Appendix: Source Code Listings

This appendix contains the Verilog and SystemVerilog source code used during the experimental analysis. These listings include both the professional reference designs and the AI-generated implementations for the BRAM and GPIO benchmarks.

A. Professional BRAM Reference (Xilinx Template)

```
// Professional Source: Xilinx Vivado Language Template (Fixed)
// Module: True Dual Port RAM with Dual Clocks
// Standardized values: Width=32, Depth=1024 for benchmarking

module xilinx_true_dual_port_ram #(
    parameter RAM_WIDTH = 32,                       // Fixed: 32-bit data
    parameter RAM_DEPTH = 1024,                     // Fixed: 1024 entries
    parameter RAM_PERFORMANCE = "HIGH_PERFORMANCE", // Select "HIGH_PERFORMANCE" or "LOW_LATENCY"
    parameter INIT_FILE = ""                        // Leave blank
)(
    input [clogb2(RAM_DEPTH-1)-1:0] addra,  // Port A address bus
    input [clogb2(RAM_DEPTH-1)-1:0] addrb,  // Port B address bus
    input [RAM_WIDTH-1:0] dina,             // Port A RAM input data
    input [RAM_WIDTH-1:0] dinb,             // Port B RAM input data
    input clka,                             // Port A clock
    input clkb,                             // Port B clock
    input wea,                              // Port A write enable
    input web,                              // Port B write enable
    input ena,                              // Port A RAM Enable
    input enb,                              // Port B RAM Enable
    input rsta,                             // Port A output reset
    input rstb,                             // Port B output reset
    input regcea,                           // Port A output register enable
    input regceb,                           // Port B output register enable
    output [RAM_WIDTH-1:0] douta,           // Port A RAM output data
    output [RAM_WIDTH-1:0] doutb            // Port B RAM output data
);

    // Scaling function for address width
    function integer clogb2;
    input integer depth;
    for (clogb2=0; depth>0; clogb2=clogb2+1)
        depth = depth >> 1;
    endfunction

    // 2D Array for RAM storage
    reg [RAM_WIDTH-1:0] BRAM [RAM_DEPTH-1:0];
    reg [RAM_WIDTH-1:0] ram_data_a = {RAM_WIDTH{1'b0}};
    reg [RAM_WIDTH-1:0] ram_data_b = {RAM_WIDTH{1'b0}};

    // Initialize memory to zero
    generate
        if (INIT_FILE != "") begin: use_init_file
```

```
            initial
                $readmemh(INIT_FILE, BRAM, 0, RAM_DEPTH-1);
        end else begin: init_bram_to_zero
            integer ram_index;
            initial
                for (ram_index = 0; ram_index < RAM_DEPTH; ram_index =
    ram_index + 1)
                    BRAM[ram_index] = {RAM_WIDTH{1'b0}};
        end
    endgenerate

    // PORT A OPERATION
    always @(posedge clka) begin
        if (ena) begin
            if (wea)
                BRAM[addra] <= dina;
            ram_data_a <= BRAM[addra];
        end
    end

    // PORT B OPERATION
    always @(posedge clkb) begin
        if (enb) begin
            if (web)
                BRAM[addrb] <= dinb;
            ram_data_b <= BRAM[addrb];
        end
    end

    // Output Register Stage
    generate
        if (RAM_PERFORMANCE == "LOW_LATENCY") begin: no_output_register
            assign douta = ram_data_a;
            assign doutb = ram_data_b;
        end else begin: output_register
            reg [RAM_WIDTH-1:0] douta_reg = {RAM_WIDTH{1'b0}};
            reg [RAM_WIDTH-1:0] doutb_reg = {RAM_WIDTH{1'b0}};

            always @(posedge clka) begin
                if (rsta)
                    douta_reg <= {RAM_WIDTH{1'b0}};
                else if (regcea)
                    douta_reg <= ram_data_a;
            end

            always @(posedge clkb) begin
                if (rstb)
                    doutb_reg <= {RAM_WIDTH{1'b0}};
                else if (regceb)
                    doutb_reg <= ram_data_b;
            end

            assign douta = douta_reg;
            assign doutb = doutb_reg;
        end
    endgenerate

