پروژه وریلاگ پایان ترم درس مدارهای منطقی

استاد مربوطه: دکتر سارا ارشادی‌نسب

پاییز ۱۴۰۴

مقدمه

در این پروژه شما باید یک بازی رقابتی دو نفره را در Verilog طراحی و شبیه‌سازی کنید. هسته بازی بر پایه‌ی ماشین حالت متناهی (FSM) است. با استفاده از کلاک در هر نوبت هر بازیکن حرکت مورد نظر خود را انجام میدهد.

برای پوشش مفاهیم بیشتر مدار منطقی، یک سیستم «اعتباری» و «خرید اکشن » نیز اضافه شده است؛ در قبل از شروع هر بازی، دو بازیکن، اکشن های مورد نظر خود را با توجه به اعتبار اولیه ای که دارند و همچنین موجودی اکشن های داخل فروشگاه، خریداری میکنند.

این ترکیب دو منظوره باعث می‌شود شما با کاربرد مدارهای ترتیبی و ترکیبی(FSM ، شمارنده‌ها، رجیسترها، جمع و تفریق دودویی، مالتی‌پلکسر) بیشتر آشنا شوید.

در ادامه جزئیات طراحی پروژه گفته شده است.

---

۱. توضیحات فروشگاه

فروشگاه برای هر بازیکن به‌صورت **کاملاً مستقل** پیاده‌سازی می‌شود و هیچ وابستگی یا اشتراک سیگنالی بین فروشگاه‌های بازیکنان مختلف وجود ندارد. هر فرایند خرید فقط روی همان بازیکن اثر می‌گذارد.

- **اعتبار بازیکن**
  - هر بازیکن در ابتدای بازی دارای **۵۰۰ واحد اعتبار اولیه** است.
  - در صورت انجام **خرید موفق**، مقدار قیمت آیتم خریداری‌شده از اعتبار بازیکن کسر می‌شود.
  - کسر اعتبار تنها در صورت موفق بودن خرید انجام می‌شود.

- **آیتم‌ها و اکشن‌ها**

هر فروشگاه دارای **۵ آیتم** است. انتخاب آیتم توسط سیگنال ActionNumber[۲:۰] انجام می‌شود:

  - مقادیر ۰ تا ۴ اکشن معتبر بوده که متناظر با آیتم‌های فروشگاه هستند.
  - مقادیر ۵، ۶ و ۷ اکشن نامعتبر در نظر گرفته می‌شوند.

- **تعیین قیمت آیتم‌ها**

قیمت هر آیتم با استفاده از یک **مالتی‌پلکسر** بر اساس مقدار ActionNumber[۲:۰] (نوع اکشن انتخابی) انتخاب می‌شود:

- پنج قیمت ورودی Price۰ تا Price٤ برای آیتم‌ها وجود دارد.
- در صورت انتخاب اکشن نامعتبر، آیتم غیرمجاز شناسایی شده و خرید انجام نمی‌شود .

- **شمارنده موجودی آیتم‌ها**

برای هر آیتم یک **شمارنده موجودی مستقل** در نظر گرفته شده است.

- با هر خرید موفق، شمارنده مربوط به آن آیتم **یک واحد کاهش** می‌یابد.
- اگر موجودی یک آیتم صفر باشد، خرید آن آیتم مجاز نیست و سیگنال خطای مربوطه فعال می‌شود.

- **شرط انجام خرید**

خرید تنها در صورتی انجام می‌شود که **تمام شرایط زیر هم‌زمان برقرار باشند**:

۱. اکشن انتخاب‌شده معتبر باشد.
۲. اعتبار بازیکن **بیشتر یا مساوی** قیمت آیتم انتخابی باشد.
۳. موجودی آیتم انتخاب‌شده بزرگ‌تر از صفر باشد.

- در صورت برقرار بودن این شرایط، سیگنال purchase_success فعال شده و:
  - خرید ثبت می‌شود،
  - موجودی آیتم کاهش می‌یابد،
  - در مرحله بعد، اعتبار بازیکن به میزان قیمت آیتم کاهش داده می‌شود.

