Digital IMC BNN Accelerator for Real-Time Space Debris Classification

RTL Design and Verification Report

Sree Prabha
Aswathi
Emerson
Amal M

Digital University Kerala

October 15, 2025

Abstract

This document presents the complete Register-Transfer Level (RTL) design and functional verification of a hardware accelerator for a Binarized Neural Network (BNN). The accelerator is designed to perform the computationally intensive convolutional layer for a real-time space debris classification application. The design is modular, written in Verilog, and verified using a bottom-up strategy with dedicated testbenches for each component. This report details the architecture, module implementation, and verification methodology, serving as the primary documentation for the hardware design phase of the project.

Contents

1 Introduction

The objective of this project is to design a low-power, high-performance hardware accelerator for a BNN from specification to a GDSII layout file. This document focuses on the initial and most critical phase: the RTL design and its functional verification. The hardware implements the core computational logic of a 3x3 binarized convolution, which consists of parallel XNOR operations followed by a popcount and activation function.

2 System Architecture

The accelerator is designed as a hierarchical system composed of a datapath and a control path. The datapath performs the actual computation, while the control path manages data flow, stores configuration (weights and thresholds), and communicates with an external host controller via an SPI interface.

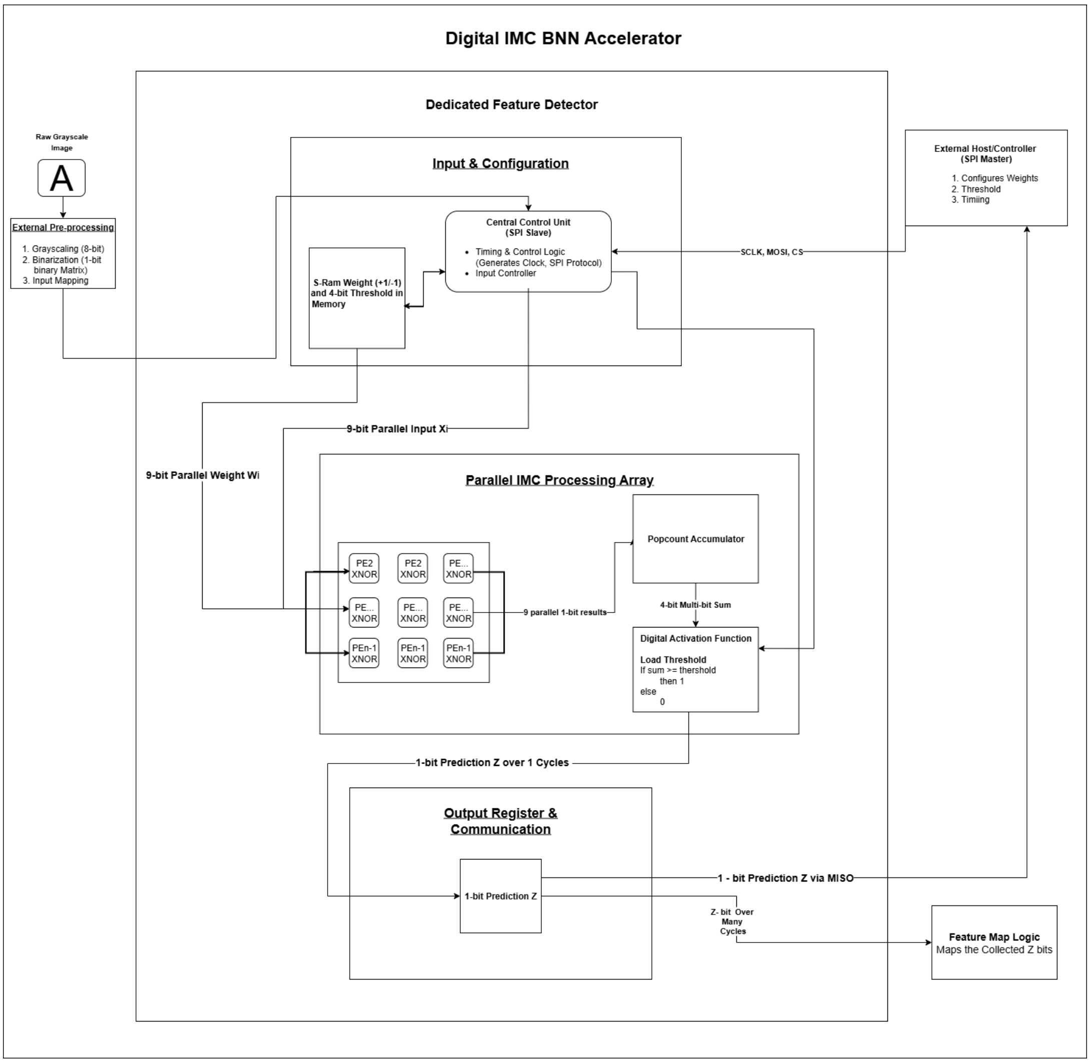

Figure 1: Hardware block diagram of the BNN accelerator.

3 RTL Module Implementation: Datapath

The datapath is a purely combinational pipeline designed for maximum parallelism and throughput.

3.1 Processing Element: pe_xnor.v

The fundamental building block. It performs a single bitwise XNOR operation, which is the mathematical equivalent of multiplication for binarized values.

```
module pe_xnor (
    input  wire data_in,
    input  wire weight_in,
    output wire result_out
);
    assign result_out = ~ (data_in ^ weight_in);
endmodule
```

Listing 1: Verilog code for the Processing Element.

