ORION VLSI Technologies

Bit Manipulation Unit (BMU)

Specification Document(v1.1)

Authors

Naser Tayeh

30/6/2025

orionvtech.com

Overview

The Bit_Manipulation_Unit (BMU) is a synthesizable RTL block that implements bit manipulation functionality compliant with the RISC-V BitManip extension. The unit supports instructions from Zbb, Zbs, Zbp, and Zba subsets and handles bit-level arithmetic, logical, and data packing operations. It is integrated into a larger processor pipeline and contributes to the integer execution path.

Features

- Compliant with RISC-V BitManip extensions:
  - **Zbb**: CLZ, CTZ, CPOP, MIN, MAX, SEXT.B, SEXT.H, ANDN, ORN, XNOR
  - **Zbs**: BSET, BCLR, BINV, BEXT
  - **Zbp**: ROL, ROR, PACK, PACKU, PACKH, GREV (subset), GORC (subset)
  - **Zba**: SH1ADD, SH2ADD, SH3ADD
- Integrated control and data path
- Flushing and prediction support for branch misprediction recovery
- Configurable instruction enablement using compile-time parameters
- Robust error detection logic
- Clock-gated PC register for low-power optimization

Interface Definition

Port Name	**Direction**	**Width**	**Description**
clk	Input	1 bit	Clock
rst_l	Input	1 bit	Active-low synchronous reset
scan_mode	Input	1 bit	Scan test mode control
valid_in	Input	1 bit	Instruction valid flag
ap	Input	Struct	Decoded instruction control signals
csr_ren_in	Input	1 bit	CSR read-enable
csr_rddata_in	Input	32 bits	CSR read data
a_in, b_in	Input	32 bits	Operands A and B
result_ff	Output	32 bits	Final computed result
error	Output	1 bit	Error indicator

Functional Description

Extension	Operations
Zbb	CLZ, CTZ, CPOP, MIN, MAX, ANDN, ORN, XNOR, SEXT.B/H
Zbs	BSET, BCLR, BINV, BEXT
Zbp	ROL, ROR, PACK, PACKU, PACKH, GREV (rev8), GORC (orc_b)
Zba	SH1ADD, SH2ADD, SH3ADD

Each instruction is conditionally enabled based on the `pt.BITMANIP_XXX` parameters.

Data Path and Submodules

- **Arithmetic and Shift Logic:**
  - Adder handles both ADD/SUB and SHxADD operations.
  - Shift unit supports SLL, SRL, SRA, ROL, ROR.
- **CLZ/CTZ Logic:**
  - Uses Leading Zero Detection (LZD) for CLZ.
  - CTZ is implemented by reversing the input and applying LZD.
- **CPOP Logic:**
  - Counts set bits using a 32-bit population count loop.
- **SEXT.B / SEXT.H:**
  - Sign-extend byte and halfword.
- **MIN/MAX:**
  - Compares signed/unsigned operands and selects accordingly.
- **REV8/ORC.B:**
  - Byte-level reversal and OR-compression implemented combinationally.
- **PACK Instructions:**
  - Combine halves of A and B in different formats.

Result Muxing

The result signal is formed by OR-ing individual result contributors based on decoded control signals. Only one functional unit is expected to be active per instruction

Error Detection.

Error conditions include:

- Simultaneous activation of CSR read and bit manipulation signals.
- Use of ADD/SUB inside Zba instructions.
- Misuse of SHxADD without Zba enable.

BMU Operation Descriptions

CSR Operations

Overview

This section covers CSR-related logic in the Bit Manipulation Unit (BMU), including bypass read and write operations. These operations are influenced by the fields in the ap control structure and selected inputs (csr_ren_in, csr_rddata_in, csr_write, etc.).

CSR Write Data

Description

Write operation to a CSR is controlled by ap.csr_write. The value written depends on whether the immediate flag ap.csr_imm is set.

Logic Rule

- If ap.csr_write is asserted:
    - If ap.csr_imm == 1: result = b_in
    - Else: result = a_in

Use Case Example(CSR Write Data)

```
csr_ren_in   = 1;
ap           = 0;            // All fields cleared
csr_rddata_in = 32'hABCD_1234;
// ------------------------------------------
// Output
result = 32'hABCD_1234;
error  = 0;
```

Logic operations

Overview

This section defines logical operations performed by the BMU, primarily controlled by the ap.land field. The operation mode depends on the ap.zbb extension bit and requires all other control fields to be disserted.

