بسمه تعالی

Codesign Hw2

سید محمدرضا حسینی 97243129

سید عباس میرقاسمی 97243068

۱ مقدمه

در این گزارش به طراحی و پیاده سازی یک FSMD یا Finite State Machine with Datapath برای محاسبه حاصل ضرب گفته شده در صورت سوال میپردازیم. به طور خلاصه برای دستیابی به یک Datapath مناسب ابتدا لازم است که یک CFG و در ادامه یک DFG از روی آن تولید شود. در یک DFG، data edge های روی آن به شکلی ارتباطات Datapath ما را تشکیل می دهند و بدین صورت Datapath ما به طور خاص در دسترس است؛ همچنین control edge های CFG برای واحد کنترل این Datapath استفاده می شود.

هر variable در زبان C به یک رجیستر تبدیل می شود و در این حالت نیاز به یک multiplexer برای انتخاب خروجی های variable جهت نوشته شدن در رجیستر لازم است. FSMD نیز به طور ضمنی شکلی از DFG ایجاد شده با در نظر گرفتن run ها در state های مختلف می باشد.

ابزار gezel یک ابزار کاربردی برای ملموس سازی توصیف سخت افزار و نرم افزار در کنار هم می باشد و در این گزارش به پیاده سازی FSMD ذکر شده به وسیله این ابزار می پردازیم .

۲ نحوه پیاده سازی پروژه با ابزار Gezel

در ابزار Gezel متغیر ها می توانند به صورت sig یا reg مورد استفاده قرار بگیرند که تفاوت آنها مانند تفاوت آنها در زبان HDL است .

(رجیستر ها صرفا می توانند دو مقدار داشته باشند و)... در صورت استفاده از یک Always" Statement" مقادیر داخل آن با هر کلاک سیستم تغییر می کنند؛ نکته ای که اینجا مد نظر است، عدم دسترسی مستقیم به کلاک توسط ماست و کلاک به صورت مستقل توسط سیستم تنظیم شده و تغییر می کند. always و sfg که به ترتیب SFG "بی نام" و "با نام" در این ابزار نام دارند در حقیقت مجموعه ای از دستورات ریاضی و منطقی هستند که در کلاک های مختلف سیستم اجرا می شوند و تفاوت آنها در این است که SFG های با نام بر خلاف بی نام ها صرفا در کلاک هایی که controller دستور اجرا شدن آنها را صادر کند، execute می شوند. با در کنار هم قرار گرفتن این SFG ها Datapath ما شکل می گیرد.

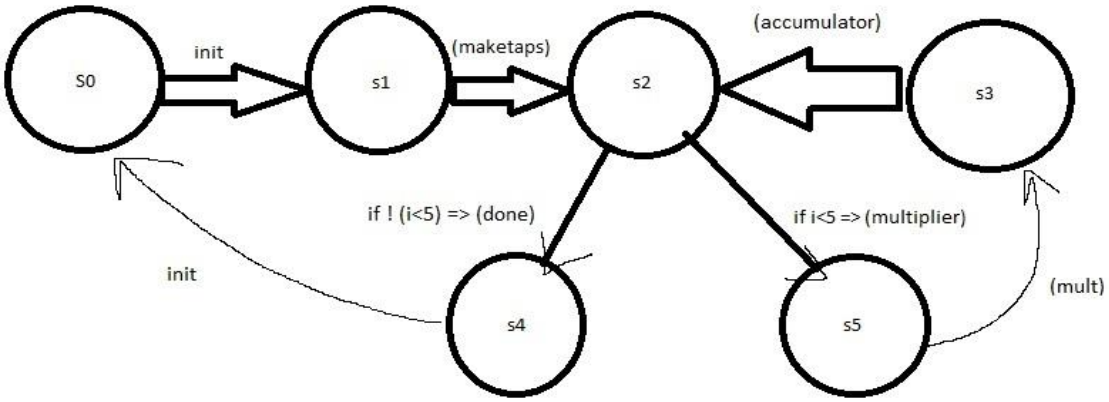
در بخش datapath filter با توجه به تمپلیت گفته شده در آخر pdf سوالات ، ورودی و خروجی رو مشخص کرده و سپس رجیستر های مورد نیاز را برای محاسبه حاصلضرب را تعریف کردیم . دلیل استفاده از نوع دیتای TC این است که در اعداد گفته شده اعداد منفی وجود دارد و استفاده از ns موجب عوض شدن حاصلضرب خواهد شد .

