طراحی سیستم های مبتنی بر ASIC/FPGA

تمرین سری ۱

علی یداللهی

شماره دانشجویی : ۴۰۰۱۰۲۲۳۳

۱

الف

- FPGA تراشه‌های نیمه‌رسانایی هستند که از تعداد زیادی جزء کوچک الکترونیکی به نام بلوک منطقی logic block ساخته شده‌اند. در این حالت، تراشه یک ساختار خام دارد و شما می‌توانید ساختار و معماری و نحوه ارتباطات بین گیت‌های منطقی را خودتان تعریف کنید. نتیجه این تمایز این می‌شود که FPGA یک برد از پیش آماده نیست، بلکه با انتخاب و طراحی کاربر، می‌تواند مثل یک مدار الکترونیکی ساده یا یک واحد پردازش سیگنال و یا حتی مثل یک CPU عمل کند. از طرفی FPGA قابلیت برنامه‌ریزی مجدد را نیز دارد که دست طراح را برای انجام تغییرات باز می‌گذارد.
- یک مدار مجتمع با کاربرد خاص و یا (ASIC)، مدار مجتمعی است که به منظور انجام عملیات خاصی طراحی می‌گردد. ASIC مثل FPGA تک‌منظوره است با این تفاوت که پس از ساخت، دیگر قابل تغییر نیست و همیشه فقط همان عملکرد را دارد.
- Processor ها برای اجرای دستورات کلی طراحی شده اند و قابل برنامه ریزی و انعطاف پذیر هستند.
  processor ها تراشه‌های چندمنظوره (General Purpose) هستند. بنابراین سیستم‌عامل می‌تواند به نحوه‌ای که ترجیح می‌دهد آن را کنترل کند و برنامه‌های مختلف می‌توانند به گونه‌ای که نیاز دارند از آن استفاده کنند. کاری که CPU انجام می‌دهد در زمان‌های مختلف متفاوت است و حتی فرکانس کاری (که سرعت آن را تعیین می‌کند) و میزان استفاده از آن (Utilization) همواره در حال تغییر است. در سوی مقابل FPGA همیشه یک عمل ثابت را انجام می‌دهد و در صورتی که بخواهید عملکرد آن را تغییر دهید باید طراحی گیت‌های آن اصلاح شود.

مقایسه

- توان مصرفی : FPGA ها توان بیشتری نسبت به ASIC مصرف می کنند اما در مقایسه با processor ها از لحاظ مصرف توان بهینه تر هستند.
- هزینه : FPAG ها از نظر هزینه برای prototyping و تولید به تعداد پایین مناسب هستند اما با افزایش تولید به مقدار زیاد هزینه تولید افزایش زیادی خواهد داشت.
  هزینه اولیه برای اولید ASIC ها زیاد است. اما هزینه تولید به ازای هر تراشه نسبت به FPGA ها کمتر است. بنابراین در تولید به تعداد بالا ASIC از نظر هزینه مناسب تر است.
  پردازنده ها برای کاربردهای کلی طراحی شده اند و در این زمینه از نظر هزینه مناسب تر هستند اما برای کاربردهای اختصاصی تر مناسب نیستند.
- زمان طراحی : به طور کلی زمان لازم برای ASIC ها از همه بیشتر و برای FPGA از همه کمتر است.
- Verification : به طور کلی در FPGA ها مدت زمان Verification کوتاه تر است چون در FPGA ها به دلیل داشتن ساختار برنامه پذیر می توان پیکربندی های مختلف را تست کرد. اما در ASIC ها به خاطر ساختار تغییرناپذیر

فرایند Verification طولانی تر است.
در پروسسورها فرایند Verification از ASIC ها کوتاه تر است.

- موارد استفاده : FPGA ها برای prototyping و کاربردهایی که نیاز به انعطاف پذیری دارند مناسب هستند. ASIC ها برای محصولات با تولید زیاد و انجام یک کار به خصوص به صورت بهینه و مصرف کمتر توان مناسب هستند. پردازنده ها برای کارهای پردازشی کلی از تلفن های همراه گرفته تا کامپیوترها استفاده می شوند.

ب

در حوزه مخابرات FPGA ها عملکرد مناسب تری دارند همچنین چون تولید گیرنده مخابراتی تنها ۲ عدد است استفاده از FPGA مناسب تر است.
تولید تراشه محاسبات ریاضی نسبتا زیاد و برابر با ۲۰۰۰ عدد است بنابراین بهتر است از ASIC استفاده کنیم.

ج

چون برای طراحی ASIC زمان زیادی مورد نیاز است برای کاهش time to market بهتر است در هر دو مورد از FPGA استفاده کنیم.

