به نام خداوند بخشنده مهربان

طراحی سیستم‌های دیجیتال — پاییز ۱۴۰۰

تمرین ۳، سوال ۱ — آرین احدی نیا، ۹۸۱۰۳۸۷۸

برای حل این سوال ابتدا لازم است که به رابطه‌ای برای Increament کردن یک عدد که در gray code بیان شده است، پی ببریم. برای این منظور، جدول کارنو را برای هر یک از بیت‌ها رسم می‌کنیم و رابطه منطقی میان ورودی که خود یک عدد چهار بیتی در gray code است و بیت iام خروجی را به دست می‌آوریم.

در جدول مقابل میتوانید حالت بعد از Increament شدن هر عدد در نمایش gray code را مشاهده بفرمایید.

| AB / CD | **00** | **01** | **11** | **10** |
|---|---|---|---|---|
| **00** | 0001 | 1100 | 1101 | 0000 |
| **01** | 0011 | 0100 | 1111 | 1000 |
| **11** | 0010 | 0101 | 1110 | 1001 |
| **10** | 0110 | 0111 | 1010 | 1011 |

اگر پس از Increament حاصل $A'B'C'D'$ باشد، برای $A'$ خواهیم داشت.

$$A' = BC'D' + AC + AD$$

| AB / CD | **00** | **01** | **11** | **10** |
|---|---|---|---|---|
| **00** | 0 | 1 | 1 | 0 |
| **01** | 0 | 0 | 1 | 1 |
| **11** | 0 | 0 | 1 | 1 |
| **10** | 0 | 0 | 1 | 1 |

اگر پس از Increament حاصل $A'B'C'D'$ باشد، برای $B'$ خواهیم داشت.

$$B' = A'CD' + BC' + BD$$

| AB / CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **00** | 0 | 1 | 1 | 0 |
| **01** | 0 | 1 | 1 | 0 |
| **11** | 0 | 1 | 1 | 0 |
| **10** | 1 | 1 | 0 | 0 |

اگر پس از Increament حاصل $A'B'C'D'$ باشد، برای $C'$ خواهیم داشت.

$$C' = A'B'D + ABD + CD'$$

| AB / CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **00** | 0 | 0 | 0 | 0 |
| **01** | 1 | 0 | 1 | 0 |
| **11** | 1 | 0 | 1 | 0 |
| **10** | 1 | 1 | 1 | 1 |

اگر پس از Increament حاصل $A'B'C'D'$ باشد، برای $D'$ خواهیم داشت.

$$D' = A'B'C' + A'BC + ABC' + AB'C$$

| AB / CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **00** | 1 | 0 | 1 | 0 |
| **01** | 1 | 0 | 1 | 0 |
| **11** | 0 | 1 | 0 | 1 |
| **10** | 0 | 1 | 0 | 1 |

با توجه به روابط فوق، میتوانیم هم به صورت gray code نیز عمل Increament را انجام دهیم.

پیاده‌سازی را به این صورت انجام می‌دهیم که reset به صورت active high باشد و زمانی که تبدیل به ۱ شود، مقدار شمارنده به صفر بازگردد.

این شمارنده را به صورت rising edge triggered طراحی میکنیم. به این صورت که در کلاک بالارونده، مقدار جدید محاسبه و در خروجی قرار داده میشود. دقت بفرمایید که این در صورتی است که ورودی reset فعال نباشد.

توجه بفرمایید که مقداردهی اولیه به شمارنده با مقدار صفر انجام میشود.

```
module counter (reset, gray, clock, out);
    input reset, gray, clock;
    output reg [3:0] out;

    initial begin
        out = 4'b0000;
    end

    always @(posedge reset) begin
        out = 4'b0000;
    end

    always @(posedge clock) begin
        if (reset == 1'b0) begin
            if (gray == 1'b0) begin
                out = out + 1;
            end else begin
                out[3] <= (out[2] & ~out[1] & ~out[0]) | (out[3] & out[1]) | (out[3] & out[0]);
                out[2] <= (~out[3] & out[1] & ~out[0]) | (out[2] & ~out[1]) | (out[2] & out[0]);
                out[1] <= (~out[3] & ~out[2] & out[0]) | (out[3] & out[2] & out[0]) | (out[1] & ~out[0]);
                out[0] <= (~out[3] & out[2] & out[1]) | (out[3] & ~out[2] & out[1]) | (out[3] & out[2] & ~out[1]) |
(~out[3] & ~out[2] & ~out[1]);
            end
        end
    end
endmodule
```

