به نام خدا

آزمایشگاه طراحی سیستم‌های دیجیتال

گزارش کار نهم

TCAM

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

بهار ۱۴۰۱

استاد:

علیرضا اجلالی

دستیار آموزشی:

سحر رضاقلی

گروه:

هیربد بهنام

۹۹۱۷۱۳۳۳

علی نظری

۹۹۱۰۲۴۰۱

عرفان مجیبی

۹۹۱۰۵۷۰۷

فهرست

مقدمه ۲

گزارش آزمایش ۳
بخش ۱. طراحی TCAM ۳
قسمت ۱.۱ طراحی کلی قطعه ۳
قسمت ۱.۲ توضیح داخل قطعه ۵
قسمت ۱.۳ تست بنچ ۶
قسمت ۱.۴ سنتز ۷

نتیجه‌گیری ۹

مقدمه

در این آزمایش می‌خواهیم که یک TCAM را طراحی کنیم. حافظه‌ای که طراحی می‌کنیم بسیار شبیه به CAM است ولی با این تفاوت که زمانی که در مموری write می‌کنیم، می‌توانیم یک bit mask را نیز در مموری ذخیره کنیم که عملا یک نوع dont care در بیت‌های مموری هستند. زمانی که فرمان read به حافظه می‌دهیم، حافظه لیستی از آدرس عبارات match شده را بر می‌گرداند. این آدرس به صورت bit mask است. بدین معنی که در صورت بیت iام برابر یک باشد، بدین معنی است که در این خانه‌ی حافظه مقداری وجود دارد. همچنین در صورتی که حافظه پر باشد، دیگر چیزی در آن نوشته نمی‌شود و باید حتما ریست شود.

گزارش آزمایش

بخش ۱. طراحی *TCAM*

قسمت ۱.۱ طراحی کلی قطعه

کد این آزمایش در زیر آمده است:

```
module TCAM(
    input wire clk,
    input wire reset,
    input wire write_enable,
    input wire [15:0] input_data,
    input wire [15:0] input_unknown_bits,
    output reg [15:0] hits,
    output reg write_success
);

    reg [15:0] unknown_bit_positions [15:0];
    reg [15:0] mem [15:0];
    reg [15:0] mem_filled;

    integer i, j;

    reg match_in_read;

    // Write part
    always @(posedge clk) begin
        // Preset values
        hits = 0;
        write_success = 0;
        // Check status
        if (reset)
            mem_filled = 0;
        else begin
            if (write_enable) begin
                for (i = 0; i < 16; i = i + 1) begin
                    // Only write if we have not written anything before and this memory is empty
                    if (!write_success && !mem_filled[i]) begin
                        write_success = 1;
                        mem[i] = input_data;
                        unknown_bit_positions[i] = input_unknown_bits;
```

```
                        mem_filled[i] = 1;
                    end
                end
            end else begin
                // Check all memory places
                for (i = 0; i < 16; i = i + 1) begin
                    // Check if there is something in memory
                    if (mem_filled[i]) begin
                        match_in_read = 1; // Assume that there is match then remove if needed later
                        for (j = 0; j < 16; j = j + 1) begin
                            // If bits are not equal and the position is not unknown (x) then remove the match
                            if (mem[i][j] != input_data[j] && !unknown_bit_positions[i][j])
                                match_in_read = 0;
                        end
                        // Check if it was a match
                        if (match_in_read)
                            hits[i] = 1;
                    end
                end
            end
        end
    end
endmodule
```

در ابتدا سیگنال‌های خروجی و ورودی مدار را بررسی می‌کنیم:

- clk و reset: همان طور که از اسم این ورودی‌ها پیدا است، این ورودی‌ها کلاک و ریست هستند. دقت کنید که ریست سنکرون است.
- write_enable: در صورتی که این سیگنال فعال باشد عملیات نوشتن بر روی حافظه را انجام می‌دهیم. در غیر این صورت عملیات خواندن را انجام می‌دهیم. دقت کنید که مموری که طراحی می‌کنیم امکان خواندن و نوشتن همزمان را ندارد.
- input_data: این ورودی به دو منظور استفاده می‌شود: یکی اینکه زمانی که می‌خواهیم یک write بر روی مموری انجام دهیم، بیت‌های غیر dont care از این ورودی می‌آیند و در حافظه ذخیره می‌شوند. درباره‌ی بیت‌های dont care در ادامه توضیح داده خواهد شد. در زمان read مقدار این ورودی مورد جست و جو قرار می‌گیرد.
- input_unknown_bits: زمانی که می‌خواهیم دیتایی را در مموری بریزیم، یک سری از بیت‌های آن dont care (یا همان x) هستند. برای اینکه بتوانیم این بیت‌ها را مشخص کنیم دو ورودی دریافت می‌کنیم. یکی همان input_data بود و یکی دیگر همین input_unknown_bits. تمامی بیت‌های ۱ داخل input_unknown_bits عملا dont care تلقی می‌شوند و در زمان سرچ با ۰ و ۱ مچ می‌شوند. دقت کنید که بیت‌های متناظر ۱ در input_data مهم نیست که ۰ باشند یا ۱.
- hits: این مقدار نشان می‌دهد که بعد از read در کدام خانه‌های حافظه match رخ داده است. این یک آرایه‌ی ۱۶ بیتی است. هر بیت ۱ نشان دهنده‌ی یک مچ در خانه‌ی متناظرش در مموری است. به عنوان مثال در صورتی که بیت سوم ۱

باشد، یک مچ در خانه‌ی سوم رخ داده است. طول این آرایه باید به اندازه‌ی تعداد خانه‌های حافظه باشد که در اینجا ۱۶ است.

- write_success: مشخص است که حافظه‌ای که در اختیار داریم تعداد خانه‌ی محدودی دارد. به عنوان مثال در حافظه‌ای که طراحی کردیم پس از ۱۶ بار رایت، دیگر حافظه پر می‌شود و نمی‌توان چیزی در آن نوشت. حال زمانی که دستور رایت به حافظه می‌دهیم از کجا باید بدانیم که حافظه پر بوده است یا نه؟ به کمک این سیگنال خروجی می‌توان این موضوع را فهمید؛ زمانی که بعد از دستور رایت، این سیگنال برابر ۱ است، بدین معنی است که مموری جای خالی داشته است و نوشتن به درستی صورت گرفته است. در غیر این صورت بدین معنا است که حافظه دیگر جای خالی ندارد و برای خالی کردن آن باید سیگنال ریست را فعال کنیم.

قسمت ۲.۱ توضیح داخل قطعه

در ابتدا در قطعه رجیسترهای مربوط به مموری را تعریف می‌کنیم. اسم یکی از این رجیسترها mem است که 16×16 بیت است. این رجیستر عملا تمامی ورودی‌هایی که در حافظه نوشتیم را ذخیره سازی می‌کند. همان طور که معلوم است این یک پک ۱۶تایی اعداد ۱۶ بیتی است که به ترتیب اندازه‌ی حافظه و طول کلمات را مشخص می‌کند. دقت کنید که در این سری رجیستر dont careها ذخیره نمی‌شوند. در عوض بیت‌های dont care در رجیسترهای unknown_bit_positions ذخیره می‌شود. هر بیت یک در این ارایه بیتی نشان دهنده‌ی dont care بود آن در حافظه اصلی (mem) است. در نهایت یک آرایه‌ی ۱۶ بیتی نیز داریم که به ما می‌گوید که در کدام یک از خانه‌های حافظه مقدار با معنایی وجود دارد. این سری رجیستر برای این بدین صورت تعریف شده‌اند که در صورت نیاز بتوان به راحتی امکان حذف یک داده از مموری را به قطعه اضافه کرد. در صورتی که تمام بیت‌های این رجیستر ۱ باشد، مموری پر بوده است و نمی‌توان دیگر writeی انجام داد.