```

```
endmodule
```

Listing 1: Xilinx True Dual Port RAM Template

B. AI-Generated BRAM (ChatGPT-5.2)

```
module true_dual_port_ram #(
    parameter int RAM_WIDTH = 32,
    parameter int RAM_DEPTH = 1024,
    parameter string RAM_PERFORMANCE = "LOW_LATENCY", // or "HIGH_PERFORMANCE"
    parameter string INIT_FILE = ""                   // optional memory init file
)(
    // Port A
    input  logic                          clka,
    input  logic                          wea,
    input  logic [$clog2(RAM_DEPTH)-1:0] addra,
    input  logic [RAM_WIDTH-1:0]          dina,
    output logic [RAM_WIDTH-1:0]          douta,

    // Port B
    input  logic                          clkb,
    input  logic                          web,
    input  logic [$clog2(RAM_DEPTH)-1:0] addrb,
    input  logic [RAM_WIDTH-1:0]          dinb,
    output logic [RAM_WIDTH-1:0]          doutb,

    // Reset (used only for output registers)
    input  logic                          rst
);

    localparam int ADDR_WIDTH = $clog2(RAM_DEPTH);

    // Memory array
    logic [RAM_WIDTH-1:0] ram [0:RAM_DEPTH-1];

    // Internal read data (pre-output register)
    logic [RAM_WIDTH-1:0] douta_int;
    logic [RAM_WIDTH-1:0] doutb_int;

    // ------------------------------------------------------------
    // Optional memory initialization
    // ------------------------------------------------------------
    generate
        if (INIT_FILE != "") begin : gen_init_file
            initial $readmemh(INIT_FILE, ram);
        end else begin : gen_init_zero
            integer i;
            initial begin
                for (i = 0; i < RAM_DEPTH; i = i + 1)
                    ram[i] = '0;
            end
        end
    endgenerate

    // ------------------------------------------------------------
```

```
    // Port A - Read-First behavior
    // ------------------------------------------------------------
    always_ff @(posedge clka) begin
        // Read old data first
        douta_int <= ram[addra];

        // Write after read
        if (wea) begin
            ram[addra] <= dina;
        end
    end

    // ------------------------------------------------------------
    // Port B - Read-First behavior
    // ------------------------------------------------------------
    always_ff @(posedge clkb) begin
        // Read old data first
        doutb_int <= ram[addrb];

        // Write after read
        if (web) begin
            ram[addrb] <= dinb;
        end
    end

    // ------------------------------------------------------------
    // Output register stage (performance selection)
    // ------------------------------------------------------------
    generate
        if (RAM_PERFORMANCE == "HIGH_PERFORMANCE") begin :
   gen_high_perf
            // Extra registered output stage (2-cycle latency)
            always_ff @(posedge clka) begin
                if (rst)
                    douta <= '0;
                else
                    douta <= douta_int;
            end

            always_ff @(posedge clkb) begin
                if (rst)
                    doutb <= '0;
                else
                    doutb <= doutb_int;
            end
        end else begin : gen_low_latency
            // Direct output (1-cycle latency)
            always_comb begin
                douta = douta_int;
                doutb = doutb_int;
            end
        end
    endgenerate

endmodule
```

