported Opcode Handling

- More than one ap.* bit active → error
- Undefined instruction combination
- Disabled instruction subset (compile-time parameter off)
- ap.grev active + wrong b_in → error
- ap.sh2add active + ap.zba = 0 → error
- ap.sub active + ap.zba = 1 → error
- Default/illegal opcodes → error + result = 0

7. Random cases

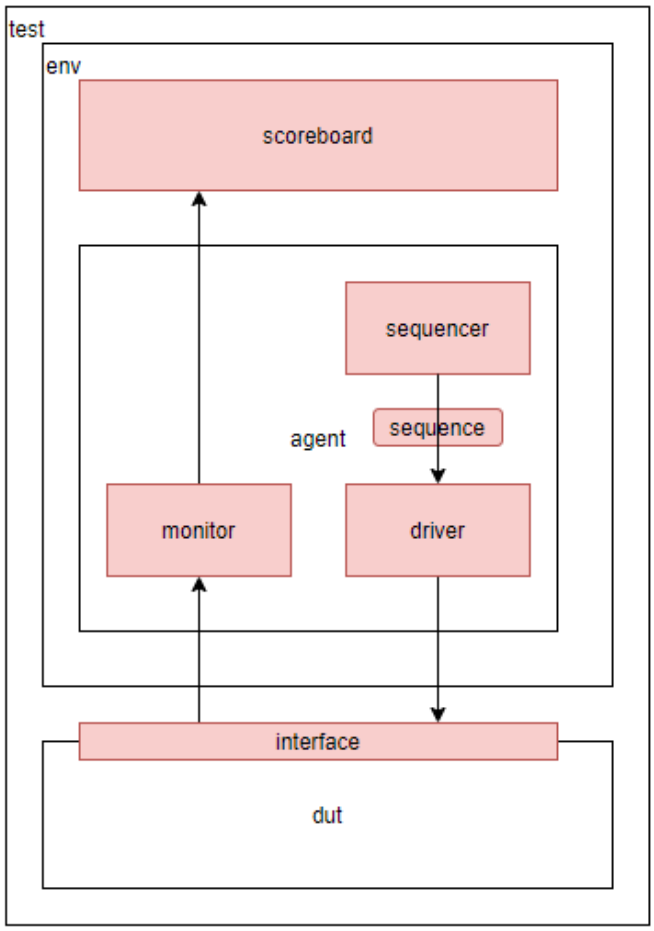
1. Random combinations of operations

- Randomly select ap.*
- Random operands
- Random CSR modes
- Random Zba enable/disable
- Random shift amounts

2. Error-random tests

- Random illegal combinations
- Random multi-active ap bits
- Random CSR conflicts
- Random invalid GREV modes

Testbench structure



Tests

For each BMU operation, a dedicated test will be implemented, consisting of all relevant directed sequences, random sequences, and error-injection sequences scenarios previously described.

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: A SystemVerilog interface is used as a middleware layer between the DUT and the UVM testbench components, enabling structured signal access through modports.

Tools and technologies:

- MobaxTerm Server .
- Intergrated 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