OR Operation

Overview

This operation performs a bitwise OR or bitwise OR with inverted b_in, depending on the Zbb extension enable field. The operation is only valid if no other instruction field is active, and CSR read is not in progress.

Operation Control Signals

- Primary Enable: `ap.lor = 1`
- Mode Select: `ap.zbb`

Execution Guard Conditions

- All other `ap.*` fields must be `0`
- `csr_ren_in` must be `0`

Behavior Description

Condition	ap.zbb	Result Expression	Error
Valid OR Operation	0	`result = a_in | b_in`	0
Valid OR with Bitwise Inversion	1	`result = a_in | ~b_in`	0
Invalid (other fields ≠ 0)	-	`result = 0`	0
Invalid (`csr_ren_in = 1`) and (other fields ≠ 0)	-	`result = 0`	1

Use Case Examples

Example 1: Standard OR

```
ap.lor       = 1;
ap.zbb       = 0;
csr_ren_in   = 0;
a_in         = 32'h0F0F_0F0F;
b_in         = 32'hF0F0_F0F0;
//----------------------------------------------------------
//Output
result = 32'hFFFF_FFFF;
error  = 0;
```

Example 2: Zbb Inverted OR

```verilog
ap.lor      = 1;
ap.zbb      = 1;
csr_ren_in  = 0;
a_in        = 32'hFFFF_0000;
b_in        = 32'h0000_00FF;
//------------------------------------------------------------------
//Output
result = 32'hFFFF_FF00;
error  = 0;
```

Example 3: Invalid OR (conflicting control fields)

```verilog
ap.lor      = 1;
ap.zbb      = 0;
ap.land     = 1;              // Invalid: another ap field is set
csr_ren_in  = 0;
// ---------------------------------------------------
// Output
result = 32'h0000_0000;
error  = 1;
```

XOR Operation

Overview

The XOR operation is part of the BMU logic functionality and is controlled by the `ap.lxor` control bit. It supports both standard bitwise XOR and XOR with an inverted operand (`~b_in`) based on the `ap.zbb` field.

Operation Control Signals

- Primary Enable: `ap.lxor = 1`
- Mode Select: `ap.zbb`

Execution Guard Conditions

- All other `ap.*` fields must be `0`
- `csr_ren_in` must be `0`

Behavior Description

Condition	ap.zbb	Result Expression	Error
Valid XOR Operation	0	`result = a_in ^ b_in`	0
Valid XOR with Bitwise Inversion	1	`result = a_in ^ ~b_in`	0
Invalid (other fields ≠ 0)	-	`result = 0`	0
Invalid (`csr_ren_in = 1`) and (other fields ≠ 0)	-	`result = 0`	1

Examples

Example 1: Standard XOR

```
ap.lxor      = 1;
ap.zbb       = 0;
csr_ren_in   = 0;
a_in         = 32'hF0F0_F0F0;
b_in         = 32'h0F0F_0F0F;
//--------------------------------------------------
//Output
result = 32'hFFFF_FFFF;
error  = 0;
```

Example 2: XOR with Inverted Operand

```
ap.lxor      = 1;
ap.zbb       = 1;
csr_ren_in   = 0;
a_in         = 32'hAAAA_AAAA;
b_in         = 32'h5555_5555;
//-----------------------------------
// Output
result = 32'h0000_0000;  // a_in ^ ~b_in = a_in ^ 0xAAAA_AAAA = 0
error  = 0;
```

Example 3: Invalid Operation (csr_ren_in is 1)

```
ap.lxor      = 1;
ap.zbb       = 0;
csr_ren_in   = 1;
//---------------------------------------------------
//Output
result = 32'h0000_0000;
error  = 1;
```

Shifting and Masking Operations

Overview

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

Note: `csr_ren_in` must be `0` for valid operation.

General Guard Conditions

- All other `ap.*` fields must be `0`
- `csr_ren_in = 0`
- Only one of the above shift/mask control signals may be active per cycle

Right Logical Shift (SRL)

Overview

The Right Logical Shift (SRL) operation is part of the BMU’s shifting functionality. It is controlled by the ap.srl signal and performs a logical (zero-fill) right shift of a_in by the amount specified in the lower 5 bits of b_in.