د

مقایسه FPGA و CPLD :

نخستين تفاوت بين اين دو تراشه در نوع حافظه مورد استفاده آنها مي باشد. تراشه هاي FPGA از حافظه RAM استفاده مي كنند به اين ترتيب با قطع ولتاژتغذيه، نياز به پيكربندي مجدد دارند در حالي كه تراشه هاي CPLD از حافظه Flash يا EEPROM استفاده مي كنند و با قطع تغذيه برنامه ذخيره شده در آنها از بين نمي رود. تفاوت ديگر در حجم و تنوع بلوكهاي در دسترس مي باشد به طور كلي قابليت هاي تراشه FPGA بسيار بيشتر از تراشه اي CPLD مي باشد به عنوان مثال بلوكهاي RAM ،ضرب كننده ها، بلوكهاي DSP ، مدارهاي سنكرون سازي كلاک و بسياري از بلوكها و قابليتهاي متنوع ديگر در معماري FPGA ها در دسترس مي باشد همچنين استاندارهاي ولتاژي و جرياني متعدد توسط اين تراشه ها پشتيباني مي شود در حالي كه CPLD ها از قابليتهاي محدودتري برخوردارند و حجم گينها و تعداد فليپ فلاپها در مقايسه با FPGA ها كمتر مي باشد. مثلا CPLD دارای حدود ۲۰۰۰ گیت می باشد این در حالی است که FPGA گیتهای بسیار زیادتری دارد و یک FPGA معمولی دارای حدود ۱۰۰۰۰۰ گیت منطقی می باشد. از سوي ديگر به دليل سادگي معماري، ميزان تاخير سيگنال در تراشه اي CPLD قابل پيش بيني است اما تاخير در FPGA به نوع برقراري اتصالات (Routing) وابسته بوده، قابل پيش بيني نمي باشد. همچنين مصرف جريان در تراشه هاي CPLD به مراتب كمتر از FPGA مي باشد.
نمونه کاربرد CPLD: انجام کارهای آزمایشگاهی و تجهیزات صنعتی ساده؛ مانند: برد کنترل کرکره برقی
نمونه کاربرد FPGA: انجام محاسبات و پردازش صوت و تصویر، امورمخابراتی؛ مانند: طراحی سویچ (شبکه های کامپیوتری)
Gate Array: ها یک basic logic gate هستند یک و بر خلاف FGPA و CPLD قابل برنامه ریزی نیستند و چیزی در بین انعطاف پذیری FPGA ها و بازده CPLD ها هستند و در کاربردهایی که توان مصرفی قابل توجه است استفاده می شوند.

ه

منابع سخت افزاری FPGA :

۱. Logic Blocks: از گیت های مختلف تشکیل شده اند و از آنها برای پیاده سازی توابع منطقی استفاده می شود.

۲. سیم هایی که قسمت های مختلف از Logic Blocks ها را به هم متصل می کنند.

۳. I/O Blocks:
وظیفه این بلوک ها تبادل اطلاعات و برقراری ارتباط با محیط خارج است.

۴. DSB Blocks: این بلوک ها برای تجزیه و تحلیل سیگنال ها استفاده می شوند. RAM Blocks: برای ذخیره سازی اطلاعات استفاده می شوند.

۵. Configuration Memory: رفتار FPGA را تعیین می کند.

۲

الف

در طراحی همروند کارها به ترتیب و پشت سر هم انجام می شوند به طوری که پس از اتمام یکی دیگری انجام خواهد شد اما در طراحی موازی امکان انجام چند کار به صورت موازی وجود دارد. برخی مزایای طراحی موازی در ادامه آورده شده است:

- افزایش Throughput
- کاهش تاخیر
- می توانیم از قسمت های بیشتری از FPGA استفاده کنیم بنابراین بهره وری افزایش پیدا می کند.
- می توانیم معماری های موازی متناسب با کاربرد مورد نظر خود طراحی کنیم که باعث افزایش بهره وری می شود.
- موازی سازی برای اجرای الگوریتم های پردازش تصویر،هوش مصنوعی،یادگیری ماشین و رمزنگاری مناسب تر است.

ب

سطوح مختلف موازی سازی:

- Hardware Level: در پایین سطح در داخل معماری FPGA نوعی موازی سازی فراهم شده است.
- Task Level: وظایغ محاسباتی به وظایف کوچکتر تقسیم می شوند و این وظایف کوچکتر به صورت موازی در FPGA پیاده سازی و اجرا می شوند که باعث بهبود عملکرد مدار خواهد شد.
- Data Level: شامل پردازش چندین داده به صورت همزمان است. مثلا پردازش موازی جریان های داده و اجرای عملیات برروی مجموعه داده ها به صورت موازی
- Instruction Level: به معنی اجرای چندین دستورعمل به صورت موازی است.
- Pipeline: یک کار به جند مرحله تقسیم می شود و این مراحل به صورت موازی در یک خط لوله(pipeline) اجرا می شوند.

ج

تفاوت‌ها بین FPGA و GPU :

- مصرف انرژی: به طور کلی FPGAها نسبت به GPU ها از نظر مصرف انرژی بهینه ترهستند که این امر به خاطر قابلیت پیکربندی آن‌ها است که اجازه می‌دهد بر اساس نیازهای خاص برنامه، مصرف انرژی بهینه شود.GPU ها برای پردازش موازی با عملکرد بالا طراحی شده‌اند که موجب مصرف بیشتری نسبت به FPGA ها می‌شود.