در ادامه دو متغیر i و j برای شمارنده‌ی `loop` نگاه کردن تمام خانه‌های حافظه تعریف شده‌اند. همچنین تعریف متغیر j برای زمانی است که می‌خواهیم تمام بیت‌های یک خانه‌ی حافظه را با بیت‌های ورودی و بیت‌های dont care مقایسه کنیم.

در نهایت یک رجیستر به نام match_in_read تعریف کرده‌ایم. وضیفه‌ی این رجیستر این است که زمانی که در حال سرچ مموری برای match هستیم، این متغیر نگه می‌دارد که آیا دیتایی که در حال بررسی کردیم یک match برای دیتای ورودی است یا خیر. (دقت کنید که در زبان وریلاگ `continue` نداریم!)

حال وارد یک بلاک `always` می‌شویم که حساس به لبه‌ی بالارونده‌ی clk است. در بلاک مذکور ابتدا خروجی‌های hits و write_success را صفر می‌کنیم. در صورت نیاز این مقادیر دوباره حساب می‌شوند. سپس چک می‌کنیم که آیا سیگنال ریست فعال است یا خیر. در صورتی که این سیگنال فعال بود، تمامی بیت‌های mem_filled را صفر می‌کنیم (عملا مموری را invalid می‌کنیم). در غیر این صورت چک می‌کنیم که آیا باید در مموری بنویسیم یا باید از آن بخوانیم با توجه به سیگنال write_enable در صورتی که نیاز به خواندن بود، باید تمامی خانه‌های حافظه را بررسی کنیم و اولین خانه‌ای را پیدا می‌کنیم که پر نباشد. پر نبودن خانه را می‌توان با ۰ بود بیت iام در mem_filled بررسی کرد. همچنین اگر دقت کنید اینجا متغیر write_success را نیز بررسی می‌کنیم. این متغیر نشان می‌دهد که آیا قبل از این بار حلقه، ما عدد خواسته شده را در جایی از حافظه نوشته‌ایم یا خیر. مشخص است که در صورتی که این عدد را جایی نوشته بودیم، دیگر نباید دوباره آن‌را در خانه‌ی دیگری از حافظه بنویسیم. حال در صورتی که مموری خالی پیدا کردیم و این دیتا را جایی ننوشته بودیم، باید دیتا را در مموری بنویسیم. برای این کار، در ابتدا input_data را وارد خانه‌ی مورد نظر در mem می‌کنیم. سپس input_unknown_bits را وارد خانه‌ی متناظر در unknown_bit_positions می‌کنیم. در نهایت نیز آن خانه از مموری را در mem_filled برابر یک قرار می‌دهیم به این منظور که در این خانه از حافظه یک دیتایی وجود دارد. همچنین همان طور که گفته شد باید write_success را برابر ۱ قرار دهیم. دقت کنید که باز هم همان طور که گفته شد این متغیر در اول بلاک always مساوی ۰ می‌شود.

در غیر این صورت باید از حافظه بخوانیم. برای خواندن و پیدا کردن دیتا در حافظه یک حلقه بر روی تمامی خانه‌ها می‌زنیم. دقت کنید که باید فقط خانه‌هایی را چک کنیم که در آنها دیتای valid وجود داشته باشد یا به عبارتی دیگر mem_filled آن‌ها برابر ۱ باشد. در صورتی که دیتای validی در مموری وجود داشت، چک کنیم که آیا دیتای ورودی با دیتای داخل مموری مطابقت

دارد یا خیر. برای چک کردن این مطابقت، تمامی بیت‌های ورودی و دیتای در مموری با هم مقایسه می‌کنیم؛ زمانی که یکی از دو بیت با هم برابر نبودند و بیت dont care برای آن بیت فعال نبود، این یک match نیست. در غیر این صورت یک match داریم. برای در آوردن مچ بودن یا نبودن ابتدا رجیستر match_in_read را برابر ۱ می‌کنیم. بر این اساس جلو می‌رویم که هر خانه از مموری یک مچ است مگر اینکه خلافش ثابت شود! به همین منظور این رجیستر را یک می‌کنیم. سپس تمامی بیت‌های مموری را با input_data مقایسه می‌کنیم. در صورتی که بیت‌ها مشابه هم نبودند و آن بیت dont care نیز نبود، match_in_read را برابر ۰ می‌کنیم. در خارج از حلقه match_in_read را چک می‌کنیم که آیا برابر ۱ است یا خیر. در صورتی که برابر ۱ بود باید آدرس حافظه را به لیست آدرس‌های match اضافه کنیم. برای این کار کافی است که بیت iام را برابر یک قرار دهیم.