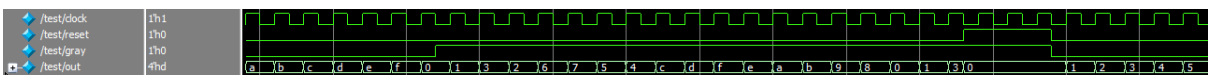
برای تست یک ماژول به نام clock generator می‌سازیم که کلاک را در مدار شبیه سازی کند. از این ماژول در سوالات دیگر نیز استفاده خواهیم کرد.

```
module clock_generator(output reg clock);
    parameter HALF_T = 1;

    initial begin
        clock = 0;
    end

    always begin
        #HALF_T clock = ~clock;
    end
endmodule
```

سپس یک نمونه از این ماژول میگیریم و آن را به ازای ورودی های مختلف تست می‌کنیم. شکل زیر بخشی از شکل موج این مدار به ازای تست نوشته شده را نشان می‌دهد.



توجه بفرمایید که عددی باینری که طولش توانی از دو باشد را می توان به شکل زیر نوشت. (طول بزرگ تر از ۳۲)

$$\begin{aligned} &a_{32n-1}a_{32n-2}a_{32n-3}\dots a_0 \\ &\quad = a_{32n-1}a_{32n-2}a_{32n-3}\dots a_{32(n-1)} \times 2^{32(n-1)} \\ &\quad + a_{32(n-1)-1}a_{32(n-1)-2}a_{32(n-1)-3}\dots a_{32(n-2)} \times 2^{32(n-2)} + \dots \\ &\quad + a_{31}a_{30}a_{29}\dots a_0 \times 2^0 \end{aligned}$$

بنابرین میتوانیم عدد را به صورت بلوک‌های ۳۲ بیتی بنویسیم.

از طرفی توجه بفرمایید که $x \times 2^k$ برابر $x \ll k$ خواهد بود.

حال اگر عدد $b$ را نیز به همین صورت بنویسیم و دو عبارت $a_{31+i}a_{30+i}a_{29+i}\dots a_i \times 2^i$ را در $a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+j}b_{30+j}b_{29+j}\dots b_j \times 2^{i+j}$ ضرب کنیم، حاصل برابر $b_{31+j}b_{30+j}b_{29+j}\dots b_j \times 2^j$ خواهد شد. بنابرین اگر همه عبارت‌های دو عدد رو نظیر به نظیر در هم ضرب کنیم، خواهیم داشت

$$\begin{aligned} a \times b &= \sum_{i=0}^{n}\sum_{j=0}^{n} a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+j}b_{30+j}b_{29+j}\dots b_j \times 2^{i+j} \\ &= \sum_{i=0}^{n}\sum_{j=0}^{n} a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+j}b_{30+j}b_{29+j}\dots b_j \ll (i+j) \end{aligned}$$

توجه بفرمایید که $a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+j}b_{30+j}b_{29+j}\dots b_j$ را می‌توانیم با ضرب کننده ۳۲ بیتی که در اختیار داریم محاسبه کنیم و با استفاده از شیفت دادن، حاصل ضرب‌های پاره‌ای را به صورت تجمعی با جواب آخر جمع کنیم. و به این صورت حاصل را به دست بیاوریم.