- **منطق خطا**

در صورت برقرار نبودن هر یک از شرایط خرید، هیچ تغییری در اعتبار بازیکن یا موجودی آیتم‌ها ایجاد نمی‌شود و سیگنال خطای مناسب فعال خواهد شد، از جمله:

- خطای اکشن نامعتبر،
- خطای کمبود اعتبار،
- خطای نبود موجودی آیتم.

- **جمع‌بندی ساختار ماژول Shop**

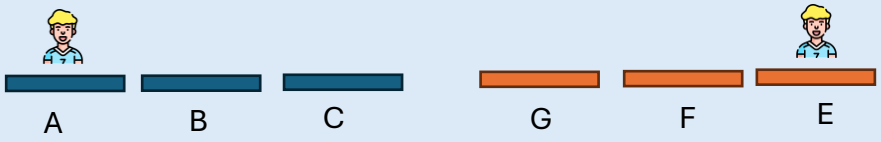
ماژول Shop شامل بخش‌های زیر است:

- مالتی‌پلکسر انتخاب قیمت آیتم،
- منطق تشخیص اکشن معتبر یا نامعتبر،
- مقایسه‌کننده اعتبار بازیکن و قیمت آیتم،
- شمارنده‌های کاهشی موجودی آیتم‌ها،
- منطق تولید سیگنال‌های موفقیت خرید و خطا.

هر فروشگاه به‌صورت مستقل برای هر بازیکن عمل کرده و هیچ تأثیری بر فروشگاه یا وضعیت بازیکن دیگر ندارد.

۲. هسته بازی

- بازی شامل ۲ بازیکن است که هر کدام ۳ خانه برای حرکت در زمین خود دارند، هر بازیکن فقط می‌تواند در زمین خود حرکات را انجام دهد.
- بازی به صورت نوبتی انجام می‌شود و هر بازیکن با توجه به اینکه نوبت آن است(Turn) می‌تواند حرکت خود را انجام دهد.
- بازی برای بازیکن سمت راست از اولین خانه سمت راست و برای بازیکن سمت چپ از اولین خانه سمت چپ شروع می‌شود.
- هدف هر بازیکن کم کردن تعداد جان‌های بازیکن رقیب می‌باشد که با توجه به حرکت هایی که در جدول توضیح داده شده میتواند این کار را انجام دهد.
- اگر حرکت بازیکن فعلی باعث کاهش جان حریف مقابل شود باید در همین کلاکی که حرکت انجام شده از تعداد جان‌های حریف کم شود.
- بازی زمانی به پایان می‌رسد که تعداد جان یکی از بازیکنان برابر صفر شود.
- در انتهای بازی در خروجی باید بازیکن برنده مشخص شود.



- حرکات بازیکن ها:
  - هر بازیکن در ابتدای بازی با توجه به موجودی ای که در اختیارش قرار گرفته تعدادی حرکت را خریداری می‌کند(باید تعداد هر حرکت خاص هر بازیکن در شمارنده مخصوص آن حرکت بازیکن ذخیره شود تا در صورت استفاده از آن حرکت با استفاده از شمارنده کاهشی (Decrement Counter) از تعداد آن حرکت بازیکن کاسته شود).
  - اگر بازیکن قصد انجام حرکتی داشته باشد ولی از آن حرکت در موجودی خودش وجود نداشته باشد(یعنی مقدار رجیستر آن حرکت ۰ باشد) باید سیگنال ارور چاپ شود.
  - باید مکان خانه های بازیکنان را به صورت کدگذاری شده در رجیستر ذخیره کرده تا در هنگام بررسی اثر ضربه ها بر روی بازیکن حریف امکان محاسبه فاصله مکانی آن‌ها به خصوص در دو حرکت Kick , Punch ممکن باشد.

