گزارش آزمایش نهم

دانشگاه صنعتی شریف – **بهار ۱۴۰۰**

آزمایشگاه طراحی سیستم‌های دیجیتال – دکتر اجلالی

نویسندگان:
سروش جهان‌زاد – ۹۸۱۰۰۳۸۹
علی حاتمی تاجیک – ۹۸۱۰۱۳۸۵

۱ مقدمه

در مستند پیش‌رو گزارشی بر روند انجام آزمایش نهم درس آزمایشگاه طراحی سیستم‌های دیجیتال (طراحی یک واحد حافظه TCAM) ارائه شده است. کدهای مربوط به این آزمایش در کنار این فایل در پوشه مربوطه موجود است. از صبر و بردباری شما در مطالعه این گزارش سپاسگزاریم.

با احترام

سروش جهان‌زاد، علی حاتمی تاجیک – **بهار ۱۴۰۰**

۲ طرح اولیه مسئله

با توجه جست‌وجوی انجام شده[i] برای TCAM، به طور خلاصه این واحد حافظه یکی از انواع Content Addressable Memory ها است. این نوع از حافظه‌ها برعکس حافظه‌های عادی که یک آدرس گرفته و داده آن آدرس را تحویل می‌دهند، داده گرفته و آدرس آن را در صورت وجود برمی‌گرداند. تفاوت عملکرد TCAMها با CAM های عادی در این است که TCAM ها در هنگام جست‌وجو فقط یک سری از بیت‌های داده (که در هنگام نوشتار داده بر مموری مشخص شده اند) مهم هستند و عملیات Match کردن داده مورد جست‌وجو با داده‌های موجود تنها روی آن بیت‌ها صورت می‌گیرد. ذخیره اینکه کدام بیت‌ها باارزش هستند معمولا با استفاده از یک ماسک بیتی صورت می‌گیرد (یعنی به ازای هر بیت یک بیت دیگر وجود دارد که مشخص می‌کند که آیا آن بیت باارزش است یا خیر).

۳ طراحی اولیه

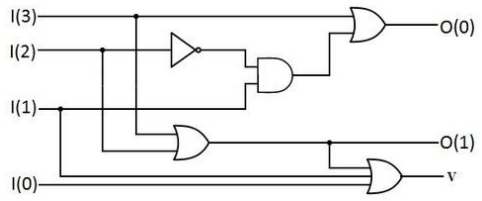
طرح اولیه که برای پیاده سازی این حافظه ریخته شد این است که بیت‌های بی‌ارزش را با ۱ و بیت‌های با ارزش را با ۰ مشخص می‌کنیم و آنها را در کنار داده‌های اصلی نگه‌داری می‌کنیم. پس مموری ما دو سری رجیستر داده و ماسک خواهد داشت. سپس با OR کردن داده با ماسک آن تمام بیت‌های بی‌اهمیت آنرا به یک تبدیل می‌کنیم تا در هنگام مقایسه موردی پیش نیاید. در یک کلاک تمام مچ های ممکن پیدا خواهند شد و در رجیستر سومی ذخیره می‌شوند که محتویات آن رجیستر با استفاده از یک Priority Encoder به آدرس نهایی تبدیل خواهد شد. اگر هم آدرس پیدا نشود (تمام بیت‌های آن رجیستر سوم ۰ باشند) بیتی که نمایانگر یافته شدن اطلاعات است برابر با صفر می‌شود.

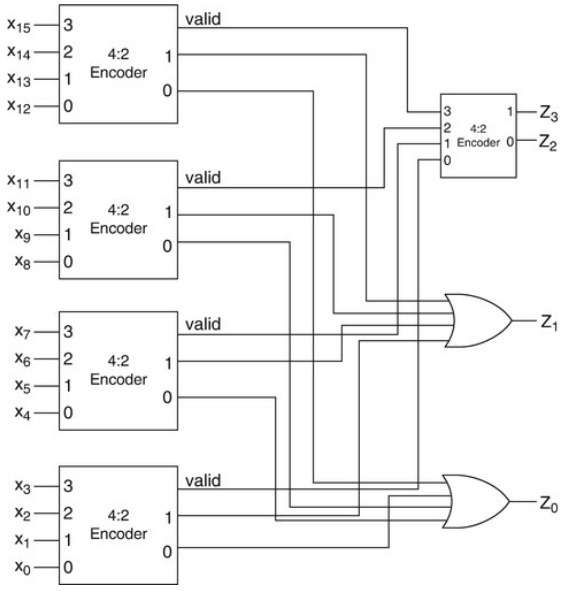
۴ پیاده‌سازی

پیاده‌سازی از دو بخش کلی تشکیل شده است: پیداکردن خانه‌های مچ شده و انکد کردن آنها.