Listing 2: ChatGPT-Generated Dual Port RAM

C. Professional GPIO Reference (OpenTitan)

```
// PROFESSIONAL BENCHMARK: OpenTitan GPIO (Cleaned for Synthesis)
// Source: Derived from lowRISC OpenTitan 'gpio.sv'

// 1. PACKAGE DEFINITION
package gpio_pkg;
  parameter int NumIOs = 32;
endpackage

// 2. MAIN MODULE
module opentitan_gpio
  import gpio_pkg::*;
(
  input  logic             clk_i,
  input  logic             rst_ni,

  // Register Interface (Simplified for Benchmarking)
  input  logic             reg_we,
  input  logic [31:0]      reg_addr,
  input  logic [31:0]      reg_wdata,
  output logic [31:0]      reg_rdata,

  // GPIO Ports
  input  logic [NumIOs-1:0] cio_gpio_i,
  output logic [NumIOs-1:0] cio_gpio_o,
  output logic [NumIOs-1:0] cio_gpio_en_o,

  // Interrupt Output
  output logic [NumIOs-1:0] intr_gpio_o
);

  // Registers
  logic [31:0] data_in_q;
  logic [31:0] direct_out_q;

  // Input Synchronization (Standard Double Flop for Safety)
  logic [31:0] cio_gpio_sync_1;
  logic [31:0] cio_gpio_sync_2;

  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
        cio_gpio_sync_1 <= '0;
        cio_gpio_sync_2 <= '0;
    end else begin
        cio_gpio_sync_1 <= cio_gpio_i;
        cio_gpio_sync_2 <= cio_gpio_sync_1;
    end
  end

  // Edge Detection Logic
  logic [31:0] event_rise;
  logic [31:0] event_fall;

  always_ff @(posedge clk_i) begin
    data_in_q <= cio_gpio_sync_2;
  end

```

```
  assign event_rise = cio_gpio_sync_2 & ~data_in_q;
  assign event_fall = ~cio_gpio_sync_2 & data_in_q;

  // ----------------------------------------------------------
  // THE CRITICAL TEST: MASKED WRITES
  // Professional code uses efficient bitwise logic in one cycle.
  // ----------------------------------------------------------
  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
      direct_out_q <= '0;
    end else if (reg_we) begin
      case (reg_addr[7:0])
        8'h00: direct_out_q <= reg_wdata; // Direct Write

        // Masked Lower: Update lower 16 bits based on upper 16 mask
        8'h04: direct_out_q[15:0] <= (direct_out_q[15:0] & ~reg_wdata
    [31:16]) | (reg_wdata[15:0] & reg_wdata[31:16]);

        // Masked Upper: Update upper 16 bits based on upper 16 mask
        8'h08: direct_out_q[31:16] <= (direct_out_q[31:16] & ~reg_wdata
    [31:16]) | (reg_wdata[15:0] & reg_wdata[31:16]);
      endcase
    end
  end

  assign cio_gpio_o = direct_out_q;
  assign cio_gpio_en_o = {32{1'b1}}; // All outputs enabled for this
    test

  // Interrupt Logic (Simplified for synthesis comparison)
  // Hardcoded enables to force synthesis to keep the logic
  logic [31:0] intr_ctrl_en_rising  = 32'hFFFFFFFF;
  logic [31:0] intr_ctrl_en_falling = 32'hFFFFFFFF;

  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni)
      intr_gpio_o <= '0;
    else
      intr_gpio_o <= (event_rise & intr_ctrl_en_rising) |
                     (event_fall & intr_ctrl_en_falling);
  end

endmodule
```