قطعه کد زیر دقیقا همین روند را انجام می‌دهد.

```
module multiplier (clock, start, in1, in2, ready, out);
    parameter N = 128;

    input clock, start;
    input [N-1:0] in1, in2;

    output reg ready;
    output reg [2*N-1:0] out;

    reg [15:0] i, j;

    reg [31:0] op1, op2;
    wire [63:0] res;

    DSP high_speed_mult (res, op1, op2);

    always @(posedge start) begin
        ready = 1'b0;
        out = 1'b0;
        i = 16'b0;
        j = 16'b0;
    end
```

```
    always @(posedge clock) begin
        if (start == 1'b1 && ready == 1'b0) begin
            op1 = in1 >> i;
            op2 = in2 >> j;
        end
    end

    always @(negedge clock) begin
        if (start == 1'b1 && ready == 1'b0 && res !== 64'bx) begin
            out = out + (res << (i + j));
            j = j + 32;
            if (i == N) begin
                ready = 1'b1;
            end else if (j == N) begin
                i = i + 32;
                j = 0;
            end
        end
    end
endmodule
```

در ماژول فوق، یک ضرب کننده ۳۲ بیتی داریم. با شروع عملیات و ۱ شدن سیگنال start مقدار صفر به خروجی‌ها و برخی مقادیر کمکی نسبت داده می‌شود.

در این مدار ۳۲ بیت ۳۲ بیت از دو ورودی جدا می کنیم و در لبه بالارونده در op1 و op2 قرار می‌دهیم. سپس در لبه پایین رونده حاصل را دریافت کرده و با اعمال شیفت مناسب، با خروجی جمع میکنیم. در هر مرحله نیز i و j را به عنوان متغیرهایی پیماینده پایش می‌کنیم تا در زمان اتمام، سیگنال‌های مورد نظر را فعال کنیم.

برای تست، دو عدد را به صورت رندوم تولید می‌کنیم. توجه کنید که تابع $random یک عدد تصادفی ۳۲ بیتی خروجی می‌دهد. بنابرین برای اینکه عدد تصادفی ۱۲۸ بیتی داشته باشیم، میتوانیم ۴ بار از این تابع استفاده کنیم و خروجی‌ها را به هم متصل کنیم. سپس حاصل ضرب این دو عدد را یکبار به کمک ضرب داخلی خود وریلاگ و یکبار به کمک ماژولی که نوشتیم محاسبه می‌کنیم. در صورتی که دو پاسخ یکی باشد، ماژول به ازای این دو عدد صحیح کار میکند. با تغییر seed میتوانیم اعداد مختلفی تولید کنیم و به ازای چندین ترکیب مختلف ورودی برنامه خود را تست کنیم.

```
module test;
    wire clock;

    reg start;
    reg [127:0] in1, in2;

    wire ready;
    wire [255:0] out;

    integer i;

    reg [255:0] expected;

    clock_generator clk_ins (clock);
    multiplier mult_ins (clock, start, in1, in2, ready, out);

    initial begin
        in1 = 128'b0;
        in2 = 128'b0;
        for (i = 0; i < 4; i = i + 1) begin
            in1 = in1 + ($random << (32 * i));
            in2 = in2 + ($random << (32 * i));
        end
        $display("%H", in1);
```

```
        $display("%H", in2);
        start = 1'b1;
    end

    always @(posedge ready) begin
        expected = in1 * in2;
        $display("ex:\t%H%H", expected[255:128], expected[127:0]);
        $display("ac:\t%H%H", out[255:128], out[127:0]);
    end

    initial begin
        #200 $finish;
    end
endmodule
```

به نام خداوند بخشنده مهربان

طراحی سیستم‌های دیجیتال — پاییز ۱۴۰۰

تمرین ۳، سوال ۳ — آرین احدی نیا، ۹۸۱۰۳۸۷۸

برای پیاده‌سازی رفتاری، از always @(in) استفاده میکنیم تا زمانی که ورودی in عوض شد، خروجی نیز بروزرسانی شود. توجه بفرمایید که out[i] زمانی برابر یک است که ورودی in == i باشد. بنابرین می‌توانیم به شکل زیر پیاده‌سازی را انجام دهیم.

```
module decoder2_4 (in, out0, out1, out2, out3);
    input [1:0] in;
    output reg out0, out1, out2, out3;

    always @(in) begin
        out0 = (in == 2'b00);
        out1 = (in == 2'b01);
        out2 = (in == 2'b10);
        out3 = (in == 2'b11);
    end
endmodule
```

