به نام خداوند بخشنده مهربان

طراحی سیستم های دیجیتال

دکتر اجلالی

۴۰۱۱۰۶۳۲۸ - نیکا قادری
بهار ۱۴۰۳

تاریخ گزارش: ۳۱ اردیبهشت ۱۴۰۳ | **طراحی حافظه $Tcam$ با طول پارامتریک** | سوال هفتم

فهرست مطالب

۱ معرفی و بررسی هدف تمرین

حافظه‌های $Ternary\,Content\,Addressable\,Memory\,(TCAM)$ نوع خاصی از حافظه‌های رایانه‌ای هستند که در برخی از برنامه‌های جستجو با سرعت بسیار بالا استفاده می‌شوند. این نوع حافظه کابرد های متفاوتی دارد، برای مثال در فشرده سازی و بانک داده ها و سیستم های هوشمند. تفاوت $TCAM$ های سه گانه با CAM های عادی در این است که در CAM فقط مقادیر باینری ۰ و ۱ را میتوان ذخیره کرد. اما در $TCAM$ می توان سه مقدار ۰، ۱ و x را ذخیره کرد بدین صورت که x با هر دو بیت ۰ و ۱ منطبق می شود. به همین دلیل به این نوع CAM ها، سه گانه گفته میشود. برای مثال دو داده $x11$ و $xx0$ منطبق هستند و $match$ می شوند. در ادامه ابتدا به توضیح ماژول اصلی می پردازیم، سپس کد تست بنچ را بررسی کرده، و در آخر خروجی آن را مورد بررسی قرار می دهیم.

۲ ماژول ها و توضیحات

۱.۲ comparator

ورودی و خروجی ها و مقادیر تعریف شده:

۱. a: ورودی. عدد ورودی اول. تعداد بیت های آن برابر با طول هر خانه از حافظه اصلی می باشد و $tcam$، این مقدار را پارامتری وارد می کند.

۲. b: ورودی. عدد دومی که قرار است با اولی مقایسه شود.

۳. $match$: خروجی. نشان می دهد آیا تطابقی بین دو عدد وجود دارد یا خیر.

۴. $matches$:رجیستر. مشخص می کند هر دو بیت متناظر در دو عدد با هم تطابق دارند یا خیر. اندازه آن برابر با اعداد ورودی می باشد.

```
module comparator #(
    parameter MEM_SIZE = 16,
    parameter MEM_LENGTH = 16
) (
    a,
    b,
    match
);

    input   [MEM_LENGTH-1:0]  a;
    input   [MEM_LENGTH-1:0]  b;
    output          match;

wire    [MEM_LENGTH-1:0]  matches;
```

شکل ۱: ورودی و خروجی در مقایسه کننده

```
genvar i;
generate
    for(i = 0; i < MEM_LENGTH; i = i + 1) begin
        assign matches[i] = a[i] === 1'bx || b[i] === 1'bx || ~(a[i] ^ b[i]);
    end
endgenerate

assign match = &matches;

endmodule
```

شکل ۲: بدنه $comparator$

حال بدنه اصلی این ماژول را مورد بررسی قرار می دهیم که شامل یک حلقه for می شود.
کافی است روی تمام بیت های یک خانه حلقه ای زده و اگر هر دو کاملا با هم برابر بودند یا حداقل یکی x بود - x با هر دو منطبق می شود - بیت متناظر از رجیستر $matches$ را یک کنیم.
در آخر، از بیت های $matches$، and گرفته و در خروجی قرار می دهیم. اگر دو عدد مطابق باشند، تمام بیت ها یک هستند و در نتیجه خروجی نیز یک می باشد.

tcam ۲.۲

ورودی ها:

۱. clk: سیگنال کلاک

۲. $rstN$: ورودی ریست. که اگر صفر شود مقادیر کل خانه های حافظه صفر می شوند.

۳. we: $write\,enable$، اگر یک شود داده ورودی روی آدرس مشخص شده نوشته می شود.