- time to market: GPUها برای رندرینگ گرافیک و محاسبه موازی بهینه شده‌اند، اما ممکن است برای بهره‌برداری کامل از ظرفیت آن‌ها برای وظایف غیر گرافیکی نیاز به تلاش اضافی داشته باشد که این موضوع ممکن است منجر به دوره‌های توسعه بلندتری شود. درحالی که FPGA ها به دلیل پروتوتایپ سازی سریع و قابل برنامه‌ریزی مجدد، که اجازه می‌دهد تا ادامه طراحی سریع‌تر صورت گیرد؛زمان سریع‌تری را برای ورود به بازار فراهم می‌کنند.
- پیچیدگی طراحی: FPGA ها در طراحی سخت افزار انعطاف پذیری دارند اما ممکن است برای عملکرد کاملا بهینه و استفاده مناسب از منابع نیاز به طراحی ویژه ای داشته باشند.
GPU ها اصلی برای پردازش موازی و رندرینگ گرافیکی طراحی شده‌اند که ممکن است نیاز به یک رویکرد متفاوت برای استفاده بهینه از آن‌ها برای وظایف محاسباتی غیر گرافیکی داشته باشند.
- کاربردها:
FPGAها برای پروتوتایپ سریع، پردازش سیگنال، رمزنگاری، شبکه‌ها، و برنامه‌های اینترنت اشیاء مناسب هستند.
GPUها در وظایف پردازش موازی مانند رندرینگ گرافیک، یادگیری ماشین، شبیه سازی‌های علمی، یادگیری عمیق، هوش مصنوعی، و محاسبات با عملکرد بالا عموماً موفق هستند.

۳

الف

زبان Verilog قابلیت طراحی یک ماژول در چندین سبک کدنویسی را دارد. بسته به نیازهای یک طراحی، می توان را می توان از چهار سطح abstraction استفاده کرد. صرف نظر از سطح abstraction داخلی، ماژول دقیقاً به روشی مشابه با محیط خارجی رفتار می کند. در ادامه چهار سطح مختلف abstraction آورده شده است که با چهار سبک کدگذاری مختلف زبان Verilog قابل توصیف است:

- Behavioral level
- Dataflow level
- Gate level
- Switch level

ترتیب ذکر شده در بالا از بالاترین تا پایین ترین سطح abstraction است.

۱. Behavioral level:

- این بالاترین سطح abstraction ارائه شده توسط Verilog HDL است.
- یک ماژول را می توان بر اساس الگوریتم طراحی مورد نظر بدون نگرانی برای جزئیات پیاده سازی سخت افزار پیاده سازی کرد.
- مدار را بر حسب رفتار مورد انتظارش مشخص می کند.
- این نزدیکترین توصیف به زبان طبیعی از عملکرد مدار است، اما ترکیب آن نیز دشوارترین است.

مدل رفتاری $Mux4:1$:

```
module Mux_4to1(

input [3:0] i,
```

```
4           input [1:0] s,
5           output reg o
6           );
7
8           always @(s or i)
9
10          begin
11
12          case (s)
13
14          2'b00 : o = i[0];
15          2'b01 : o = i[1];
16          2'b10 : o = i[2];
17          2'b11 : o = i[3];
18          default : o = 1'bx;
19
20          endcase
21          end
22          endmodule
23
```

۲. Dataflow level:

- در این سطح، ماژول با مشخص کردن جریان داده طراحی می شود.
- با نگاهی به این طرح، می توان متوجه شد که چگونه داده ها بین ثبات های سخت افزاری جریان می یابد و چگونه داده ها در طراحی پردازش می شوند.
- این سبک شبیه معادلات منطقی است. مشخصات شامل عباراتی است که از سیگنال های ورودی تشکیل شده و به خروجی ها اختصاص داده شده است.
- در بیشتر موارد، چنین رویکردی را می توان به راحتی به یک ساختار ترجمه کرد و سپس اجرا کرد.

مدل جریان داده $Mux4:1$:

```
1           module Mux_4to1_df(
2           input [3:0] i,
3           input [1:0] s,
4           output o
5           );
6
7           assign o = (~s[1] & ~s[0] & i[0]) | (~s[1] & s[0] & i[1]) | (s[1] & ~s[0] & i[2]) | (
    s[1] & s[0] & i[3]);
8
9           endmodule
10
```

در این رویکرد از عبارت "assign" استفاده می شود. دستور Assign یک عبارت پیوسته است که در هر تغییر در سیگنال های سمت راست، سیگنال خروجی به روز می شود. هر گونه تغییر در سیگنال های ورودی دستور assign را اجرا می کند و مقدار به روز شده در خروجی "o" منعکس می شود. تغییرات در ورودی ها به طور مداوم نظارت می شود.

۳. Gate level:

- ماژول از نظر گیت های منطقی و اتصالات بین این گیت ها پیاده سازی شده است.
- شبیه یک نقشه شماتیک با اجزای متصل به سیگنال است.
- تغییر در مقدار هر سیگنال ورودی یک جزء، مؤلفه را فعال می کند. اگر دو یا چند کامپوننت به طور همزمان فعال شوند، اقدامات خود را نیز همزمان انجام خواهند داد.
- بازنمایی سیستم ساختاری به اجرای فیزیکی نزدیکتر است تا رفتاری، اما به دلیل تعداد زیاد جزئیات بیشتر درگیر است. از آنجایی که گیت منطقی محبوب‌ترین مؤلفه است، Verilog مجموعه‌ای از گیت‌های منطقی از پیش تعریف‌شده دارد که به عنوان اولیه شناخته می‌شوند. هر مدار دیجیتالی را می توان از این موارد اولیه ساخت.