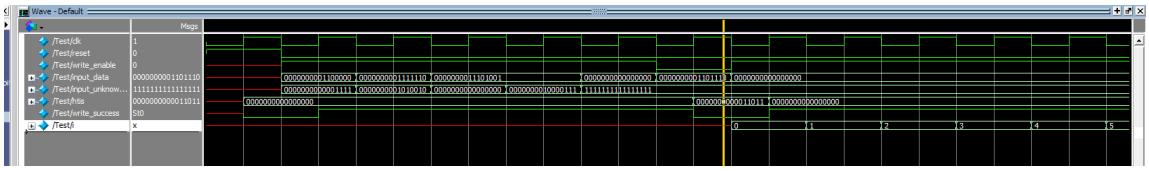
قسمت ۳.۱ تست بنچ

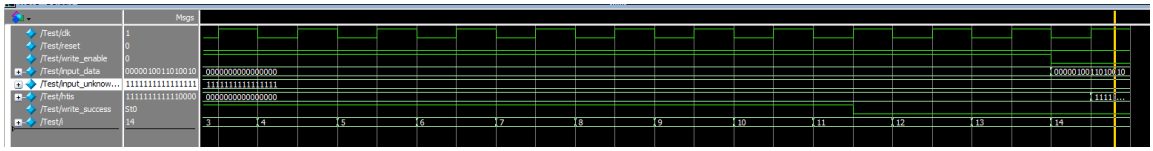
کدمان را به کمک کد زیر تست می‌کنیم:

```
`include "TCAM.v"
module Test;
    reg clk, reset, write_enable;
    reg [15:0] input_data, input_unknown_bits;
    wire [15:0] htis;
    wire write_success;
    TCAM tcam(clk, reset, write_enable, input_data, input_unknown_bits,
   htis, write_success);

    initial clk = 0;
    always #5 clk = ~clk;

    integer i;

    initial begin
        reset = 1;
        #10;
        reset = 0;
        write_enable = 1;
        input_data = 16'b01100000;
        input_unknown_bits = 16'b1111;
        #10;
        input_data = 16'b01111110;
        input_unknown_bits = 16'b01010010;
        #10;
        input_data = 16'b11101001;
        input_unknown_bits = 0;
        #10;
        input_data = 16'b11101001;
        input_unknown_bits = 16'b10000111;
        #10;
```

```
        input_data = 0;
        input_unknown_bits = {16{1'b1}};
        #10;
        write_enable = 0;
        input_data = 16'b01101110;
        #10;
        write_enable = 1;
        input_data = 0;
        input_unknown_bits = {16{1'b1}};
        for (i = 0; i < 14; i = i + 1)
            #10;
        write_enable = 0;
        input_data = 1234;
        #10;
        $finish;
    end
endmodule
```