  - در جدول زیر انواع حرکات موجود در بازی مشخص شده است:

اسم حرکت	نوع حرکت	فاصله اثر حرکت بر رقیب	تاثیر ضربه موفق بر تعداد جان رقیب
Kick	لگد زدن	در فاصله۱ بر رقیب اثر می‌کند	۱ جان از رقیب کم می‌کند
Punch	مشت زدن	در فاصله ۰ بر رقیب اثر می‌کند	۲ جان از رقیب کم می‌کند

Move left	حرکت به سمت خانه چپ	تاثیری بر رقیب ندارد	بی اثر
Move right	حرکت به سمت خانه راست	تاثیری بر رقیب ندارد	بی اثر
Wait	ایستادن روی حالت فعلی	تاثیری بر رقیب ندارد	بی اثر

- **ماشین حالت هر بازیکن:(FSM)**

دو ماشین FSM با توجه به اینکه در این کلاک نوبت کدام بازیکن است و با استفاده از Turn فعالیت خود را اجرا می‌کنند.

- **حالت‌ها:** مکان خانه های هر بازیکن به عنوان حالت های او در نظر گرفته میشود، به عنوان مثال در نمونه بالا هر بازیکن ۱ و ۲ به ترتیب دارای حالت های A,B,C و E,F,G می باشد، علاوه بر این حالت ها باید یک حالت برای زمان پایان بازی نیز در نظر گرفته شود.
- **ورودی ها:** نوبت بازی(Turn)، تعداد جان بازیکن طرف مقابل، حرکتی که بازیکن قصد دارد آن را انجام دهد.
- **خروجی‌ها**: آدرس خانه جدید بازیکن در محیط بازی، تعداد جان بازیکن حریف

مقادیر ورودی و خروجی می‌تواند بر اساس پیاده سازی شما تغییر کند.

---

۳. نمایشگر

- نمایش وضعیت‌ها و خروجی‌های بازی (مثل ارور ورودی نامعتبر و نتیجه هر دور بازی).
- مقادیر عددی مربوط به بازی (مثل جان و اعتبار هر بازیکن، تعداد هر اکشن هر بازیکن و مکان فعلی بازیکن) با دستور display در خروجی نمایش داده می‌شوند.
- غیر از مقادیر مربوط به بازی، مقادیر مربوط به فروشگاه مثل کد حرکت‌ها، موجودی فعلی و قیمت هر کدام را هم در خروجی نمایش بدهید.
- به صورت کلی انتظار می‌رود در این بخش تمام مواردی که بعد از انجام هر حرکت یا خرید تغییر میکند، چاپ شود.
- در مثال زیر تنها نمونه ای از نمایش برخی موارد نشان داده است:

```
module display_example;
  reg [1:0] p1_health;
  reg [1:0] p2_health;
  reg [9:0] p1_credit;
  reg [9:0] p2_credit;
```

```
  initial begin
    p1_health = 2'd3;
    p2_health = 2'd3;
    p1_credit = 10'd500;
    p2_credit = 10'd500;

    $display("P1 Health: %0d", p1_health);
    $display("P2 Health: %0d", p2_health);
    $display("P1 Credit: %0d", p1_credit);
    $display("P2 Credit: %0d", p2_credit);

    #10 $finish;
  end

endmodule
```

---

٤. **بخش نمره اضافه: شروع مجدد بازی پس از پایان یک دور**

در این بخش باید به ماشین حالت هر بازیکن یک حالت «فروشگاه» اضافه کنید. یک حالت جدید D=SHOP تعریف می‌کنیم تا چرخه‌ی بازی به شکل «دور اول ← فروشگاه ← دور دوم» انجام شود.

