Card Verifier

دانشگاه بوعلی سینا

Bu-Ali Sina University

محققین : متین امیرپناه فر - نیما مخملی

استاد درس: دکتر حاتم عبدلی

فهرست

گزارش کار پروژه طراحی دیجیتال Card Verifier

مقدمه

در این پروژه، یک سیستم دیجیتال برای اعتبارسنجی شماره کارت‌های بانکی طراحی و پیاده‌سازی شده است. هدف این سیستم بررسی صحت شماره کارت‌های ورودی به کمک الگوریتم Luhn می‌باشد. این پروژه با استفاده از زبان توصیف سخت‌افزار VHDL و بر روی معماری دیجیتال مناسب توسعه داده شده است. سیستم طراحی‌شده در محیط شبیه‌سازی و سخت‌افزار FPGA قابل پیاده‌سازی است.

در این پروژه معماری به صورت structural هست و هر ماژول نیز به همین صورت پیاده سازی شده است.
همچنین این پروژه شامل 6 ماژول جدا هست که به شرح زیر میباشند

BCD_Multiplier_2
Card_Verifier
Double_Odd_Position
Luhn_Validator
compute_sum
reverse_bytes

معرفی الگوریتم Card Verifier

Card Verifier سیستمی است که شماره کارت‌های بانکی ورودی را دریافت کرده و با بررسی ساختار آنها، مشخص می‌کند که آیا این شماره معتبر است یا خیر. این بررسی با استفاده از الگوریتم‌های استاندارد اعتبارسنجی انجام می‌شود.

یکی از این الگوریتم‌های رایج، الگوریتم Luhn است که توسط اغلب بانک‌ها و سیستم‌های پرداخت مانند Visa و MasterCard مورد استفاده قرار می‌گیرد.

توضیح الگوریتم Luhn

الگوریتم Luhn که به نام "Checksum Algorithm" نیز شناخته می‌شود، روشی ساده و مؤثر برای تشخیص خطاهای ورود داده در شماره کارت‌ها است. این الگوریتم به صورت زیر عمل می‌کند:

مراحل الگوریتم:

1. شماره کارت ورودی به صورت رشته‌ای از اعداد دریافت می‌شود.

2. از سمت راست به چپ (به جز آخرین رقم که رقم کنترلی است)، هر رقم دوم را دو برابر می‌کنیم.

- اگر حاصل ضرب بیش از 9 شد، مجموع ارقام آن محاسبه می‌شود (به‌عبارت‌دیگر، حاصل ضرب را 9 واحد کم می‌کنیم).

3. مجموع تمامی ارقام را محاسبه می‌کنیم.

4. اگر مجموع به‌دست‌آمده مضربی از 10 باشد، شماره کارت معتبر است؛ در غیر این صورت نامعتبر است.

مثال:

شماره کارت: 5467 1234 2612 4561

- دو برابر کردن هر رقم دوم: 8، 5، 2، 2، 4 و ...

- محاسبه مجموع ارقام: 70

- چون 70 مضربی از 10 است، این شماره کارت معتبر است.

	7	9	9	2	7	3	9	8	7	1
Multipliers	2	1	2	1	2	1	2	1	2	1
	=	=	=	=	=	=	=	=	=	=
	14	9	**18**	2	**14**	3	**18**	8	**14**	1
Sum digits	**5** (1+4)	9	**9** (1+8)	2	**5** (1+4)	3	**9** (1+8)	8	**5** (1+4)	1

الگوریتم Luhn تمام خطاهای تک رقمی و همچنین تقریباً همه جابجایی ارقام مجاور را شناسایی می کند. با این حال، جابجایی دنباله دو رقمی 09 به 90 (یا برعکس) را تشخیص نمی دهد. اکثر خطاهای احتمالی دوقلو را شناسایی می کند (22 ↔ 55، 33 ↔ 66 یا 44 ↔ 77 را تشخیص نمی دهد).

سایر الگوریتم های پیچیده تر چک رقمی (مانند الگوریتم Verhoeff و الگوریتم Damm) می توانند خطاهای رونویسی بیشتری را شناسایی کنند. الگوریتم Luhn mod N پسوندی است که رشته های غیر عددی را پشتیبانی می کند.

از آنجایی که الگوریتم بر روی ارقام به صورت راست به چپ عمل می‌کند و ارقام صفر تنها در صورتی بر نتیجه تأثیر می‌گذارند که باعث جابجایی در موقعیت شوند، صفر کردن ابتدای رشته‌ای از اعداد بر محاسبه تأثیری ندارد. بنابراین، سیستم‌هایی که به تعداد مشخصی از ارقام اضافه می‌کنند (مثلاً با تبدیل 1234 به 0001234) می‌توانند اعتبارسنجی Luhn را قبل یا بعد از padding انجام دهند و به همان

نتیجه برسند.