۴. $waddr$: آدرس خانه ای که می خواهیم روی آن بنویسیم.

۵. $data$: داده ای که می خواهیم در حافظه جستجو، یا روی حافظه نوشته شود.

۶. $search$: ورودی جستجو که با یک شدن آن، در حافظه به دنبال داده ورودی می گردیم.

خروجی ها:

۱. $saddr$: اگر جستجو موفقیت آمیز باشد، آدرس خانه پیدا شده در $saddr$ قرار می گیرد. در غیر این صورت مقدار آن ناشناخته[۱] است. (x)

۲. $sdata$: مقدار داده ی تطابق داده شده در صورتی که جستجو موفقیت آمیز باشد.

۳. $found$: سیگنالی که مشخص می کند آیا داده ورودی پیدا شده است یا مطابق با مقدار خانه ای در حافظه است یا نه.

مقادیر تعریف شده در بدنه درونی کد:

۱. mem: آرایه ای از بردارها که نقش حافظه اصلی را در ماژول دارد و داده ها روی آن ذخیره می شوند.

۲. $matches$: همواره مشخص می کند که آیا داده ورودی $(data)$ با تک تک خانه های حافظه $match$ شده است یا نه.

۳. $search\_res$: (شماره)اندیس[۲] خانه ی تطابق یافته را نگه می دارد.

۴. $addrx$: عددی به تعداد بیت های آدرس و برابر با x می سازد.

۵. $datax$: عددی برابر با x و به اندازه داده ورودی می سازد.

همانطور که در شکل ۳ پیداست، تمام ابعاد به صورت پارامتریک و بر اساس اندازه و طول حافظه تعیین شده اند. توجه شود که برای تعیین بعد آدرس، با استفاده از تابع $clog۲$ از سایز حافظه در مبنای دو لگاریتم گرفته شده است.

برای ابعاد حافظه، دو پارامتر به نام های MEM_SIZE و MEM_LENGTH تعریف شده است که اولی تعداد خانه (سطر) های موجود در کل حافظه را نمایش می دهد و دومی طول هر خانه از حافظه را بر حسب بیت نشان می دهد. همان طور که در تصویر ۴ مشاهده می شود تمام ابعاد استفاده شده در برنامه مشتقاتی از همین دو پارامتر هستند که باعث می شود ابعاد حافظه متغیر، و قابل مقداردهی باشد.

---
[۱] unknown
[۲] index

```
input                           clk;
input                           rstN;
input                           we;
input   [$clog2(MEM_SIZE)-1:0]  waddr;
input   [MEM_LENGTH-1:0]        data;
input                           search;
output  [$clog2(MEM_SIZE)-1:0]  saddr;
output  [MEM_LENGTH-1:0]        sdata;
output                          found;


reg     [MEM_LENGTH-1:0]        mem [MEM_SIZE-1:0];
wire    [MEM_SIZE:0]            matches;
reg     [$clog2(MEM_SIZE)-1:0]  search_res;
reg     [$clog2(MEM_SIZE)-1:0]  addrx;
reg     [MEM_LENGTH-1:0]        datax;
```

شکل ۳: ورودی و خروجی

```
module tcam #(
    parameter MEM_SIZE   =  16,
    parameter MEM_LENGTH =  16
) (
                clk,
                rstN,
                we,
                waddr,
                data,
                search,
                saddr,
                sdata,
                found
);
```

شکل ۴: پارامترها

در مرحله بعد، ابتدا دو رجیستر $addrx$ و $datax$ را همواره و با هر تغییراتی، برابر با x قرار می دهیم تا در هنگام $assign$ ها مشکلی برای ما ایجاد نکند. سپس خروجی ها را مقدار دهی می کنیم:

- $saddr$: اگر در حال جستجو باشیم، مستقیما برابر با $search\_res$ می باشد که اندیس خانه مپ شده را نشان می دهد. اگر هم در حال جستجو نباشیم یا جستجو موفقیت آمیز نباشد، این مقدار را با کمک رجیسترهایی که بالاتر تعریف شد برابر با x قرار می دهیم.
- $sdata$: مقدار داده ی مچ شده را نشان می دهد. در اینجا نیز اگر جستجو نداشتیم یا نتیجه ای یافت نشد برابر با x می شود.
- $found$: در صورتی که در حال جستجو باشیم - یعنی $search$ یک باشد - و همچنین آرایه $matcher$ تماما x نباشد و حداقل یک بیت ۱ داشته باشد، این بیت باید یک شود. به این معنی که عدد ورودی با حداقل یکی از اعداد موجود در حافظه مطابق است.

```
always @* begin
        addrx = {$clog2(MEM_SIZE){1'bx}};
        datax = {MEM_LENGTH{1'bx}};
    end


assign saddr = search ? search_res : addrx;
assign sdata = search ? mem[search_res] : datax;
assign found = (|matches) & search & (|matches !== 1'bx);
```

شکل ۵: $assign$ های اولیه

سپس مطابق با شکل ۶ برای هر خانه حافظه - که آرایه ای از بیت ها است - نمونه[۳]ای از ماژول $comparator$ قرار می دهیم که همواره

[۳] instance

```
genvar i;
generate
    for (i = 0; i < MEM_SIZE; i = i + 1)
        comparator #(MEM_SIZE, MEM_LENGTH) cmp(mem[i], data, matches[i]);
endgenerate
```

شکل ۶: مقایسه کننده در هر ردیف از حافظه

ورودی $data$ را با حافظه مقایسه می کند و طبق آن $matches$ را مقداردهی می کند.
در بخش آخر، مطابق با تصویر ۷ بدنه اصلی این ماژول پیاده سازی می شود. ابتدا اگر سیگنال ریست فعال باشد، تمام نقاط حافظه با صفر مقداردهی می شوند. اگر این اتفاق نیفتد، مقادیر پیش فرض همه خانه ها برابر با x می شود و هر جستجویی با خانه های حافظه مطابقت پیدا می کند. همچنین $search\_res$ را هم صفر می کنیم.
در غیر این صورت، اگر سیگنال نوشتن فعال باشد، داده ورودی را روی اندیسی از حافظه که توسط $waddr$ به ماژول داده شده است، می‌نویسیم.
در آخر، اگر به جای we، $search$ فعال باشد، ابتدا $search\_res$ را با x مقداردهی می کنیم. سپس روی تمام خانه های حافظه حرکت کرده و اندیس اولین خانه ای که $matches$ متناظر با آن یک شده بود را به عنوان جواب در $search\_res$ ذخیره می کنیم.

```
integer j, z;
always @(posedge clk or negedge rstN) begin
    if (~rstN) begin
        for (j = 0; j < MEM_SIZE; j = j + 1)
            mem[j]  = 0;
        search_res = 0;
    end
    else if (we)
        mem[waddr] = data;
    else begin
        search_res = addrx;
        if (search && found)
            for (z = 0; z < MEM_SIZE; z = z + 1)
                if (search_res === addrx && matches[z]) search_res = z;
    end
end

endmodule
```

شکل ۷: بخش آخر ماژول $tcam$

testbench ۳.۲

از آنجایی که ابعاد حافظه پارامتریک هستند، برای این ماژول سه تست بنچ جدا نوشته شده است. (البته ابعاد دیگری را نیز می توان در نظر گرفت).
در اینجا آخرین تست بنچ، یعنی $tcam_tb$۳ را مورد بررسی قرار می دهیم، که حافظه ای با اندازه ۱۰ خانه می سازد که طول هر خانه در آن ۱۷ بیت است.
ابتدا مطابق با شکل ۸ تعاریف و مقداردهی های اولیه را انجام می دهیم.

```
module tcam_TB3;

reg             clk = 1, rstN = 0;
reg             we = 0;
reg             [3:0] waddr;
reg             [16:0] data;
reg             search = 0;
wire    [3:0]   saddr;
wire    [16:0]  sdata;
wire            found;

tcam #(10, 17) TCAM3(clk, rstN, we, waddr, data, search, saddr, sdata, found);

always #5 clk = ~clk;
```