Priority Encoder[ii]

برای ساخت یک انکودر اولویتدار، ابتدا یک انکودر اولیت‌دار ۴ به ۲ می‌سازیم و سپس با استفاده از چند واحد آن یک انکودر ۱۶ به ۴ می‌سازیم (ما ۱۶ خانه حافظه داریم که هرکدام از آنها می‌تواند مچ شده باشد یا خیر. اولین مچ یافته شده برابر با آدرسی است که مموری باید آنرا برگرداند و این آدرس یک عدد ۴ بیتی خواهد بود).





```
module pr_encoder_4_to_2 (
  D,      /* input bits/signals */
  Y,      /* Encoded Signal */
  valid   /* if is valid (there is at least one 1 bit) */
);
  input wire  [3:0] D;
  output wire [1:0] Y;
  output wire valid;

  assign Y[0] = D[3] || ((~D[2])&&D[1]);
  assign Y[1] = D[3] || D[2];
  assign valid = Y[1] || D[1] || D[0];
endmodule


module pr_encoder_16_to_4 (
  D,
  Y,
  valid
);
  input  wire [15:0] D;
  output wire [3:0] Y;
  output wire valid;

  wire valids [5:0];
  wire [1:0] results [3:0];

  pr_encoder_4_to_2 encoder1 (D[3:0], results[0], valids[0]);
  pr_encoder_4_to_2 encoder2 (D[7:4], results[1], valids[1]);
  pr_encoder_4_to_2 encoder3 (D[11:8],results[2], valids[2]);
  pr_encoder_4_to_2 encoder4 (D[15:12],results[3],valids[3]);
  pr_encoder_4_to_2 encoder5 ({valids[3],valids[2],valids[1],valids[0]},Y[3:2],);

  assign Y[1] = results[3][1] || results [2][1] || results [1][1] || results [0][1];
  assign Y[0] = results[3][0] || results [2][0] || results [1][0] || results [0][0];

  assign valid = valids [0] || valids [1] || valids [2] ||valids [3];

endmodule
```

TCAM

این واحد یک سیگنال ریست (resetN) نیاز دارد تا تمام داده‌ها را به صفر تغییر دهد. چون داده‌ها ماسک دارند و داده مهم است تا آدرس آن ممکن است داده‌های ناخواسته ای قبل از داده اصلی که در مموری ریخته شده است پیدا شوند.

این واحد یک سیگنال خواندن/نوشتن (write_readN) دارد که در هنگام نوشتن داده بر روی مموری باید یک باشد. همینطور یک سیگنال آدرس نوشتن داده نیز دارد تا بداند داده‌ای که قرار است نوشته شود کجا باید ذخیره شود (write_address)

یک داده (data) باید باشد که هم برای نوشتن داده و هم برای پیدا کردن آدرس باید از آن استفاده شود. همینطور یک ماسک (dontcare) هم باید ورودی بگیریم تا ماسک داده ای که نوشته می‌شود هم ذخیره شود و don’t care های هر داده مشخص باشند.

خروجی‌های آدرس داده (found_address) و اینکه آیا داده ای پیدا شده یا نه (found_any) نیز مورد نیاز است.

