Direct-Mapped Cache Memory Design using Verilog

Name: Ragul Ganesh Anitha Palanivel

Institution: Binghamton University

Abstract

This report presents the implementation of a Direct-Mapped Cache memory system using Verilog HDL. The design incorporates a byte-addressable memory interface with four cache lines, a 6-bit tag, and a 2-bit index. It simulates memory operations using a backing main memory and tests read/write actions including cache hits and misses. The project is verified through simulation and waveform analysis, with supporting schematic views generated from Vivado.

Table of Contents

Introduction to Cache Memory

A cache memory is a high-speed data storage layer that stores a subset of data from a slower main memory. It is widely used in CPUs and embedded systems to reduce memory access latency and improve performance.

Key Functions and Uses of Cache:
- Temporarily stores frequently accessed data for faster access.
- Reduces the average time to access memory.
- Minimizes bottlenecks caused by slower main memory.
- Provides a mechanism for locality of reference (both spatial and temporal).
- Commonly used in CPUs, GPUs, embedded systems, and high-performance computing environments.

Project Summary

This project demonstrates the design and verification of a simple Direct-Mapped Cache using Verilog. The cache is byte-addressable and interfaces with a backing memory of 256 bytes. It includes:
- 4 cache lines
- 6-bit tag, 2-bit index

- Write-through policy for memory consistency

Key operations:
1. Write: Updates both memory and cache.
2. Read Hit: Directly serves data from cache.
3. Read Miss: Fetches data from memory and updates cache.

The design is verified using a System Testbench that covers all operations (write, read hit, read miss).

Working of the Designed Cache

This project implements a direct-mapped cache with the following characteristics:

- Address Width: 8-bit (256 locations)
- Cache Lines: 4 (2-bit index)
- Tag Width: 6 bits (upper 6 bits of address)
- Data Width: 8-bit (byte-addressable)
- Main Memory: Simulated with a 256-byte array

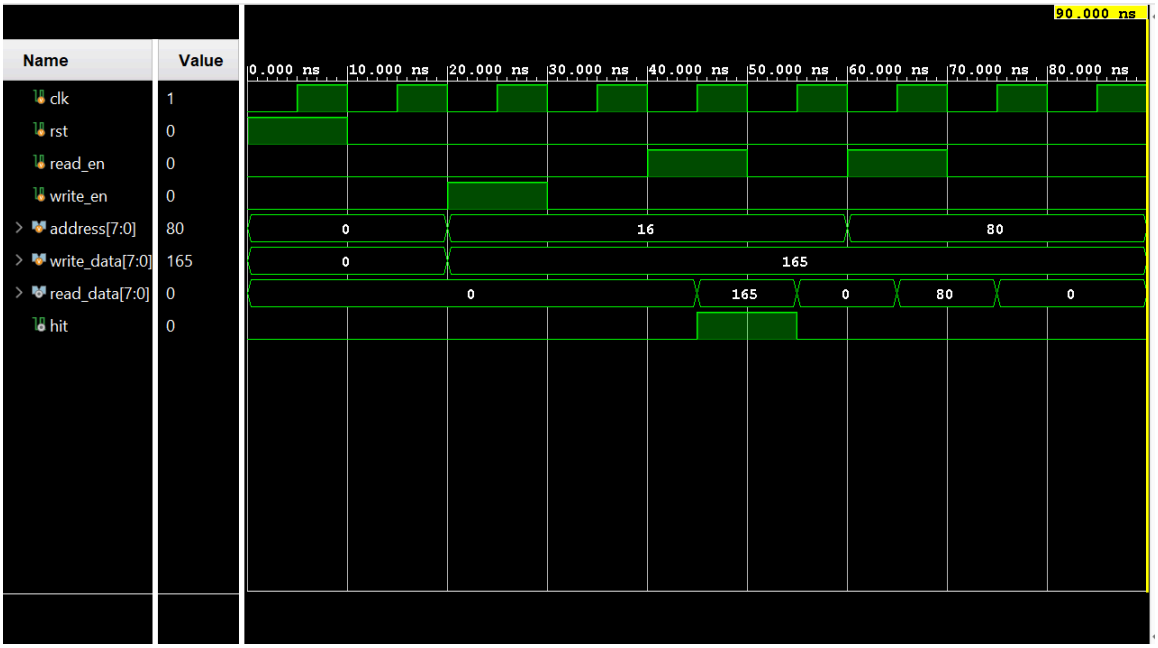
Operations:
1. Write: Updates both main memory and cache using write-through policy. On a tag match, the cache line is updated directly.
2. Read Hit: If the cache line is valid and the tag matches, data is served from cache (hit).
3. Read Miss: If no tag match or invalid data is loaded from memory into cache (miss) and served.

This structure uses valid bits and tag comparison to simulate realistic cache behavior. On reset, the cache is cleared, and memory is initialized.