مدل سطح گیت $Mux4:1$:

```
module Mux_4to1_gate(
input [3:0] i,
input [1:0] s,
output o
);

wire NS0, NS1;
wire Y0, Y1, Y2, Y3;
not N1(NS0, s[0]);
not N2(NS1, s[1]);
and A1(Y0, i[0], NS1, NS0);
and A2(Y1, i[1], NS1, s[0]);
and A3(Y2, i[2], s[1], NS0);
and A4(Y3, i[3], s[1], s[0]);
or O1(o, Y0, Y1, Y2, Y3);
endmodule

```

Switch level: سطح سوئیچ مدل سازی سطحی بین سطوح منطقی و ترانزیستوری آنالوگفراهم می کند. این ارتباط گیت ها و ارتباطات آنها را با کمک ترانزیستورها توصیف می کند.ترانزیستورها در این سطح به صورت روشن یا خاموش، رسانا یا نارسانا مدل‌سازی می‌شوند. طراحان معمولا از این سطح زبان به دلیل پیچیدگی و زمانبر بودن طراحی مدارها به ویژه مدارهای بزرگ استفاده نمی کنند. چند مثال از syntax :

```
// instantiate a nmos switch
nmos n1(out, data, control);
// instantiate a pmos switch
pmos p1(out, data, control);
//instantiate cmos gate
cmos c1(out, data, ncontrol, pcontrol);

```

ب

بالاترین سطح abstraction ارائه شده توسط Behavioral level Verilog است. بنابراین با استفاده از این سطح می توان در هفته تعداد بیشتری گیت پیاده سازی کرد.

۴

الف

به طور کلی سنتز در پیاده‌سازی، تبدیل از یک توصیف سخت‌افزاری سطح بالا به توصیف سطح پایین‌تر است. سنتز در FPGAها، پیاده‌سازی مدار توصیف شده به زبان، HDL به کمک منابع دیجیتالی موجود در FPGA است.
ابزارهای سنتز فرایند سنتز را برای ما انجام می دهند.
کارهایی که این ابزار ها انجام می دهند به صورت زیر است:

- این ابزارها ساختارهای سطح بالا را به اجزای سطح پایین تر مانند گیت ها تبدیل می کنند.
- مدار مورد نظر را برای هماهنگی با محدودیت های طراحی بهینه سازی می کنند.
- بررسی می کنند که آیا مدار سنتز شده محدودیت های زمانی و فرکانسی را برآورده می کند.
- صحت مدار سنتز شده را بررسی می کند.

ب

- جانمایی یا Placement : در این مرحله مشخص می‌کنیم منابعی که در مرحله‌ی سنتز برای پیاده‌سازی طرح استفاده شده است، دقیقا در کجای FPGA قرار می‌گیرند.
  فرض کنیم در FPGAی که استفاده می‌کنیم LUT ۴۰۰۰ برای پیاده‌سازی مدار قرار داده شده است.

  در مرحله‌ی جانمایی مشخص می‌شود که از میان این LUT ۴۰۰۰، کدام دو LUT برای پیاده‌سازی جمع‌کننده استفاده می‌شود.

- مسیریابی یا Routing :

در FPGAها تعداد زیادی سیم وجود دارد که بایدبا اتصال مناسب این سیم‌ها به LUTها یا به یکدیگر، مدار موردنظرتان را تکمیل کنیم. اینکه دقیقا از چه مسیری این هاLUT را به هم متصل کنیم به الگوریتم‌های Routing و مسیریابی موجود در نرم‌افزارهای پیاده‌ساز و معیارهایی که برای این مسئله وجود دارد، مربوط می‌شود.

یکی از این معیارها، انتخاب کوتاه‌ترین مسیر بین منابع دیجیتالی است.
محدودیت ها :

۱. محدودیت زمانی: فرآیند place و root نقش مهمی در برآوردن محدودیت‌های زمان‌بندی ایفا می‌کند.این محدودیت ها شامل فرکانس کلاک، hold time، setup time و بیشینه تاخیر مسیر است. ابزارهای سنتز مدار با بهینه سازی فرآیند lrplace و root این محدودیت ها را برآورده می کنند.

۲. محدودیت فرکانسی: بهینه سازی lrplacing و rooting، critical path و ماکزیمم فرکانس کلاک قابل دستیابی را تحت تاثیر قرار می دهد و با انجام بهینه سازی می توان به فرکانس های مطلوب تر رسید.

ج

عملکرد حلقه‌ها:

زبان‌های برنامه‌نویسی:

عملکرد: در زبان‌های برنامه‌نویسی مانند C، Python یا Java، حلقه‌ها راهی برای اجرای یک بلوک کد تکراری بر اساس یک شرط یا تعداد تکرارهای مشخص فراهم می‌کنند. آن‌ها برای حلقه زدن روی ساختارهای داده، انجام محاسبات و کنترل جریان اجرا استفاده می‌شوند.