در ادامه 7 sfg (شامل یک sfg ، Always و شش sfg با نام استفاده میکنیم . در always مقدار r را برابر رجیستر acc قرار میدهیم تا هنگام محاسبه نهایی حاصل بر روی خروجی نشان داده شود . در sfg init مقادیر اولیه به reg ها داده میشود با توجه به اینکه در صورتی که به taps ها مقدار داده نشود ، مقدار آن صفر در نظر گرفته میشود برای اطمینان از حاصلضرب به آن ها مقدار دلخواه دادیم . در maketaps عملیات انجام گرفته در حلقه اول کد c که یک شیفت به راست است انجام میشود . در ادامه به sfg multiplier میرویم در ابتدا چون میخواهیم فقط یک ضرب کننده داشته باشیم با یک سری ternary operation و با توجه به یک رجیستر l که خاصیت یک شمارنده را دارد یکی از taps ها انتخاب میشود و به sfg mult که برای به دست آوردن

حاصلضرب taps در c است میرویم . سپس به sfg accumulator میرویم و حاصضلرب به دست آمده را در acc میریزیم . در این مرحله به sfg ، multiplier برگشته و با توجه به مقدار I ، یا با sfg done که sfg نهاییست رفته یا اینکه دوباره عملیات ذکر شده در بالا را تکرار میکنیم تا در آخر به حالت نهایی برسیم .

بخش filter_ctl نیز برای برای کنترل حالات و state های پیاده سازی شده و ترتیب sfg های گفته شده در بالا را با توجه به fsm طراحی شده ، مینویسیم . این fsm از یک initial state و 5 state دیگر تشکیل شده است که نحوه اجرای آن به صورت ترکیبی در بخش بالا ذکر شده است .

در پایان با ایجاد یک sysfilter و نوشتن یک TB برای اطمینان از صحت عمکرد سیستم کار به اتمام میرسد .



بخش DP :

```
dp filter(
        in a : tc(8);
        out r : tc(16)
        )
{
    lookup c : tc(8) = {-1, 5, 10, 5, -1};
    reg i,acc , mul :tc(8);
    reg taps0 , taps1,taps2,taps3,taps4,temp : tc(8);

    always{
        r = acc;
        // $display("clk= ",$cycle ," r= " , r , " a= " , a);
    }
```

```
always{
    r = acc;
    // $display("clk= ",$cycle ," r= " , r , " a= " , a);
}

sfg init{
    taps0 = 1;
    taps1 = 2;
    taps2 = 3;
    taps3 = 4;
    taps4 = 5;
    i = 0;
    acc = 0;
}

sfg maketaps{
    // $display("inter to maketaps");
    taps4=a;
    taps3=taps4;
    taps2=taps3;
    taps1=taps2;
    taps0=taps1;
    mul = 0;
}

sfg multiplier{
    // $display("inter to multiplier");
    temp = (i==0) ?
        taps0
        :
        i==1 ?
        taps1
        :
        i==2 ?
        taps2
        :
        i==3 ?
        taps3
        :
        taps4;
    // $display(temp);
}

sfg mult {
    mul = temp * c(i);
    i= i+1;
}

sfg accumulator{
    // $display("inter to accumulator " , "mul=" , mul , "temp= " , temp , "i= " , i);
    acc = mul + acc;
}

sfg done{
    $display("clk= ",$cycle ," result= " , r , " input= " , temp);
}
```

```
fsm filter_ctl(filter) {
    initial s0;
    state s1 ,s2 , s3 ,s4,s5;
    @s0(init) -> s1;
    @s1(maketaps) -> s2;
    @s2 if (i<5) then (multiplier)-> s5;
                 else (done)-> s4;
    @s3 (accumulator) -> s2;
    @s4 (init) -> s0;
    @s5 (mult) -> s3;
}
```

```
dp TB(
    out a:ns(8)
    )
{
    reg b:ns(8);
    always{
        a = b;
    }

    sfg run{
        b = b + 1;
    }
}

hardwired TB_ctl(TB){run;}

dp sysfilter{
    sig a : ns(8);
    sig r : ns(16);

    use TB(a);
    use filter(a,r);
}


system S {
    sysfilter;
}
```