Waveform Behavior Explanation

The simulation waveform illustrates:
1. A WRITE to address 0x10 with data 0xA5. This updates the cache line and memory.
2. A READ from the same address 0x10. Since it's already cached, it is a HIT and returns 0xA5.
3. A READ from address 0x50. Although it shares the same index as 0x10, the tag is different, resulting in a MISS. It fetches data from main memory (which was initialized to 0x50 = 80) and updates the cache.

The `hit` signal is HIGH only during the read hit at 0x10. The `read_data` shows expected values for each read.

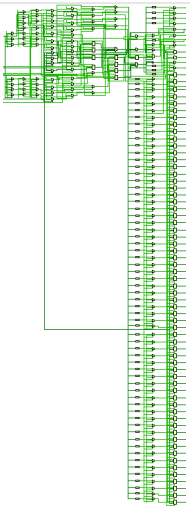


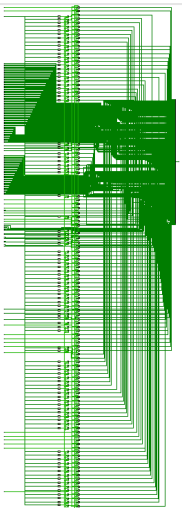
Schematic Diagram Explanation

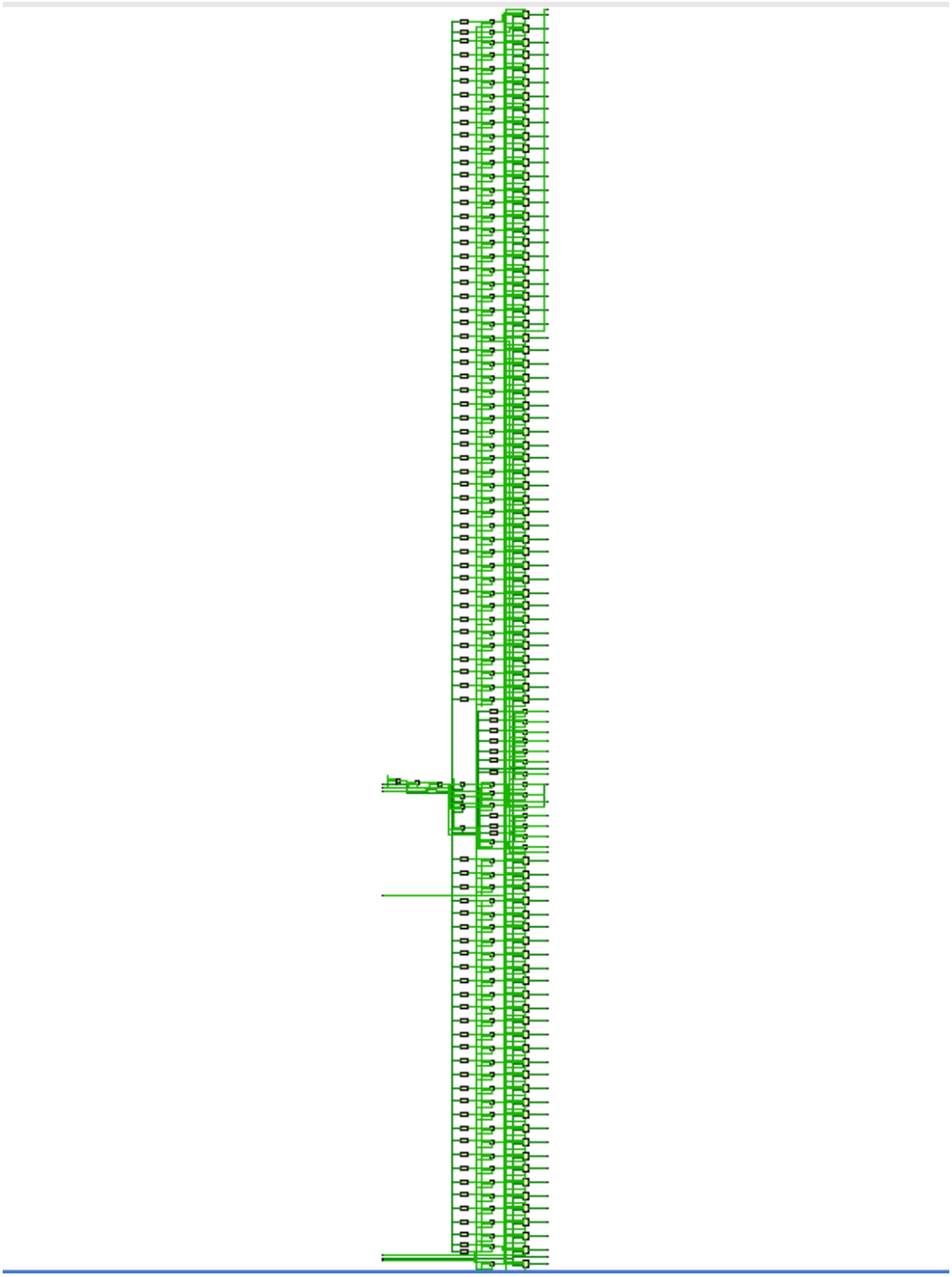
These schematics are auto-generated by Vivado and show hierarchical views of the cache system:

1. First schematic: Represents the RTL view of the top-level cache module and its integration.
2. Second schematic: A fully expanded netlist view showing internal connections between registers, comparators, and control signals.
3. Third schematic: Flattened view highlighting the I/O ports and memory arrays.