زبان‌های طراحی سخت‌افزار:
عملکرد: در زبان‌های طراحی سخت‌افزار مانند Verilog یا VHDL، حلقه‌ها به طور مفهومی هدف مشابهی با زبان‌های برنامه‌نویسی دارند؛ با اجازه اعمال تکراری، طراحان می‌توانند عملیات موازی یا متوالی را در ارتباط با عناصر سخت‌افزاری در مدار دیجیتال انجام دهند.
پیاده‌سازی حلقه‌ها:
زبان‌های برنامه‌نویسی:
پیاده‌سازی: در زبان‌های برنامه‌نویسی نرم‌افزاری، حلقه‌ها با استفاده از سازوکارهای مانند for، while پیاده‌سازی می‌شوند. این‌ها بر روی یک پردازنده متوالی اجرا می‌شوند و کنترل جریان اجرا بر اساس شرایط یا تعداد تکرارهای تعریف شده در ساختار حلقه است.
زبان‌های طراحی سخت‌افزار:
پیاده‌سازی: در زبان‌های طراحی سخت‌افزار، حلقه‌ها با استفاده از سازه‌های مانند حلقه‌های for یا ماژول‌های بازگشتی پیاده‌سازی می‌شوند. این حلقه‌ها در طول فرآیند سنتز به منطق سخت‌افزار تبدیل می‌شوند و همزمان یا موازی در منابع منطقی قابل پیکربندی FPGA یا ASIC اجرا می‌شوند، بسته به روش طراحی خاص سخت‌افزار.
تفاوت‌های در اجرا:
زبان‌های برنامه‌نویسی:
در برنامه‌نویسی نرم‌افزار، حلقه‌ها به ترتیب بر روی پردازنده اجرا می‌شوند، به صورتی که هر تکرار به صورت خطی بر اساس شرایط یا معیارهای تعریف شده در ساختار حلقه پردازش می‌شود.
زبان‌های طراحی سخت‌افزار:
در طراحی سخت‌افزار، حلقه‌ها به ساختمان‌های سخت‌افزار موازی تبدیل می‌شوند و اجازه اجرای هم‌زمان چندین تکرار حلقه را در منابع منطقی قابل پیکربندی FPGA فراهم می‌کنند.

د

- LUT-Based Logic:

منطق مبتنی بر LUT در FPGA ز جداول جستجو استفاده می‌کند که واحدهای حافظه قابل پیکربندی هستند و بر اساس ترکیب‌هایورودی، مقادیر خروجی را ذخیره می‌کنند. این LUT ها به عنوان اجزای اساسی برای پیاده‌سازی توابع منطقی در طراحی‌های FPGA عمل می‌کنند.
پیاده‌سازی: ترکیب‌های ورودی به LUT مقدار خروجی را تعیین می‌کنند، که طراحان قادرند با برنامه‌ریزی LUT ها هر تابع بولی را پیاده‌سازی کنند.
انعطاف‌پذیری: منطق مبتنی بر LUT انعطاف پذیری بالایی را در پیاده‌سازی توابع منطقی پیچیده با بهینه‌سازی جداول حقیقت یا معادلات در تنظیمات LUT ارایه می‌دهد.
کاربرد: منطق مبتنی بر LUT معمولاً برای توابع منطقی چندمنظوره، منطق ترکیبی، عملیات حساب، و پیاده‌سازی توابع منطقی پیچیده در طرح‌های FPGA استفاده می‌شود.

- MUX-Based Logic

منطق مبتنی بر MUX در FPGA از مالتی‌پلکسرها استفاده می‌کند که یکی از سیگنال‌های ورودی چندین را انتخاب کرده و به عنوان خروجی بر اساس مقادیر خطهای انتخابی ارایه می‌دهد
پیاده‌سازی: مالتی‌پلکسرها می‌توانند به صورت پی‌درپی یا توالی اتصال یافته و ترکیب‌های پیچیده تر یا مسیرهای سیگنال را تشکیل دهند، که مسیردهی داده و انتخاب در طرح‌های FPGA را فراهم می‌کند.

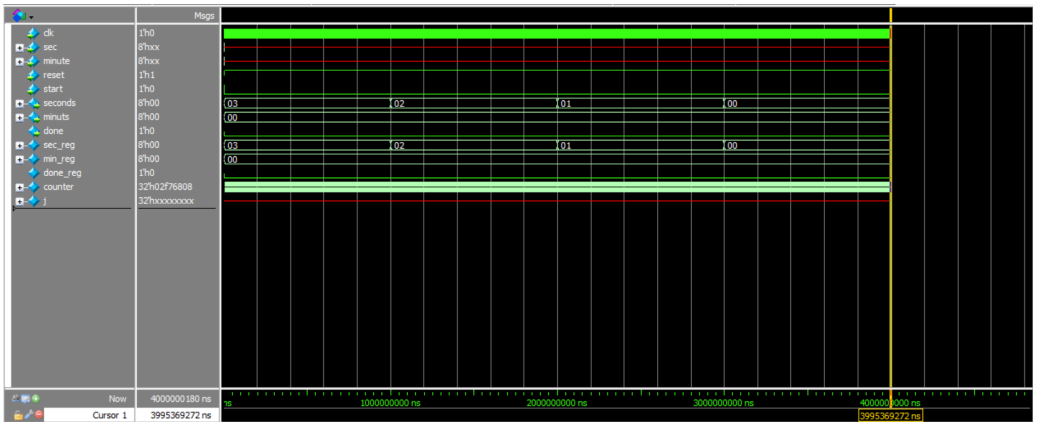
کارایی: منطق مبتنی بر MUX امکان استفاده بهینه از منابع را با انتخاب ورودی‌های مختلف به صورت پویا بر اساس سیگنال‌ها یا شرایط فراهم می‌کند.
کاربرد: منطق مبتنی بر MUX معمولاً برای مالتی‌پلکسینگ سیگنال، مسیردهی داده، انتخاب بین عملیات‌ها یا مسیرهای داده مختلف، و پیاده‌سازی ساختارهای حافظه مانند RAM ها یا ROM ها استفاده می‌شود.