Operation Control Signals

- Primary Enable: ap.srl = 1

Execution Guard Conditions

- All other ap.* fields must be 0
- csr_ren_in must be 0

Behavior Description

Condition	Result Expression	Error
Valid SRL Operation	SRL	0
Invalid (other ap.* fields ≠ 0)	result = 0	0
Invalid (csr_ren_in = 1) and (other fields ≠ 0)	result = 0	1

Example

```
// Inputs
a_in         = 32'hF000_0000;
b_in         = 32'h0000_0004;
ap.srl       = 1'b1;
csr_ren_in   = 1'b0;
ap.*         = all other fields = 0;

// Output
result = 0x0F00_0000; // = 0x0F00_0000
error  = 0;
```

Right Arithmetic Shift (SRA)

Overview

The Right Arithmetic Shift (SRA) operation is part of the BMU's shifting functionality. It preserves the sign of a signed number during the shift by extending the sign bit as it shifts right. The shift amount is determined by the lower 5 bits of `b_in`.

Operation Control Signals

- Primary Enable: `ap.sra = 1`

Execution Guard Conditions

- All other `ap.*` fields must be `0`
- `csr_ren_in` must be `0`

Behavior Description

Condition	Result Expression	Error
Valid SRA Operation	SRA	`0`
Invalid (other ap.* fields ≠ 0)	`result = 0`	0
Invalid (`csr_ren_in = 1`) and (other fields ≠ 0)	`result = 0`	`1`

Example

```
// Inputs
a_in          = 32'hF000_0000;  // Interpreted as negative in signed context
b_in          = 32'h0000_0004;
ap.sra        = 1'b1;
csr_ren_in    = 1'b0;
ap.*          = all other fields = 0;

// Output
result = 0xFF00_0000; // = 0xFF00_0000 (sign-extended)
error  = 0;
```

Rotate Right (ROR)

Overview

The Rotate Right (ROR) operation is a bit manipulation instruction that rotates the bits of a_in to the right by the number of positions specified in the lower 5 bits of b_in. Bits shifted out from the LSB side are reintroduced at the MSB side.

Operation Control Signals

- Primary Enable: ap.ror = 1

Execution Guard Conditions

- All other ap.* fields must be 0
- csr_ren_in must be 0

Behavior Description

Condition	Result Expression	Error
Valid ROR Operation	ROR	0
Invalid (other ap.* fields ≠ 0)	`result = 0`	0
Invalid (`csr_ren_in = 1`) and (other fields ≠ 0)	`result = 0`	`1`

Example(ROR)

```
// Inputs
a_in          = 32'h89ABCDEF;
b_in          = 32'h0000_0004;
ap.ror        = 1'b1;
csr_ren_in    = 1'b0;
ap.*          = all other fields = 0;
//-----------------------------------------------------------------
// Output
result = 0xF89ABCDE; // = 0xF89ABCDE
error  = 0;
```

Bit Inverse (BINV)

Overview

The Bit Inverse (BINV) operation toggles (inverts) a single bit in the operand a_in at the bit position specified by the lower 5 bits of b_in. All other bits remain unchanged.

Operation Control Signals

- Primary Enable: ap.binv = 1

Execution Guard Conditions

- All other ap.* fields must be 0
- csr_ren_in must be 0

Behavior Description

Condition	Result Expression	Error
Valid BINV Operation	BINV	`0`
Invalid (other ap.* fields ≠ 0)	`result = 0`	0
Invalid (`csr_ren_in = 1`) and (other fields ≠ 0)	`result = 0`	`1`

Example(BINV)

```
// Inputs
a_in        = 32'hFFFF_FFFF;    // All bits set
b_in        = 32'h0000_0002;    // Bit index 2 to invert
ap.binv     = 1'b1;
csr_ren_in  = 1'b0;
ap.*        = all other fields = 0;
// -----------------------------------------------------------
// Output
// Invert bit 2 of a_in (bit 2 was 1)
result = 32'hFFFF_FFFB;  // bit 2 cleared (inverted)
error  = 0;
```

Shift Left by 2 and Add (SH2ADD)

Overview

The Shift Left by 2 and Add (SH2ADD) operation shifts `a_in` left by 2 bits and adds the result to `b_in`. This operation is valid only when `ap.zba` is asserted (equal to 1).

