گزارش تکلیف چهارم درس هم طراحی سخت افزار و نرم افزار

علیرضا مهدی برزی
حسین قنبری

۱ مقدمه

در این گزارش به طراحی و پیاده سازی یک FSMD یا Finite State Machine with Datapath برای یک عملیات تقسیم به روش ترمیمی می پردازیم. 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 Statement ”Always“ 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.

هدف ما پیاده سازی یک عملیات تقسیم به صورت ترمیمی می باشد بدین صورت که اگر داشته باشیم: $X.2^p=Q.Y + R$

برای دو مقدار تقسیم شونده X و Y، به ازای هر iteration و انجام اعمال بر روی بیت های در دست تغییر، باید عبارت بالا برقرار گردد. در صورت اتمام iteration ها و برقرار بودن عبارت بالا ما به state پایانی رسیده ایم. برای تست این فرآیند با ورودی های مشخص، طی ۱۵ سیکل از کلاک، خروجی ها را دریافت و گزارش می کنیم.

در ابتدای کد، ورودی ها و خروجی ها از جمله سیگنال های کنترلی مربوط به datapath به عنوان اسکلت اصلی کد تعریف می شوند و در ادامه رجیستر های میانی برای نگهداری دیتا های مربوط به عملیات تقسیم تعریف می شوند. علاوه بر متغیر های ورودی و خروجی اصلی از جمله x, y, start, done، متغیر هایی که ضرب هر یک از متغیر ها در یکدیگر را در یک کلاک مشخص در خود نگه داری خواهند کرد را نیز تعریف می کنیم. (شکل ۱)

در ادامه ۶ sfg (که شامل یک sfg بی نام always و پنج sfg با نام هستند) که در نهایت برای شکل گرفتن datapath کنار هم قرار می گیرند را تعریف می کنیم. این دستورات در هر کلاک یک تغییر را در بیت های مقادیر مشخص شده انجام می دهند. علاوه بر sfg های done و busy که مشخص کننده پایان یافتن یا نیافتن عملیات و رسیدن به جواب مورد نظر است، یک always sfg برای نگه داری متغیر های ایجاد شده در هر کلاک (برای از دست نرفتن متغیر های اصلی، و سه sfg شامل init, go, iteration نیز مورد تعریف قرار گرفته اند. در ابتدا یک حالت اولیه برای شروع عملیات تقسیم در نظر گرفته شده و با چک کردن برقراری رابطه مد نظر در هر کلاک با استفاده از شروط مشخص شده (0x7f = 0b1111111) این روند ادامه می یابد و با شیفت یافتن دوباره و تست برقراری معادله روند انجام کار ادامه می یابد. (شکل ۲)

سپس در بخش div_ctl حالات کنترلی مد نظر برای مشخص سازی یک state در روال حل مساله تشکیل می شوند و شروطی که برای یک transition درست به حالات دیگر لازم است مطرح می شوند. به طور کلی در این روش از حل از دو حالت کلی استفاده می شود. یک initial state برای حالت اولیه تعریف می شود که در صورت ارسال سیگنال start عملیات سه گانه برای بررسی شرایط انجام می گیرد و حالت مساله به state s1 تغییر داده می شود. (و در غیر این صورت در همان حالت باقی خواهیم ماند) در ادامه در صورت پایان یافتن iteration ها، انجام سیکل های مشخص شده از کلاک و به نتیجه رسیدن معادله کار به اتمام رسیده و به state ابتدایی خواهیم رفت و در غیر این صورت عملیات سه گانه همچنان انجام می پذیرد. (شکل ۳) همچنین State Machine در (شکل ۴) نشان داده شده است.

در پایان با ایجاد یک sysdiv و تعریف یک testbench بر روی آن ، ابتدا برای هر یک از ورودی ها مقداری مشخص تعیین کرده و برای مشخص شدن خروجی ها در کنسول در هر کلاک با استفاده از always مقادیر x, y, q, r را که متغیر های معادله اصلی هستند را با دستور $display چاپ میکنیم تا ببینیم در چه کلاکی به نتیجه مد نظر می رسیم. همچنین به مانند قبل دو حالت go و w8 با مقادیر start متفاوت برای جابجایی صحیح بین state ها و یافتن جواب مدنظر ایجاد می کنیم و در TB آنها را به مانند نمونه سوال با اعداد ۱۲ و ۱۵ مقدار دهی کرده و با دستور system S عملیات به تعداد کلاک های مدنظر انجام می پذیرد تا نتیجه نشان داده شود. (شکل ۵)