۵

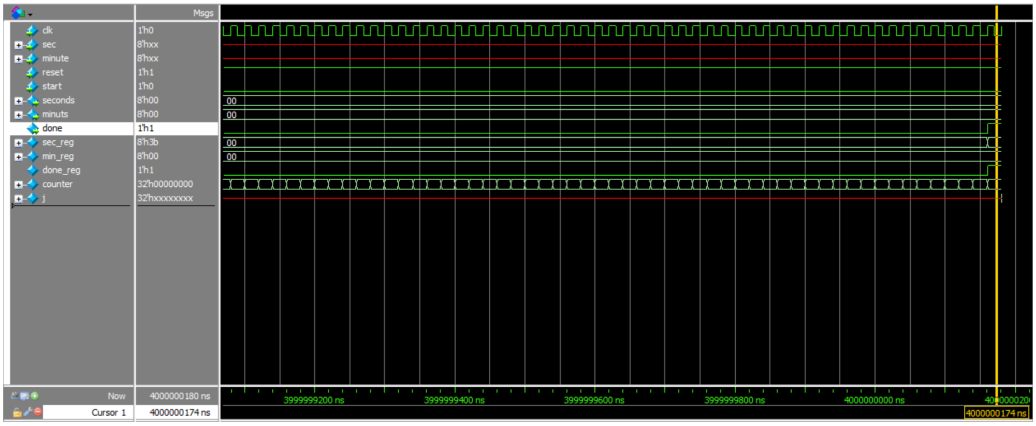
فرکانس کلاک مدار برابر ۵۰ مگاهرتز است بنابراین در هر ثانیه کلاک ۵۰ میلیون پریود خود را سپری می کند.
ماژول ما مقدار دقیقه و ثانیه ورودی ، سیگنال start و reset و کلاک را می گیرد و دقیقه و ثانیه و سیگنال done را به خروجی می دهد. یک متغیر داخلی به نام counter در کد تعریف می کنیم که هر بار از ۰ تا ۵۰ میلیون می شمارد که رسیدن آن به ۵۰ میلیون نشان دهنده گذشت یک ثانیه است. سپس مقدار ثانیه یکی کم شده و counter ریست می شود و فرآیند ادامه می یابد. اگر مقدار ثانیه برابر صفر باشد ولی دقیقه هنوز صفر نشده باشد مقدار ثانیه برابر ۵۹ می شود و یکی از دقیقه کم می شود. در انتها هم پس از اتمام شمارش مقدار done یک می شود و فرآیند به اتمام می رسد.
در تست بنچ ابتدا start برابر یک تعریف می شود تا ورودی ها در متغیرهای داخلی ماژول ذخیره شوند، سپس در سیکل بعدی start صفر شده تا ورودی جدیدی لود نشود و فرآیند ادامه پیدا خواهد کرد.
کد ماژول طراحی شده و خروجی در waveform نرم افزار modelsim به ازای ورودی ۳ ثانیه آورده شده اند:

```
1   `timescale 1ns/1ns
2   module model
3   (
4   input clk,
5   input [7:0] sec,
6   input [7:0] minute,
7   input reset,
8   input start,
9   output reg [7:0] seconds,
10  output reg [7:0] minuts,
11  output  done
12  );
13
14  reg [7:0] sec_reg;
15  reg [7:0] min_reg;
16  reg done_reg;
17
18  integer counter = 0;
19  integer j;
20
21
22  assign done = done_reg;
23  always @(posedge clk)
24
25  if(start) begin
26  min_reg <= minute;
27  sec_reg <= sec;
28  done_reg <= 1'b0;
29  end
30  else if(!reset)begin
31  min_reg <= minute;
32  sec_reg <= sec;
33  end
```

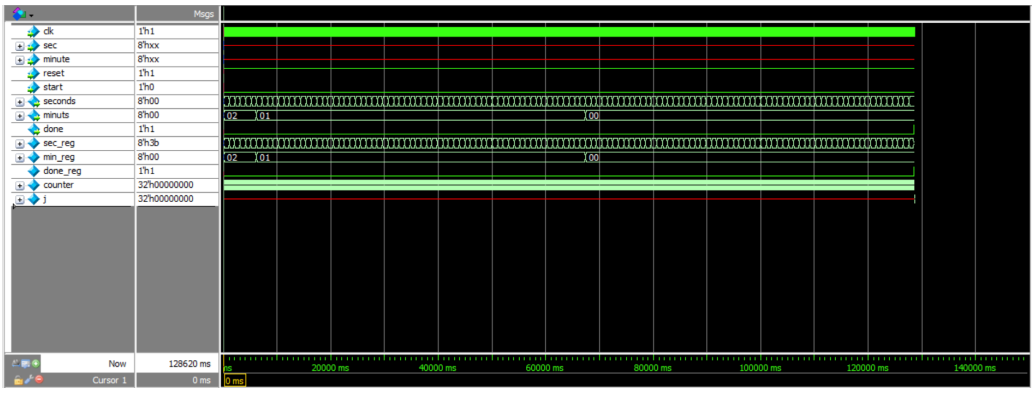
```
34          else if(!done)begin
35          seconds <= sec_reg;
36          minuts <= min_reg;
37          if(counter == 50000000)begin
38          counter <= 0;
39          if(sec_reg == 8'b00000000)begin
40          sec_reg <= 8'b00111011;
41          if (min_reg == 8'b00000000)begin
42          done_reg <= 1;
43          end
44          else
45          min_reg <= min_reg - 1;
46          end
47          else
48          sec_reg = sec_reg - 1;
49
50          end
51          else
52          counter <= counter + 1;
53          end
54          else if(done)begin
55          for(j=0; j<=8000; j=j+1)
56          $stop;
57          end
58          endmodule
59
```