نتیجه اجرا در gezel :

```
root@2df5f30843ea:/opt/src# fdlsim filter2.fdl 200
clk= 17 result= 4d input= 1/1
clk= 36 result= 3a input= 14/14
clk= 55 result= 27 input= 27/27
clk= 74 result= 14 input= 3a/3a
clk= 93 result= 1 input= 4d/4d
clk= 112 result= ffee input= 60/60
clk= 131 result= ffdb input= 73/73
clk= 150 result= ffc8 input= 86/86
clk= 169 result= ffb5 input= 99/99
clk= 188 result= ffa2 input= ac/ac
root@2df5f30843ea:/opt/src#
```

در ادامه با استفاده از دستور fdlvhd کد vhdl حاصل از کد gezel استخراج میشود که در کنار گزارش قرار دارند.

```
root@2df5f30843ea:/opt/src# fdlvhd -d filter2.fdl
Pre-processing System ...
Output VHDL source ...
----------------------------
Generate file: filter.vhd
Generate file: TB.vhd
Generate file: sysfilter.vhd
Generate file: system.vhd
Generate file: std_logic_arithext.vhd

debug option enabled ...
----------------------------
Generate report file: report.txt
root@2df5f30843ea:/opt/src#
```

سوال 8 : در این سوال باید در یک کلاک همه ی عملیات ها انجام شود پس به همین دلیل فقط به یک sfg بدون نام always احتیاج داریم

```
dp filter(
        in a : tc(8);
        out r : tc(16)
        )
{
    reg taps0,taps1,taps2,taps3,taps4,temp :tc(8);
```

```
    always{
        // taps0=2;
        // taps1=3;
        // taps2=4;
        // taps3=5;
        // taps4=a;

        taps4=a;
        taps3=taps4;
        taps2=taps3;
        taps1=taps2;
        taps0=taps1;

        r = -1*taps0 + 5*taps1 + 10*taps2 + 5*taps3 - taps4;
        $display("clk= ",$cycle ," result= " , r , " input= " , a);
    }
}
```

```
root@2df5f30843ea:/opt/src# fdlsim filterQ8.fdl 20
clk= 0 result= 0 input= 0
clk= 1 result= 0 input= 1
clk= 2 result= ffff input= 2
clk= 3 result= 3 input= 3
clk= 4 result= 11 input= 4
clk= 5 result= 24 input= 5
clk= 6 result= 36 input= 6
clk= 7 result= 48 input= 7
clk= 8 result= 5a input= 8
clk= 9 result= 6c input= 9
clk= 10 result= 7e input= a
clk= 11 result= 90 input= b
clk= 12 result= a2 input= c
clk= 13 result= b4 input= d
clk= 14 result= c6 input= e
clk= 15 result= d8 input= f
clk= 16 result= ea input= 10
clk= 17 result= fc input= 11
clk= 18 result= 10e input= 12
clk= 19 result= 120 input= 13
```

```
dp TB(
    out a:ns(8)
    )
{
    reg b:ns(8);
    always{
        a = b;
    }

    sfg run{
        b = b + 1;
    }
}

hardwired TB_ctl(TB){run;}

dp sysfilter{
    sig a : ns(8);
    sig r : ns(16);

    use TB(a);
    use filter(a,r);
}


system S {
    sysfilter;
}
```

تست بنچ و نتیجه اجرای این کد که برای سوال 8 است در بالا مشخص است

```
root@2df5f30843ea:/opt/src# fdlvhd -d filterQ8.fdl
Pre-processing System ...
Output VHDL source ...
----------------------------
Generate file: filter.vhd
Generate file: TB.vhd
Generate file: sysfilter.vhd
Generate file: system.vhd
Generate file: std_logic_arithext.vhd

debug option enabled ...
----------------------------
Generate report file: report.txt
root@2df5f30843ea:/opt/src#
```