Operation Control Signals

- Primary Enable: `ap.sh2add = 1`
- Required Mode: `ap.zba = 1`

Execution Guard Conditions

- ap.zba must be 1
- All other ap.* fields must be 0 except ap.sh2add and ap.zba
- csr_ren_in must be 0

Behavior Description

Condition	Result Expression	Error
Valid SH2ADD Operation	**SH2ADD**	`0`
Invalid (`ap.zba != 1`)	`result = 0`	1
Invalid (other `ap.*` fields ≠ 0 and `csr_ren_in`)	`result = 0`	`1`

Example(SH2ADD)

```
// Inputs
a_in       = 32'd4;        // Binary: 0000...0100
b_in       = 32'd7;        // Binary: 0000...0111
ap.sh2add  = 1'b1;
ap.zba     = 1'b1;
csr_ren_in = 1'b0;
ap.*       = all other fields = 0;
//-------------------------------------------------------
// Output
result = 32'd23;
error  = 0;
```

Arithmetic Operations

Overview
The arithmetic operations in the Bit Manipulation Unit (BMU) perform basic addition, subtraction, and set-less-than comparisons. Each operation is controlled by a dedicated ap.* field. All other ap.* fields must be 0 except the one corresponding to the active operation.

Subtract (sub)

Overview

The subtract operation is part of the arithmetic functionality of the BMU. It is controlled by the ap.sub control bit and requires ap.zba to be 0. It subtracts the value in b_in from a_in.

Operation Control Signals

- Primary Enable: ap.sub = 1
- Mode Constraint: ap.zba = 0

Execution Guard Conditions

- All other ap.* fields must be 0.
- csr_ren_in must be 0.
- If ap.zba != 0, the result is forced to 0 and error is set to 1.

Behavior Description

Condition	Result Expression	Error
Valid Subtraction (`ap.sub=1, ap.zba=0`)	`result = a_in - b_in`	`0`
Invalid (`ap.zba != 0`)	`result = 0`	1
Invalid (other fields ≠ 0 and csr_ren_in=1)	`result = 0`	`1`

Example(SUB)

```
a_in  = 32'd20;
b_in  = 32'd7;
ap.sub  = 1'b1;
ap.zba  = 1'b0;
// ----------------------------------------------------
// Output
result  = 32'd13;
error   = 0;
```

Bit Manipulation

Overview

Bit manipulation instructions are specialized operations that directly process and modify individual bits within a register.

Set on Less Than (SLT)

Overview

The SLT operation compares two operands (a_in and b_in) and sets the result to 1 if a_in is less than b_in.

The comparison is signed if ap.unsign = 0.

Operation Control Signals

- Primary Enable: ap.slt = 1
- other enables = ap.sub = 1
- Signed/Unsigned Select: ap.unsign
  - 0 → Signed comparison (default for SLT)
  - 1 → Unsigned comparison (SLTU equivalent)

Execution Guard Conditions

- All other ap.* fields must be 0.
- csr_ren_in must be 0.

Behavior Description

Condition	ap.unsign	Result Expression	Error
Valid SLT operation	0	SLT (SIGNED)	0
Valid SLTU operation	1	SLT(UNSIGNED)	0
Invalid (other ap fields ≠ 0 and csr_ren_in = 1)	-	`result = 0`	1

Examples(SLT)

Example1

```
// Example 1: Signed SLT
a_in  = 32'hFFFFFFFE; // -2
b_in  = 32'h00000001; // 1
ap.slt = 1;
ap.unsign = 0;
ap.sub    = 1;
// Output
result = 1 ; // result = 1
error = 0
```

Example2

```
// Example 2: Unsigned SLTU
a_in  = 32'hFFFFFFFE; // 4294967294
b_in  = 32'h00000001; // 1
ap.slt = 1;
ap.sub = 1;
ap.unsign = 1;
result = 0; // result = 0Count Leading Zero Bits (CLZ)
```

Count Trailing Zero Bits (CTZ)

Overview

The CTZ operation counts the number of consecutive zeros in a_in, starting from the least significant bit (LSB) and moving toward the most significant bit (MSB). It is useful in bit scanning and locating the first set bit from the right.