Listing 3: OpenTitan GPIO Benchmark

D. AI-Generated GPIO (Claude Sonnet 4.5)

```
module gpio #( parameter int unsigned GpioWidth = 32 ) (
  input logic clk_i, rst_ni,
  input logic [GpioWidth-1:0] cio_gpio_i,
  output log// Professional GPIO Module with Interrupt Support
// Copyright (c) 2024. All rights reserved.
// Designed for 32-bit General Purpose Input/Output control

module gpio #(
  parameter int unsigned GpioWidth = 32
```

```
) (
  input  logic                      clk_i,
  input  logic                      rst_ni,

  // Pad interface
  input  logic [GpioWidth-1:0]    cio_gpio_i,
  output logic [GpioWidth-1:0]    cio_gpio_o,
  output logic [GpioWidth-1:0]    cio_gpio_en_o,

  // Register interface - Direct Write
  input  logic                      direct_out_we_i,
  input  logic [GpioWidth-1:0]    direct_out_data_i,
  input  logic                      direct_oe_we_i,
  input  logic [GpioWidth-1:0]    direct_oe_data_i,

  // Register interface - Masked Write Upper [31:16]
  input  logic                      masked_out_upper_we_i,
  input  logic [15:0]               masked_out_upper_data_i,
  input  logic [15:0]               masked_out_upper_mask_i,
  input  logic                      masked_oe_upper_we_i,
  input  logic [15:0]               masked_oe_upper_data_i,
  input  logic [15:0]               masked_oe_upper_mask_i,

  // Register interface - Masked Write Lower [15:0]
  input  logic                      masked_out_lower_we_i,
  input  logic [15:0]               masked_out_lower_data_i,
  input  logic [15:0]               masked_out_lower_mask_i,
  input  logic                      masked_oe_lower_we_i,
  input  logic [15:0]               masked_oe_lower_data_i,
  input  logic [15:0]               masked_oe_lower_mask_i,

  // Interrupt configuration
  input  logic [GpioWidth-1:0]    intr_ctrl_en_rising_i,
  input  logic [GpioWidth-1:0]    intr_ctrl_en_falling_i,
  input  logic [GpioWidth-1:0]    intr_ctrl_en_lvlhigh_i,
  input  logic [GpioWidth-1:0]    intr_ctrl_en_lvllow_i,

  // Interrupt status (write-1-to-clear)
  input  logic                      intr_state_we_i,
  input  logic [GpioWidth-1:0]    intr_state_data_i,
  output logic [GpioWidth-1:0]    intr_state_o,

  // Interrupt output
  output logic [GpioWidth-1:0]    intr_o
);

  //
   ============================================================================

  // Internal Signals
  //
   ============================================================================

  logic [GpioWidth-1:0] gpio_out_q, gpio_out_d;
  logic [GpioWidth-1:0] gpio_oe_q, gpio_oe_d;
  logic [GpioWidth-1:0] gpio_in_sync1_q, gpio_in_sync2_q;
  logic [GpioWidth-1:0] gpio_in_prev_q;
  logic [GpioWidth-1:0] intr_state_q, intr_state_d;
```

```
64
65  // Edge detection signals
66  logic [GpioWidth-1:0] rising_edge, falling_edge;
67
68  //
     ============================================================================
69  // Input Synchronization (Double-Flop for Metastability Protection)
70  //
     ============================================================================
71  always_ff @(posedge clk_i or negedge rst_ni) begin
72    if (!rst_ni) begin
73      gpio_in_sync1_q <= '0;
74      gpio_in_sync2_q <= '0;
75      gpio_in_prev_q  <= '0;
76    end else begin
77      gpio_in_sync1_q <= cio_gpio_i;
78      gpio_in_sync2_q <= gpio_in_sync1_q;
79      gpio_in_prev_q  <= gpio_in_sync2_q;
80    end
81  end
82
83  //
     ============================================================================
84  // Edge Detection Logic
85  //
     ============================================================================
86  assign rising_edge  = gpio_in_sync2_q & ~gpio_in_prev_q;
87  assign falling_edge = ~gpio_in_sync2_q & gpio_in_prev_q;
88
89  //
     ============================================================================
90  // GPIO Output Data Register Logic
91  //
     ============================================================================
92  always_comb begin
93    gpio_out_d = gpio_out_q;
94
95    // Direct write has priority
96    if (direct_out_we_i) begin
97      gpio_out_d = direct_out_data_i;
98    end else begin
99      // Masked write upper [31:16]
100     if (masked_out_upper_we_i) begin