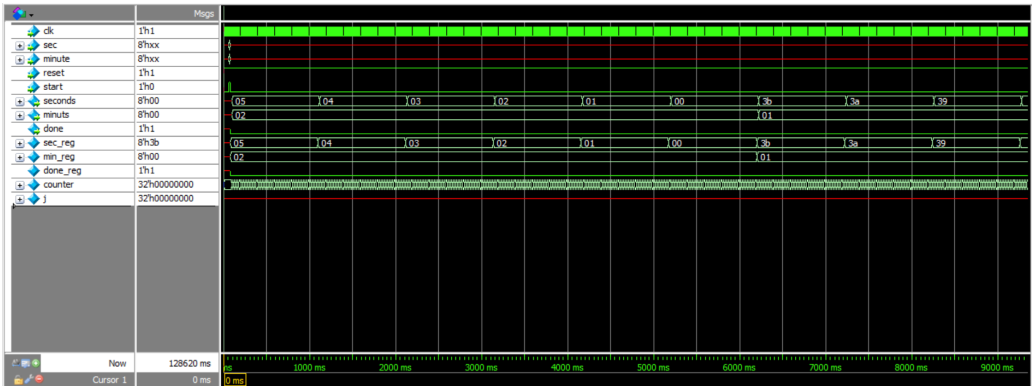


اگر کمی در انتهای کار زوم کنیم می بینیم که مقدار done برابر یک شده و پس از کمی تاخیر (این تاخیر عمدا و برای نمایش تغییر مقدار done در نظر گرفته شده است.) پروسه شمارش متوقف می شود.



به دلیل طولانی بودن زمان اجرای کد در نهایت فرکانس را به ۵۰ هرتز کاهش داده و شبیه سازی را برای زمان ۲ دقیقه و ۵ ثانیه انجام دادم.





۶

عمل جمع ، تفریق و ضرب به صورت عادی انجام شده اند. برای عمل تقسیم از دو تابع Divide برای قسمت قبل از اعشار تقسیم و تابع Fraction برای قسمت اعشاری استفاده شده است.در مشخص کردن ارقام بعد از ممیز از روش Restoring برای تقسیم اعداد Fixed point استفاده شده است.

```
1        `timescale 1ns/1ns
2        module alu(
```

```
3           input [3:0] A, B,
4           input [1:0] sel,
5           input Clk,
6           output reg [7:0] ALU_out
7           );
8
9           wire [3:0] Q;
10          wire [3:0] q;
11          wire [3:0] R;
12
13          Divide d(.A(A), .B(B), .Clk(Clk), .Q(Q));
14          Fraction f(.A((A - (Q*B))),.B(B),.Clk(Clk),.Q(q));
15
16
17          always @(posedge Clk) begin
18          case (sel)
19          2'b00 : ALU_out = A+B;
20          2'b01 : ALU_out = A-B;
21          2'b10 : ALU_out = A*B;
22          2'b11 : ALU_out = {Q,q};
23          default: ALU_out = 8'b00000000;
24          endcase
25          end
26
27          endmodule
28
29          module Divide (
30          input [3:0] A, B,
31          input Clk,
32          output reg [3:0] Q
33          );
34          reg [3:0] r;
35
36          always @(posedge Clk ) begin
37          r <= A;
38          Q = 4'b0000;
39          while (r >= B) begin
40          Q = Q + 1;
41          r = r - B;
42          end
43
44          end
45
46          endmodule
47          module Fraction(
48          input [3:0] A,B,
49          input Clk,
50          output reg [3:0] Q
51
52          );
53          reg [7:0] r;
54          integer i;
55          integer s;
56
57          always @(posedge Clk) begin
```

```
58          r <= {4'b0000,A};
59          for (i = 0 ;i<4 ;i = i+1 ) begin
60          if ((2 * r) < B ) begin
61          Q[3-i] = 0;
62          if(s==1) begin
63          r = A;
64          s = 0;
65          end
66          else  begin
67          r = 2*r;
68          s = 0;
69          end
70
71          end
72          else begin
73          Q[3-i] = 1;
74          r = (2 * r) - B ;
75          s = 1;
76          end
77          end
78
79          end
80          endmodule
81
```

خروجی ها برای هر چهار عمل به ازای $A = 5$ و $B = 4$ آورده شده است.