Operation Control Signals

- Primary Enable: ap.ctz = 1

Behavior Description

Condition	Result Expression	Error
Valid input	`result = number of trailing 0 bits in a_in`	0
Input is all zeros	`result = 0`	0
Invalid (other ap fields ≠ 0 and csr_ren_in = 1)	`result = 0`	1

Example(CTZ)

```
// --- Count Trailing Zeros ---
a_in         = 32'h00000100;
ap.clz       = 0;
ap.ctz       = 1;
csr_ren_in   = 0;
// ------------------------------------
// Output
error        = 0;
result       = count_trailing_zeros(a_in); // result = 8, error = 0
```

Count Set Bits / Population Count (CPOP)

Overview

The CPOP operation counts the total number of bits set to 1 in the input `a_in`. This is commonly used in bit manipulation algorithms and for parity or weight calculations.

Operation Control Signals

- Primary Enable: `ap.cpop = 1`

Behavior Description

Condition	Result Expression	Error
Valid input	`result = number of bits set to 1 in a_in`	0
Invalid (other ap fields ≠ 0) and and csr_ren_in = 1)	`result = 0`	1

Example(CPOP)

```
// Example: Count Set Bits (Population Count)

// --- Valid Operation ---
a_in        = 32'hF0F0F00F; //
ap.cpop     = 1;
csr_ren_in  = 0;
// output
error       = 0;
result      =16; // result = 16 (there are 16 ones in a_in)
```

Sign Extend Byte (siext_b)

Overview

The siext_b operation sign-extends the lower byte (bits [7:0]) of the input a_in to the full 32-bit width by replicating the sign bit (bit 7) into the higher bits [31:8].

Operation Control Signals

- Primary Enable: ap.siext_b = 1

Behavior Description

Condition	Result Expression	Error
Valid `siext_b`	siext_b	0
Invalid (other ap fields ≠ 0 and csr_ren_in = 1)	`result = 0`	1

Example(siext_b)

```
// Example 1: Sign extend positive value
a_in        = 32'h0000007F; // +127 (bit 7 = 0)
ap.siext_b  = 1;
// output
result      = 0x0000007F; // result = 0x0000007F
```

Maximum (max)

Overview

The `max` operation compares two signed integers (`a_in` and `b_in`) and returns the larger value. It is part of the RISC-V Bit Manipulation (Zbb) extension.

Operation Control Signals

- Primary Enable: ap.max = 1
- other enables: ap.sub = 1

Behavior Description

Condition	Result Expression	Error
Valid `max` operation	max	0
Invalid (other ap fields ≠ 0 and csr_ren_in = 1)	`result = 0`	1

Example(max)

```
// Example 1: Positive numbers
a_in        = 32'h0000000A; // 10
b_in        = 32'h00000014; // 20
ap.max      = 1;
ap.sub      = 1;
result      = 20; // result = 20
```

Pack (pack)

Overview

The pack operation concatenates the least significant 16 bits of b_in and a_in to form a 32-bit result. This is part of the RISC-V Bit Manipulation (Zbp) extension.

Operation Control Signals

- Primary Enable: ap.pack = 1

Behavior Description

Condition	Result Expression	Error
Valid `pack` operation	**pack**	0
Invalid (other ap fields ≠ 0 and csr_ren_in = 1)	`result = 0`	1

Example(pack)

```verilog
// Example 1: Combine lower halves
a_in        = 32'h12345678;
b_in        = 32'hABCDEF12;
ap.pack     = 1;
// output
result      = 'hEF125678; // result = 32'hEF125678
```

Byte-Reverse Register (grev)

Overview

The `grev` operation reverses the order of bytes in `a_in`. Only byte ordering is reversed, not the individual bits within each byte. This operation is controlled by `ap.grev` and requires `b_in` to be equal to 24 (`5'b11000`) in its lower 5 bits.

Operation Control Signals

- Primary Enable: `ap.grev = 1`

- Mode Select: `b_in[4:0] = 5'b11000`

Behavior Description

Condition	Result Expression	Error
Valid `grev` (b_in = 24)	grev	0
Invalid (b_in ≠ 24)	`result = 0`	0
Invalid (other ap fields ≠ 0 and csr_ren_in = 1)	`result = 0`	1

Example(grev)

```
// Example 1: Standard byte reverse
a_in      = 32'h12345678;
b_in      = 32'd24;
ap.grev   = 1;
// ----------------------------------
// output
result    =32'h78563412 ; // result = 32'h78563412
error = 0;
```

Deliverables

- RTL files: `Bit_Manipulation_Unit.sv`, `rtl_param.vh`, `rtl_pkg.sv`
- Unit testbench with stimuli per instruction group
- Functional coverage model for each operation