These provide a visual representation of the logic synthesized and mapped onto FPGA resources.







Conclusion and Future Improvements

The implemented Direct-Mapped Cache effectively demonstrates the principles of caching, including tag matching, hit/miss detection, and integration with backing memory. It successfully simulates a working memory model and cache operation through Verilog and Vivado tools.

Future improvements could include:

- Implementing multi-level caching (L1/L2)
- Supporting block sizes >1 word
- Integrating replacement policies (e.g., LRU)
- Adding performance metrics such as hit rate calculation

Appendix: Verilog Code

direct_mapped_cache.v

```
module direct_mapped_cache (
    input clk,
    input rst,
    input read_en,
    input write_en,
    input [7:0] address,
    input [7:0] write_data,
    output reg [7:0] read_data,
    output reg hit
);

    // Cache: 4 lines
    reg [7:0] cache_data [0:3];
    reg [5:0] cache_tag [0:3];
    reg cache_valid [0:3];

    // Backing Memory: 256 locations
    reg [7:0] main_memory [0:255];

    wire [1:0] index = address[3:2];     // 2-bit index
    wire [5:0] tag = address[7:2];       // 6-bit tag

    integer i;

    // Split address for cache and memory access
```

```
always @(posedge clk or posedge rst) begin
  if (rst) begin
    for (i = 0; i < 4; i = i + 1) begin
      cache_valid[i] <= 0;
      cache_data[i] <= 0;
      cache_tag[i] <= 0;
    end

    for (i = 0; i < 256; i = i + 1)
      main_memory[i] <= i; // preload memory with dummy data (0,1,2,...)

    hit <= 0;
    read_data <= 0;
  end else begin
    hit <= 0;
    read_data <= 8'h00;

    if (read_en) begin
      if (cache_valid[index] && cache_tag[index] == tag) begin
        // Cache hit
        hit <= 1;
        read_data <= cache_data[index];
      end else begin
        // Cache miss  fetch from main memory and update cache
        hit <= 0;
        cache_data[index] <= main_memory[address];
        cache_tag[index] <= tag;
        cache_valid[index] <= 1;
        read_data <= main_memory[address];
      end
    end

    if (write_en) begin
      // Write-through policy: write to both cache (if valid & tag match) and memory
      main_memory[address] <= write_data;
      if (cache_valid[index] && cache_tag[index] == tag) begin
        cache_data[index] <= write_data;
      end else begin
        // Optionally: update cache on write miss too
        cache_data[index] <= write_data;
        cache_tag[index] <= tag;
        cache_valid[index] <= 1;
      end
```

```
        end
      end
    end
endmodule
```

tb_direct_mapped_cache.v

```
`timescale 1ns / 1ps
module tb_direct_mapped_cache;

    reg clk, rst;
    reg read_en, write_en;
    reg [7:0] address;
    reg [7:0] write_data;
    wire [7:0] read_data;
    wire hit;

    // DUT instance
    direct_mapped_cache uut (
        .clk(clk),
        .rst(rst),
        .read_en(read_en),
        .write_en(write_en),
        .address(address),
        .write_data(write_data),
        .read_data(read_data),
        .hit(hit)
    );

    // Clock
    always #5 clk = ~clk;

    initial begin
        $monitor("T=%0t | Addr=0x%0h | WData=0x%0h | RData=0x%0h | Hit=%b | Read=%b
| Write=%b",
                 $time, address, write_data, read_data, hit, read_en, write_en);

        clk = 0;
        rst = 1;
        read_en = 0;
        write_en = 0;
        address = 8'h00;
        write_data = 8'h00;
```

```
    // Reset
    #10 rst = 0;

    // ---- WRITE to address 0x10 (index=2)
    #10 address = 8'h10; write_data = 8'hA5;
    write_en = 1;
    #10 write_en = 0;

    // ---- READ (HIT) from 0x10
    #10 address = 8'h10;
    read_en = 1;
    #10 read_en = 0;

    // ---- READ (MISS) from 0x50 (same index=2 but different tag)
    #10 address = 8'h50;
    read_en = 1;
    #10 read_en = 0;

    #20 $finish;
  end
endmodule
```