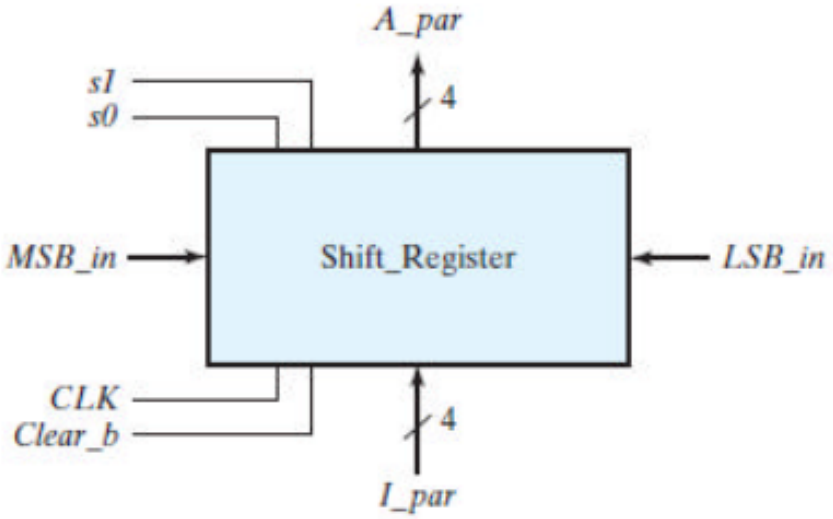
```
# 0101 + 0100 = 00001001
# 0101 - 0100 = 00000001
# 0101 * 0100 = 00010100
# 0101 / 0100 = 0001.0100
```

۷

در این سوال یک شیفت رجیستر یونیورسال پیاده سازی شده است. در این نوع شیفت رجیستر می توان محتوای شیفت رجیستر را به چپ یا راست شیفت داد همچنین می توان شیفت رجیستر را به صورت پارالل لود کرد یا آن را پاک (ریست) کرد. این شیفت رجیستر دارای ورودی های s_1 و s_0 است که مطابق جدولی که در ادامه آورده شده اعمال مورد نظر انجام می شود.دارای دو ورودی سریال است یکی از طرف چپ (MSB_in) و دیگری از طرف راست (LSB_in) که (MSB_in) هنگام شیفت به راست وارد شیفت رجیستر می شود و (LSB_in) هنگام شیفت به چپ وارد شیفت رجیستر می شود.

Function Table for the Register of Fig. 6.7

Mode Control		
s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load



ماژول پیاده سازی شده در ادامه آورده شده است:

```
`timescale 1ns/1ns
module shift_register(
input [3:0] I_par,
input s1,
input s0,
input MSB_in,
input LSB_in,
input clk,
input clear,
output reg [3:0] A_par
);
always @(posedge clk)
if(clear == 1)begin
A_par <= 4'b0000;
end
else
case({s1,s0})
2'b00: A_par <= A_par; //No change
2'b01: A_par <= {MSB_in , A_par[3:1]}; //Shift right
2'b10: A_par <= {A_par[2:0],LSB_in}; //Shift left
2'b11: A_par <= I_par; //Parallel Load of input
```

```
22          endcase
23          endmodule
24
```

خروجی به ازای چند ورودی مختلف در ادامه آورده شده است:

- ابتدا لود مقدار ورودی و سپس شیفت به اندازه ۱ واحد به سمت راست و ورود مقدار ۱

```
1           initial begin
2           clear = 0;
3           s1 = 1;
4           s0 = 1;
5           I_par = 4'b1010;
6           MSB_in = 'bx;
7           LSB_in = 'bx;
8           @(posedge clk);
9           @(posedge clk);
10          #1
11          $display("Value in binary: %b",ut.A_par);
12          clear = 0;
13          s1 = 0;
14          s0 = 1;
15          I_par = 4'b1010;
16          MSB_in = 1'b1;
17          LSB_in = 1'b1;
18          @(posedge clk);
19          @(posedge clk);
20          $display("Value in binary: %b",ut.A_par);
21          $stop;
22
```

```
Value in binary: 1010
Value in binary: 1101
```

- ابتدا لود مقدار ورودی و سپس شیفت به اندازه ۱ واحد به سمت چپ و ورود مقدار ۰

```
1           initial begin
2           clear = 0;
3           s1 = 1;
4           s0 = 1;
5           I_par = 4'b1010;
6           MSB_in = 'bx;
7           LSB_in = 'bx;
8           @(posedge clk);
9           @(posedge clk);
10          #1
11          $display("Value in binary: %b",ut.A_par);
12
13          clear = 0;
14          s1 = 1;
15          s0 = 0;
16          I_par = 4'b1010;
17          MSB_in = 1'b0;
```

```
18          LSB_in = 1'b0;
19          @(posedge clk);
20          @(posedge clk);
21          $display("Value in binary: %b",ut.A_par);
22          $stop;
23
```

```
# Value in binary: 1010
# Value in binary: 0100
```

- عدم تغییر خروجی

```
1           initial begin
2           clear = 0;
3           s1 = 1;
4           s0 = 1;
5           I_par = 4'b1010;
6           MSB_in = 'bx;
7           LSB_in = 'bx;
8           @(posedge clk);
9           @(posedge clk);
10          #1
11          $display("Value in binary: %b",ut.A_par);
12
13          clear = 0;
14          s1 = 0;
15          s0 = 0;
16          I_par = 4'b1010;
17          MSB_in = 1'b0;
18          LSB_in = 1'b0;
19          @(posedge clk);
20          @(posedge clk);
21          $display("Value in binary: %b",ut.A_par);
22          $stop;
23
```

```
Value in binary: 1010
Value in binary: 1010
```