پیاده‌سازی جریان داده‌ای نیز به طور مشابه خواهد بود. البته توجه بفرمایید که دیگر نیازی به اینکه output reg استفاده کنیم وجود ندارد.

```
module decoder2_4 (in, out0, out1, out2, out3);
    input [1:0] in;
    output out0, out1, out2, out3;

    assign out0 = (in == 2'b00);
    assign out1 = (in == 2'b01);
    assign out2 = (in == 2'b10);
    assign out3 = (in == 2'b11);
endmodule
```

در نهایت میتوانیم به شکل زیر تست بنویسیم.

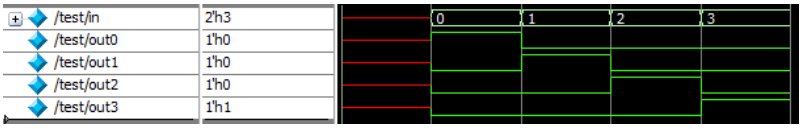
```
module test;
    reg [1:0] in;
    wire out0, out1, out2, out3;

    decoder2_4 decoder2_4_instance(in, out0, out1, out2, out3);

    initial begin
        $monitor($time, " %b %b %b %b", out0, out1, out2, out3);

        #1  in = 2'b00;
        #1  in = 2'b01;
        #1  in = 2'b10;
        #1  in = 2'b11;
        #1  $finish;
    end
endmodule
```

نتیجه شکل موج برای هر دو به شکل زیر خواهد بود.




توجه بفرمایید که نام ماژول‌ها متناسب با نوع پیاده‌سازی متفاوت است.

توجه بفرمایید که هر دو پیاده‌سازی در فایل decoder2_4 قرار دارند. یکی از آنها به صورت کامنت در آمده است.

این سوال را به شکل زیر پیاده‌سازی می‌کنیم.

```
module key_checker(clock, reset, start, in, key, ready, valid, out);
    parameter N = 32;
    parameter W = 32;
    parameter M = N * W;

    input clock, reset, start;
    input [W-1:0] in;
    input [M-1:0] key;

    output reg ready, valid;
    output reg [M-1:0] out;

    initial begin
        ready = 1'b0;
        valid = 1'b0;
        out   = 1'b0;
    end

    always @(posedge reset, posedge start) begin
        ready = 1'b0;
        valid = 1'b0;
    end

    always @(posedge clock) begin
        if (start == 1'b1 && reset == 1'b0) begin
            out = (out << W) + in;
        end
    end

    always @(negedge start) begin
        if (reset == 1'b0) begin
           valid = (out == key);
           ready = 1'b1;
        end
    end
endmodule
```

در حالت اولیه، همه مقادیر خروجی صفر خواهند بود. توجه کنید که صفر یک بیتی زمانی که به out نسبت داده میشود، از سمت چپ با صفر پر می‌شود.

هر زمان که start و یا reset برابر یک شود، مقدار ready و valid برابر صفر میشود تا با توجه به داده ورودی پردازش‌های جدید انجام شود.

همگام با لبه بالارونده کلاک، ورودی in دریافت میشود و به سمت راست چیزی که تا به حال دریافت شده، اضافه میشود. توجه بفرمایید که عرض ورودی W است. برای اضافه کردن in به سمت راست out، out را W واحد به سمت چپ شیفت میدهیم و آن را با in جمع میکنیم. البته با اپراتور concatenation نیز این کار میتوانستیم انجام دهیم. دقت بفرمایید که شرط انجام عملیات این است که start فعال و reset غیر فعال باشد. در زمان لبه پایین رونده startمی توانیم مقدار مورد نظر را محاسبه کرده و در نهایت گزارش کنیم.

برای تست، این ماژول را با N و W کوچکتر می‌سازیم و تست می‌کنیم.