101       for (int i = 0; i < 16; i++) begin
102         if (masked_out_upper_mask_i[i]) begin
103           gpio_out_d[16+i] = masked_out_upper_data_i[i];
104         end
105       end
106     end
107
108     // Masked write lower [15:0]
109     if (masked_out_lower_we_i) begin
```

```
      for (int i = 0; i < 16; i++) begin
        if (masked_out_lower_mask_i[i]) begin
          gpio_out_d[i] = masked_out_lower_data_i[i];
        end
      end
    end
  end
end

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    gpio_out_q <= '0;
  end else begin
    gpio_out_q <= gpio_out_d;
  end
end

assign cio_gpio_o = gpio_out_q;

//
 ============================================================================

// GPIO Output Enable Register Logic
//
 ============================================================================

always_comb begin
  gpio_oe_d = gpio_oe_q;

  // Direct write has priority
  if (direct_oe_we_i) begin
    gpio_oe_d = direct_oe_data_i;
  end else begin
    // Masked write upper [31:16]
    if (masked_oe_upper_we_i) begin
      for (int i = 0; i < 16; i++) begin
        if (masked_oe_upper_mask_i[i]) begin
          gpio_oe_d[16+i] = masked_oe_upper_data_i[i];
        end
      end
    end

    // Masked write lower [15:0]
    if (masked_oe_lower_we_i) begin
      for (int i = 0; i < 16; i++) begin
        if (masked_oe_lower_mask_i[i]) begin
          gpio_oe_d[i] = masked_oe_lower_data_i[i];
        end
      end
    end
  end
end

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    gpio_oe_q <= '0;
  end else begin
    gpio_oe_q <= gpio_oe_d;
```

```
    end
  end

  assign cio_gpio_en_o = gpio_oe_q;

  //
   ============================================================================

  // Interrupt Detection Logic
  //
   ============================================================================

  logic [GpioWidth-1:0] intr_detect;

  always_comb begin
    for (int i = 0; i < GpioWidth; i++) begin
      intr_detect[i] = (intr_ctrl_en_rising_i[i]  & rising_edge[i]) |
                       (intr_ctrl_en_falling_i[i] & falling_edge[i]) |
                       (intr_ctrl_en_lvlhigh_i[i] & gpio_in_sync2_q[i])
    |
                       (intr_ctrl_en_lvllow_i[i]  & ~gpio_in_sync2_q[i
   ]);
    end
  end

  //
   ============================================================================

  // Interrupt State Register (Write-1-to-Clear)
  //
   ============================================================================

  always_comb begin
    intr_state_d = intr_state_q;

    // Set interrupt bits when detected
    intr_state_d = intr_state_q | intr_detect;

    // Clear interrupt bits on write-1-to-clear
    if (intr_state_we_i) begin
      intr_state_d = intr_state_d & ~intr_state_data_i;
    end
  end

  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
      intr_state_q <= '0;
    end else begin
      intr_state_q <= intr_state_d;
    end
  end

  assign intr_state_o = intr_state_q;
  assign intr_o = intr_state_q;

  //
   ============================================================================
```

```
  // Assertions for Verification
  //
    ============================================================================

`ifndef SYNTHESIS
  // Check that direct and masked writes don't occur simultaneously
  always_ff @(posedge clk_i) begin
    if (rst_ni) begin
      assert (!(direct_out_we_i && (masked_out_upper_we_i ||
    masked_out_lower_we_i)))
        else $error("Direct and masked GPIO output writes should not
    occur simultaneously");
      assert (!(direct_oe_we_i && (masked_oe_upper_we_i ||
    masked_oe_lower_we_i)))
        else $error("Direct and masked GPIO OE writes should not occur
    simultaneously");
    end
  end
`endif

endmoduleic [GpioWidth-1:0] cio_gpio_o, cio_gpio_en_o,
  // ... [Additional Interface Ports] ...
);
  // Software-style loop-based masking (Causes Logic Inefficiency)
  always_comb begin
    gpio_out_d = gpio_out_q;
    if (masked_out_lower_we_i) begin
      for (int i = 0; i < 16; i++) begin
        if (masked_out_lower_mask_i[i]) begin
          gpio_out_d[i] = masked_out_lower_data_i[i];
        end
      end
    end
  end
endmodule
```

Listing 4: Claude-Generated GPIO with Loop-Based Masking