۳ گرافیک؛ شکل‌ها، جدول‌ها و نمودارها

```
dp divider(in x : ns(8);
           in y : ns(8);
           in start : ns(1);
           out q : ns(10);
           out r : ns(8);
           out done: ns(1)) {
    reg rq : ns(10);
    reg yr, rr: ns(8);
    reg cnt : ns(4);
    reg str : ns(1);

    sig z : ns(10);
    sig rb, a, b : ns(8);
    sig qb : ns(1);
```

شکل ۱. سیگنال ها و رجیستر ها

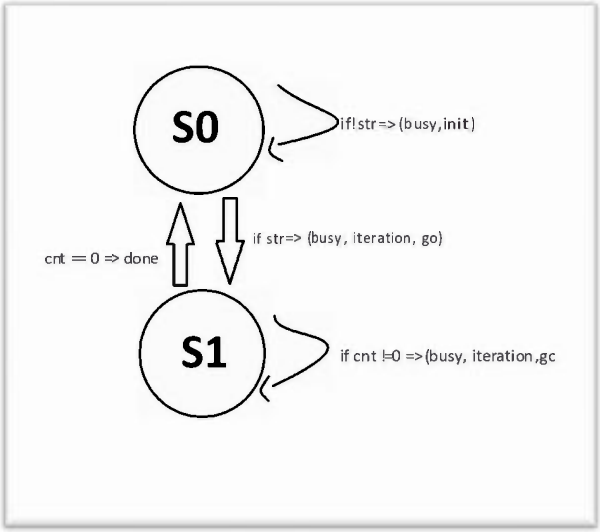
```
always { q = rq;
         r = rr;
         str = start;
         }

sfg init { rq = 0;
           rr = x;
           yr = y;
           cnt = 8;
           }

sfg go { a = rr;
         b = yr;
         rq = qb | (rq << 1);
         rr = rb;
         cnt = cnt - 1;
         }

sfg iteration { z = a*2 - b;
                qb = (z <= 0x7f) ? 1 : 0;
                rb = (z <= 0x7f) ? z : a*2;
                }

sfg done { done = 1;}

sfg busy { done = 0;}
```

شکل ۲. SFG ها

```
45  fsm div_ctl(divider) {
46      initial s0;
47      state s1;
48      @s0 if (str) then (busy, iterati0n, go) -> s1;
49          else (busy, init) -> s0;
50      @s1 if (cnt == 0) then (d0ne) -> s0;
51          else (busy, iterati0n, go) -> s1;
52  }
```

شکل ۳. کد State های کنترلی

شکل ۵. بخش TestBench



شکل ۴. State Machine

```
alireza@ubuntu:~/Desktop/Shared Folder$ fdlsim HW4.fdl 15
0  c  f  1  0  0  0
1  0  0  0  0  c  0
2  0  0  0  1  9  0
3  0  0  0  3  3  0
4  0  0  0  6  6  0
5  0  0  0  c  c  0
6  0  0  0  19  9  0
7  0  0  0  33  3  0
8  0  0  0  66  6  0
9  0  0  0  c c c  1
10  0  0  0  c c c  0
11  0  0  0  0  0  0
12  0  0  0  0  0  0
13  0  0  0  0  0  0
14  0  0  0  0  0  0
```

شکل ۶. خروجی ابزار از کد تقسیم ترمیمی در ۱۵ واحد کلاک

٤ نتیجه گیری

در نتیجه کد مد نظر را با نصب پکیج های مرتبط با ابزار Gezel در سیستم عامل لینوکس با استفاده از دستور fdlsim در کنسول و همچنین مشخص کردن مقدار کلاک های مورد نیاز برای تست به صورت مقابل وارد می کنیم: "fdlsim HW4.fdl 15"

و در نتیجه مقدار چهار متغیر معادله مذکور در ۱۵ کلاک مورد محاسبه قرار می گیرند و نشان داده می شوند (شکل ۶) همانطور که مشاهده می شود با وارد کردن این دستور، مقادیر قابل قبول معادله به عنوان خروجی هر کلاک در کنسول چاپ می شوند.

همچنین برای استخراج کد vhdl از فایل fdl می توانیم از دستور روبرو استفاده کنیم: "fdlvhd HW4.fdl"

دستور فوق در مجموع ۴ فایل مربوط به کد vhdl به ازای توابع موجود و یک فایل out. از کد اصلی تولید میکند. فایل های استخراج شده به همراه گزارش ضمیمه شده اند.