- ساختار کلی انتقال حالت‌ها: A ← B ← C ← D(SHOP) ← A ← ...
- یعنی بعد از اتمام دور ابتدایی حتماً وارد حالت SHOP شوید. در SHOP بازیکن دوباره همان اعتبار اولیه(**۵۰۰ واحد**) در موجودی اش قرار می‌گیرد و میتواند اکشن های مورد نیاز برای دور بعدی بازی خود را بخرد؛ سپس دوباره وارد A می‌شود و راند بعدی شروع می‌شود.
- نکته‌ی مهم: «پایان بازی» دیگر یک state جدا نیست. اگر در هر نقطه از بازی جان یکی از بازیکن‌ها صفر شود، به‌جای رفتن به END باید مستقیم وارد SHOP شوید و از همان‌جا بازیِ جدید را آغاز کنید.
- تفاوت اعتباری که در این دور با دور اول دارد این است که بازیکن برنده باید براساس شرایط زیر تخفیفی را برای خرید اجناس داشته باشد؛ مقدار این تخفیف با کمک یک مقایسه‌گر دودویی بر اساس شرایط گفته شده در ادامه تعیین می‌شود.

برای این کار یک شمارنده‌ی moves_to_win داشته باشید که تعداد حرکت‌های معتبر از شروع دور اول تا لحظه‌ی برد را می‌شمارد (پیشنهاد: هر سیکلِ PLAY که اکشن معتبر است، به مقدار ان اضافه شود)، وقتی برد اتفاق افتاد، با Comparator این قوانین را بسازید:

- اگر بازیکن در ۶ حرکت یا کمتر برنده شد ← تخفیف ٪۲۰

- اگر بازیکن بین ۷ تا ۹ حرکت برنده شد → تخفیف ٪۱۰
- در غیر این صورت → تخفیف ٪۵

این تخفیف در حالت SHOP روی قیمت اکشن ها اعمال می‌شود و بازیکن با همان تخفیف می‌تواند خرید انجام دهد.

- برای شروع دور جدید بازی از داخل SHOP باید بعضی رجیسترها ریست شوند:
  - جان هر دو بازیکن دوباره ۳ شود و اعتبار/امتیاز شروع برای هر دو بازیکن ۵۰۰ در نظر گرفته شود. این ریست‌ها باید همگام با کلاک و کنترل‌شده (با enable روی رجیسترها) انجام شوند؛ مثلاً با یک سیگنال start_round هنگام خروج از SHOP و ورود به A.
- در نهایت چون ورودی‌ها رشته‌ای نیستند، تعریف «حرکات دورهای بازی» باید در Testbench انجام شود: برای هر کلاک اکشن‌ها را با کد عددی اعمال کنید و وقتی سیستم وارد SHOP شد، ورودی خرید (buy_valid و buy_code) را بدهید. با این کار می‌توانید سناریوی «برد در ≥۶ حرکت» یا «برد در ۷ تا ۹ حرکت» را دقیقاً در تست‌بنچ بسازید و تخفیف را مشاهده کنید.

پیاده‌سازی ناقص این قسمت نیز، متناسب با میزان انجام‌شده، مشمول نمره‌ی اضافه خواهد بود.

---

۵. Testbench

در ابتدا باید خرید از فروشگاه برای هر بازیکن انجام شود (مقداردهی اولیه فروشگاه هم میتواند در داخل تست بنچ انجام شود هم به صورت کلی از ابتدا داخل کد باشد) و بعد از آن بازی شروع می‌شود و به ترتیب اکشن های بازیکن ها ورودی داده شوند.

---

٦. نکات تکمیلی:

- انجام پروژه به صورت انفرادی و یا گروه های دونفره است.

- در صورت مشاهده شباهت غیر متعارف میان پروژه افراد در صورتی که درصد شباهت بیشتر از ۳۰ درصد باشد نمره کل دو گروه در (درصد تشابه - ۱۰۰) ضرب میشود.

- تسلط به بخش‌های مختلف پروژه در هنگام تحویل الزامی است.

- در صورت هرگونه ابهام می‌توانید با خانم محرابی و خانم شاهرودنژاد و خانم تشتریان در ارتباط باشید.

- تحویل تکلیف پس از مهلت داده شده نمره ای نخواهد داشت .

- فایل‌های نهایی پروژه خود را به همراه شکل ماشین fsm خود به صورت zip در قالب زیر در سامانه vu بارگذاری کنید:

FirstNameLastName_StudentNumber_finalProject.zip

موفق باشید.