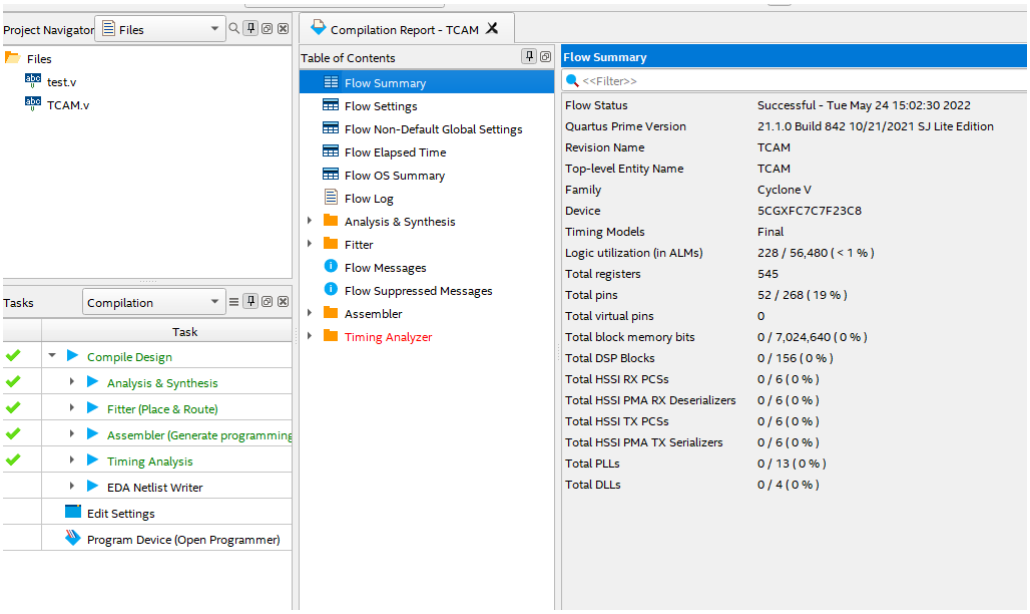
در این تست ابتدا قطعه را ریست می‌کنیم. سپس دقیقا اعداد ذکر شده در گزارش کار بعلاوه‌ی چند عدد دیگر که با عدد گزارش کار مچ نمی‌شوند را وارد مموری می‌کنیم. سپس با صفر کردن سیگنال خوندن write دستور خواندن به مموری می‌دهیم. با دادن این دستور خط آدرس‌های همان اعداد سوال در خروجی نمایان می‌شود. سپس شروع می‌کنیم و حافظه را پر dont care می‌کنیم. اعدادی که وارد حافظه می‌کنیم با هر عدد خروجی مچ می‌شوند. متوجه می‌شویم که پس از مدتی حافظه پر می‌شود و دیگر عدد در آن write نمی‌شود. در نهایت با درخواست دادن مجدد عدد داخل دستور کار می‌بینیم که اعدادی که در آخر وارد کردیم در خروجی ظاهر می‌شوند.





قسمت ۴.۱ سنتز

کد را در کوآرتوس سنتز می‌کنیم. مشاهده می‌شود که کد به درستی سنتز می‌شود.


Project Navigator
Files
Files
test.v
TCAM.v
Tasks
Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programming
Timing Analysis
EDA Netlist Writer
Edit Settings
Program Device (Open Programmer)
Compilation Report - TCAM
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Settings
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Fitter
Flow Messages
Flow Suppressed Messages
Assembler
Timing Analyzer
Flow Summary
<<Filter>>
Flow Status Successful - Tue May 24 15:02:30 2022
Quartus Prime Version 21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name TCAM
Top-level Entity Name TCAM
Family Cyclone V
Device 5CGXFC7C7F23C8
Timing Models Final
Logic utilization (in ALMs) 228 / 56,480 (< 1 %)
Total registers 545
Total pins 52 / 268 (19 %)
Total virtual pins 0
Total block memory bits 0 / 7,024,640 (0 %)
Total DSP Blocks 0 / 156 (0 %)
Total HSSI RX PCSs 0 / 6 (0 %)
Total HSSI PMA RX Deserializers 0 / 6 (0 %)
Total HSSI TX PCSs 0 / 6 (0 %)
Total HSSI PMA TX Serializers 0 / 6 (0 %)
Total PLLs 0 / 13 (0 %)
Total DLLs 0 / 4 (0 %)

نتیجه‌گیری

در این آزمایش توانستیم یک TCAM را به صورت کامل طراحی کنیم. این TCAM قابلیت این را دارد که هم یک bit mask برای dont careها ذخیره کند و هم اینکه در هر درخواست read تمامی آدرس‌هایی که یک match هستند را اعلام کند.