نحوه کار برای نوشتن داده مشخص است. نحوه انجام عملیات جست و جو به این صورت است که به صورت همزمان داده iام و داده ورودی را با ماسک iام OR می‌کند و برابری این دو را درون یک رجیستر result در ایندکس iام می‌ریزد. محتویات این رجستر هم به یک انکودر متصل است تا اولین آدرس داده شده و یا پیدا نشدن احتمالی را به خروجی بفرستد.

```
module tcam (

  output wire [3:0] found_address,
  output wire hit,
  input [15:0] data,
  input write_readN,
  input [15:0] dontcare_mask,
  input [3:0] write_address,
  input clk,
  input resetN,
);
  reg [15:0] mem [0:15];
  reg [15:0] mem_dontcare_masks [0:15];
  reg  [15:0] result;

  pr_encoder_16_to_4 encoder (result, found_address, hit);

  integer i;
  always @(posedge clk) begin
    for (i = 0; i < 16; i = i + 1) begin
      result[i] <= ((mem[i] | mem_dontcare_masks[i]) == (data | mem_dontcare_masks[i]));
    end
  end

  always @(posedge clk or negedge resetN) begin
    if (~resetN) begin
      for (i = 0; i < 16; i = i + 1) begin
        mem[i] <= 16'b0;
        mem_dontcare_masks[i] <= 16'b0;
      end
    end
    else begin
      if (write_readN) begin
        mem[write_address] <= data;
        mem_dontcare_masks[write_address] <= dontcare_mask;
      end
    end
  end
endmodule
```

۵ تست

برای تست این حافظه ابتدا آنرا ریست کرده و سپس دو داده با ماسک‌های متفاوت را روی دو خانه حافظه می‌نویسیم. سپس سه داده که مقادیر زیر ماسک آنها متفاوت است را امتحان می‌کنیم که تنها بیت‌های روی ماسک آنها مطابقت کند و آدرس مربوطه را برگرداند و سپس یک داده یک با هیچکدام از داده‌ها نمیخواند را امتحان می‌کنیم که باید سیگنال found را صفر کند. اینکه هر داده باید با چه چیزی مطابقت کند در بالای آن کامنت شده است.

```
module tcam_tb;

  wire [3:0] found_address;
  wire found_any;
  reg [15:0] data;
```

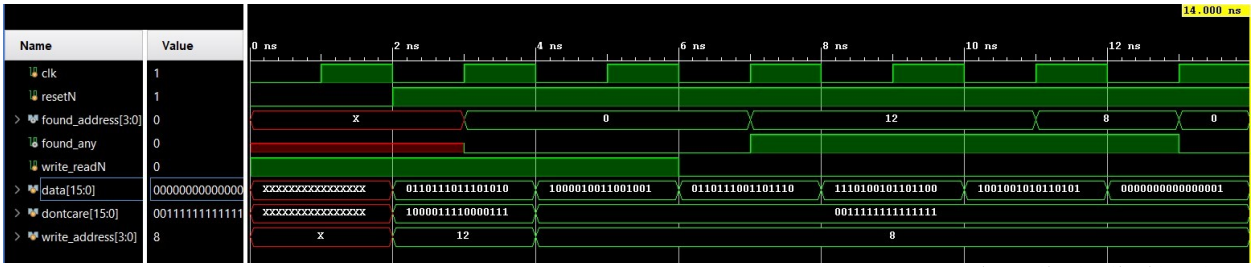
```
  reg write_readN = 1;
  reg [15:0] dontcare;
  reg [3:0] write_address;
  reg clk = 0;
  reg resetN = 0;

  tcam memory(
    found_address,
    found_any,
    data,
    write_readN,
    dontcare,
    write_address,
    clk,
    resetN);

  initial begin
    forever begin
      #1 clk = ~clk;
    end
  end

  initial begin
  # 2 resetN = 1;
  // First Entery : 12: X1101XXX_X1101XXX
  data =     16'b0_1101_110_1_1101_010;
  dontcare = 16'b1_0000_111_1_0000_111;
  write_address = 4'd12;
  # 2;
  // Second Entery : 8: 10XXXXXXXXXXXXX
  data =     16'b10_00010011001001;
  dontcare = 16'b00_11111111111111;
  write_address = 4'd8;
  #2 write_readN = 0;
  // Search for data (0110111001101110) Must be found in 12
  data = 16'b0110111001101110;
  #2
  // Search for data (11101001_01101100) Must be found in 12
  data = 16'b11101001_01101100;
  // Search for data (10_01001010110101) Must be found in 8
  #2
  data = 16'b10_01001010110101;
  // Search for data (0000000000000000) Must not be found
  #2
  data = 16'b1;
  #2 $finish;
  end
endmodule
```

نتیجه تست به صورت زیر درآمده است:



که کاملا با انتظار ما مطابقت دارد.

i Wikipedia – Content-addressable memory [link]

ii Seiffertt J. (2017) Decoders and Register Files. In: Digital Logic for Computing. Springer, Cham. https://doi.org/10.1007/978-3-319-56839-3_13 [link]