3.2 IMC Processing Array: imc_processing_array.v

This module instantiates a 3x3 array of `pe_xnor` modules to perform nine XNOR operations in parallel in a single clock cycle.

```
module imc_processing_array (
    input  wire [8:0] data_bus,
    input  wire [8:0] weight_bus,
    output wire [8:0] result_bus
);
    genvar i;
    generate
        for (i = 0; i < 9; i = i + 1) begin : pe_instance_loop
            pe_xnor pe_inst (
                .data_in    (data_bus[i]),
                .weight_in  (weight_bus[i]),
                .result_out (result_bus[i])
            );
        end
    endgenerate
endmodule
```

Listing 2: Verilog code for the 3x3 IMC Array.

3.3 Popcount Accumulator: popcount_accumulator.v

This module counts the number of '1's in the 9-bit result from the IMC array using an efficient two-stage adder tree. The output is a 4-bit sum.

```
module popcount_accumulator (
    input  wire [8:0] result_bus,
    output wire [3:0] sum_out
);
    wire [1:0] sum_stage1_a, sum_stage1_b, sum_stage1_c;

    assign sum_stage1_a = result_bus[0] + result_bus[1] + result_bus[2];
    assign sum_stage1_b = result_bus[3] + result_bus[4] + result_bus[5];
    assign sum_stage1_c = result_bus[6] + result_bus[7] + result_bus[8];

    assign sum_out = sum_stage1_a + sum_stage1_b + sum_stage1_c;
endmodule
```

Listing 3: Verilog code for the Popcount Accumulator.

3.4 Digital Activation: `digital_activation.v`

The final stage of the pipeline. It compares the 4-bit sum with a 4-bit threshold and outputs a single '1' if the sum is greater than or equal to the threshold.

```
module digital_activation (
    input  wire [3:0] sum_in,
    input  wire [3:0] threshold_in,
    output wire       prediction_z
);
    assign prediction_z = (sum_in >= threshold_in);
endmodule
```

Listing 4: Verilog code for the Digital Activation Function.

3.5 Integrated Datapath: `bnn_datapath.v`

This module integrates all the above components into a single, cohesive datapath pipeline.

```
module bnn_datapath (
    input  wire [8:0] data_bus,
    input  wire [8:0] weight_bus,
    input  wire [3:0] threshold_in,
    output wire       prediction_z
);
    wire [8:0] xnor_results;
    wire [3:0] popcount_sum;

    imc_processing_array imc_array_inst (
        .data_bus(data_bus),
        .weight_bus(weight_bus),
        .result_bus(xnor_results)
    );
    popcount_accumulator popcount_inst (
        .result_bus(xnor_results),
        .sum_out(popcount_sum)
    );
    digital_activation activation_inst (
        .sum_in(popcount_sum),
        .threshold_in(threshold_in),
        .prediction_z(prediction_z)
    );
endmodule
```

Listing 5: Verilog code for the Integrated BNN Datapath.

4 RTL Module Implementation: Control Path

4.1 SPI Slave Interface: `spi_slave.v`

This module handles communication with the external host. It deserializes incoming SPI data into 8-bit bytes.

```
module spi_slave (
    input  wire sclk,
    input  wire cs_n,
    input  wire mosi,
    output reg  rx_valid,
    output wire [7:0] rx_byte
);
    reg [2:0] bit_count;
    reg [7:0] rx_shreg;
```

```
    assign rx_byte = rx_shreg;

    always @(posedge sclk or negedge cs_n) begin
        if (!cs_n) begin
            if (bit_count == 3'd7) begin
                rx_shreg <= {rx_shreg[6:0], mosi};
                rx_valid <= 1'b1;
                bit_count <= 3'd0;
            end else begin
                rx_shreg <= {rx_shreg[6:0], mosi};
                bit_count <= bit_count + 1;
                rx_valid <= 1'b0;
            end
        end else begin
            bit_count <= 3'd0;
            rx_shreg  <= 8'd0;
            rx_valid  <= 1'b0;
        end
    end
endmodule
```

Listing 6: Verilog code for the SPI Slave Interface.

5 Verification Strategy and Testbenches

Each module was verified with a dedicated testbench to ensure functional correctness before integration. The testbenches and their primary test cases are documented below.

5.1 Testbench: `pe_xnor_tb.v`

Verified all 4 possible input combinations for the XNOR gate.

5.2 Testbench: `imc_processing_array_tb.v`

Verified the parallel operation of all 9 PEs with various test vectors.

5.3 Testbench: `popcount_accumulator_tb.v`

Verified the accumulator with zero, maximum, and mixed input patterns.

5.4 Testbench: `digital_activation_tb.v`

Verified the greater-than, less-than, and equal-to boundary conditions of the comparator.

5.5 Testbench: `bnn_datapath_tb.v`

Performed an end-to-end test of the integrated datapath to confirm expected outputs based on controlled inputs.

5.6 Testbench: `spi_slave_tb.v`

Simulated an SPI master sending multiple bytes to verify the deserialization logic.

6 Conclusion and Future Work

The datapath and communication interface for the BNN accelerator have been successfully designed in Verilog and functionally verified. The design is robust, modular, and ready for the next stage. Future work involves designing the top-level control FSM, integrating all components, and proceeding with the ASIC backend flow, including synthesis, place-and-route, and sign-off verification.