شکل ۸: مقدمه $testbench$

سپس در ۶ مرحله روی خانه های حافظه می نویسیم و در آخر این مراحل، از حافظه خروجی می گیریم تا معلوم شود چه مقادیری روی آن نوشته شود. توجه کنید که در ابتدا و قبل از مقداردهی به حافظه، سیگنال ریست برای مدتی فعال شده است.

```
initial
    begin

    #25 rstN = 1;
    #10 we = 1;

    waddr = 0;
    data = 17'b00x01010100000111;

    #10;

    waddr = 5;
    data = 17'b00x01x1110xxxx000;

    #10;

    waddr = 9;
    data = 17'b00x010x0100100100;

    #10;

    waddr = 6;
    data = 17'b11x010x010000x00x;

    #10;

    waddr = 2;
    data = 17'b00000000010000x00x;

    #10;

    waddr = 1;
    data = 17'b11111100100000000;
```

شکل ۹: نوشتن روی حافظه

```
#10;
$display(" 0:%b\n 1:%b\n 2:%b\n 3:%b\n 4:%b\n 5:%b\n 6:%b\n 7:%b\n 8:%b\n 9:%b\n10:%b\n11:%b\n12:%b\n13:%b\n14:%b\n15:%b\n16:%b\n",
    TCAM3.mem[0],
    TCAM3.mem[1],
    TCAM3.mem[2],
    TCAM3.mem[3],
    TCAM3.mem[4],
    TCAM3.mem[5],
    TCAM3.mem[6],
    TCAM3.mem[7],
    TCAM3.mem[8],
    TCAM3.mem[9],
    TCAM3.mem[10],
    TCAM3.mem[11],
    TCAM3.mem[12],
    TCAM3.mem[13],
    TCAM3.mem[14],
    TCAM3.mem[15],
    TCAM3.mem[16],
    );
```

شکل ۱۰: مشاهده حافظه کنونی

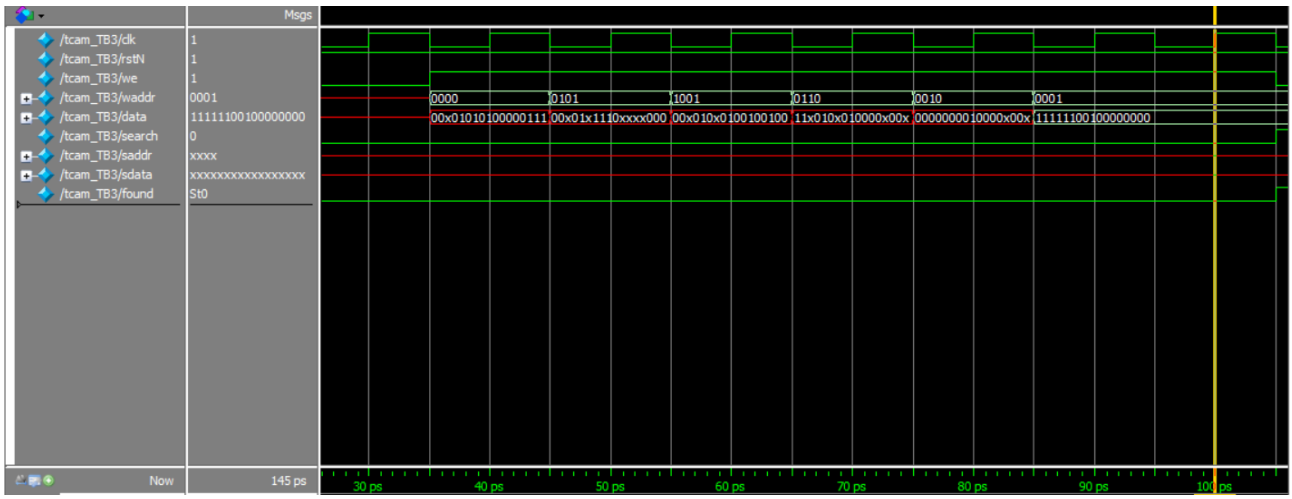
در مرحله های بعد، ابتدا سیگنال we را غیرفعال کرده، $data$ را مقداردهی می کنیم و سیگنال $search$ را فعال می کنیم. در چند مرحله، مقادیر مختلف $data$ را بررسی می کنیم. اگر داده ای با یکی از خانه های حافظه $match$ شود، سیگنال $found$ بلافاصله یک می شود و آدرس خانه پیدا شده و مقداری که $match$ شده، در اولین لبه بالا رونده بعدی کلاک، به ترتیب در $saddr$ و $sdata$ قرار می گیرند. بعد از هر مقداردهی، با استفاده از $display$ نتیجه جستجو را در ترمینال چاپ می کنیم. اگر جستجو موفقیت آمیز باشد، $found$ یک می شود و خانه حافظه ولید نمایش داده می‌شود. پیاده سازی این توضیحات در شکل ۱۱ قابل مشاهده است.

```
    #10 we = 0;

    data = 17'b11111100100000000;
    search = 1;

    #10;

    $display("finding %b: found? %d, mem[%d] = %b", data, found, saddr, sdata);

    data = 17'b1100101010xx01001;

    #10;

    $display("finding %b: found? %d, mem[%d] = %b", data, found, saddr, sdata);

    data = 17'b10x01010100000111;

    #10;

    $display("finding %b: found? %d, mem[%d] = %b", data, found, saddr, sdata);

    data = 17'b0x10xx11101010000;

    #10;

    $display("finding %b: found? %d, mem[%d] = %b", data, found, saddr, sdata);

    #10;
    $stop;
    end
endmodule
```

شکل ۱۱: $search$ در تست بنچ

۳ اجرا و شکل موج

در تصویر ۱۲ می بینیم که مقادیر مختلف بر روی حافظه نوشته می شوند. همانطور که توضیح داده شد در پایان این مرحله از حافظه در ترمینال خروجی گرفته می شود.



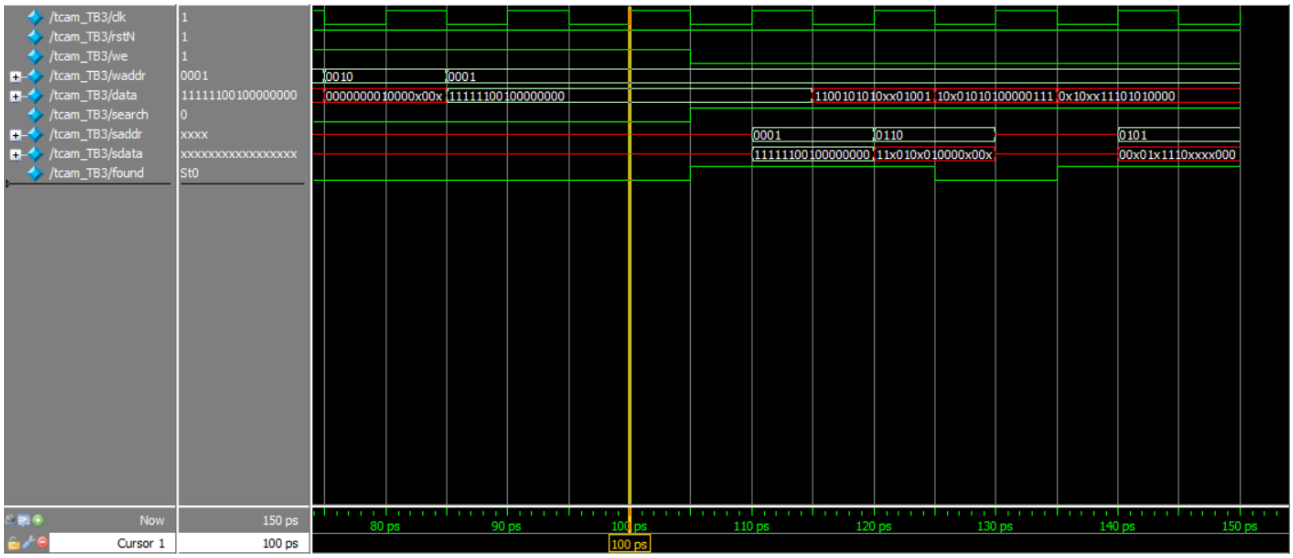
شکل ۱۲: نوشتن

شکل ۱۳ بخشی از خروجی ترمینال را نشان می دهد، که تصویری از شرایط حافظه بعد از نوشتن داده روی آن را ارائه می دهد. مشاهده می شود که خانه هایی که مقداردهی شده اند مقدار موجود در $data$ را گرفته اند و بقیه خانه ها صفر مانده اند.

```
VSIM 11> run
#   0:00x01010100000111
#   1:111111001000000000
#   2:0000000010000x00x
#   3:00000000000000000
#   4:00000000000000000
#   5:00x01x1110xxxx000
#   6:11x010x010000x00x
#   7:00000000000000000
#   8:00000000000000000
#   9:00x010x0100100100
#
```

شکل ۱۳: خروجی ترمینال پس از نوشتن روی حافظه

حال به جستجو می پردازیم.



شکل ۱۴: جستجو

ابتدا، آخرین مقداری که ذخیره کردیم را ورودی می دهیم. – $b1111111001000000000$ – که چون در ۱ ذخیره شده، سیگنال $found$ یک می‌شود و خانه ۱ و دیتای آن در خروجی قرار می گیرد.
سپس، ورودی را برابر با $b11001010100xx01001$ می‌گذاریم. اولین خانه ای که با آن $match$ می شود خانه ۶ با مقدار $b11x010x010000x00x$ می باشد، چرا که بیت های x با هر چیزی $match$ می شوند.
در ادامه، ورودی $b10x01010100000111$ داده می شود. دو بیت پرارزش این ورودی ۱۰ است اما دو بیت پرارزش مقادیر حافظه ۰۰ و ۱۱ هستند. درنتیجه این مقدار با هیچ خانه ای تطابق نمی یابد و سیگنال $found$ صفر می شود. بعد از اولین لبه بالا رونده کلاک نیز $sdata$ و $saddr$ برابر با x می شوند.
در مرحله آخر، ورودی $data$ را به $b0x10xx1110101000$ تغییر می دهیم. این بار داده ورودی با خانه ۵ با مقدار $b00x01x1110xxxx000$ تطابق می یابد و دوباره $found$ یک می شود. اطلاعات این خانه هم در $sdata$ و $saddr$ ذخیره می شود.

```
#
# finding 111111001000000000: found? 1, mem[ 1] = 111111001000000000
# finding 1100101010xx01001: found? 1, mem[ 6] = 11x010x010000x00x
# finding 10x01010100000111: found? 0, mem[ x] = xxxxxxxxxxxxxxxxx
# finding 0x10xx11101010000: found? 1, mem[ 5] = 00x01x1110xxxx000
```

شکل ۱۵: خروجی ترمینال برای سرچ های انجام شده

۴ نتیجه گیری

در این سوال، یک حافظه $tcam$ با ابعاد پارامتری ساختیم که مقادیر پیش فرض آن ۱۶ * ۱۶ می باشد اما می تواند $overwrite$ شود. سپس به شرح ماژول های درگیر و همچنین تست بنچ پرداختیم. در آخر نیز یکی از تست بنچ ها را اجرا کرده و مقادیر خروجی حافظه را بررسی کردیم.

پایان