این الگوریتم در یک پتنت ایالات متحده[1] برای یک دستگاه مکانیکی ساده و دستی برای محاسبه جمع کنترل ظاهر شد. دستگاه مود sum 10 را به روش مکانیکی گرفت. ارقام تعویض، یعنی نتایج روش دوبل و کاهش، به صورت مکانیکی تولید نشدند. در عوض، ارقام به ترتیب تغییر یافته خود بر روی بدنه دستگاه علامت گذاری شده بودند.

```
function isValid(cardNumber[1..length])
    sum := 0
    parity := length mod 2
    for i from 1 to length do
        if i mod 2 != parity then
            sum := sum + cardNumber[i]
        elseif cardNumber[i] > 4 then
            sum := sum + 2 * cardNumber[i] - 9
        else
            sum := sum + 2 * cardNumber[i]
        end if
    end for
    return cardNumber[length] == ((10 - (sum mod 10)) mod 10)
end function
```

ماژول‌های سیستم طراحی‌شده

ماژول BCD_Multiplier_2

این ماژول به زبان VHDL نوشته شده و مربوط به یک ضرب‌کننده BCD در عدد 2 است. در ادامه توضیحات هر بخش و عملکرد کلی را برای شما شرح می‌دهم:

1. کتابخانه‌ها

```
;library IEEE

;use IEEE.STD_LOGIC_1164.ALL

;use IEEE.NUMERIC_STD.ALL
```

این بخش کتابخانه‌های لازم را برای استفاده از انواع داده‌ها و عملیات منطقی و محاسبات عددی وارد می‌کند:

- STD_LOGIC_1164: برای کار با سیگنال‌های منطقی (مثل std_logic_vector)

- NUMERIC_STD: برای انجام عملیات عددی روی انواع داده مثل unsigned و signed.

2. تعریف موجودیت (Entity)

```
entity BCD_Multiplier_2 is
    ) Port (
        ;bcd_in  : in  std_logic_vector(3 downto 0)
        bcd_out : out std_logic_vector(3 downto 0)
    );(
end BCD_Multiplier_2;
```

- *bcd_in: ورودی 4 بیتی که یک عدد BCD (Binary Coded Decimal) را نمایش می‌دهد.*

- *bcd_out: خروجی 4 بیتی که نتیجه ضرب عدد BCD در 2 را نمایش می‌دهد.*

3. تعریف معماری (Architecture)

```
architecture Behavioral of BCD_Multiplier_2 is
    signal temp : unsigned(4 downto 0);
```

یک سیگنال 5 بیتی (temp) تعریف شده است تا محاسبات انجام شده در آن ذخیره شود. 5 بیت لازم است چون ممکن است ضرب باعث تولید بیت اضافه (Carry) شود.

4. پردازش ضرب و اصلاح BCD

```
process(bcd_in)
begin
    temp <= ("0" & unsigned(bcd_in)) sll 1;
```

- ابتدا bcd_in به نوع unsigned تبدیل شده و یک بیت 0 در سمت چپ آن اضافه می‌شود تا 5 بیتی شود.
- سپس با استفاده از عملیات sll 1 (Shift Left Logical) عدد دو برابر می‌شود (معادل ضرب در 2).

5. تصحیح برای BCD معتبر

```
if temp > "01001" then
    temp <= temp + "0011";
end if;
```

- اگر مقدار temp از عدد 9 (01001 در باینری) بزرگ‌تر باشد، 3 (0011) به آن اضافه می‌شود. این اصلاح برای حفظ فرمت BCD لازم است.

6. تخصیص خروجی

```
vhdl
bcd_out <= std_logic_vector(temp(3 downto 0));
```

- پایین‌ترین 4 بیت از temp به خروجی bcd_out اختصاص داده می‌شود.

عملکرد کلی

این کد عدد ورودی BCD را در 2 ضرب می‌کند و اگر نتیجه از 9 بزرگ‌تر باشد (که در BCD نامعتبر است)، با اضافه کردن عدد 3، نتیجه را به BCD معتبر تبدیل می‌کند.

مثال

اگر bcd_in = 4 باشد (معادل 4 در ده‌دهی):

- 4 * 2 = 8 → نیازی به اصلاح نیست → bcd_out = 8

اگر bcd_in = 7 باشد:

- 7 * 2 = 14 → در BCD باید اصلاح شود → bcd_out = 4 (با حمل به بیت بعدی)

ماژول reverse_bytes

این کد یک ماژول VHDL برای معکوس کردن ترتیب بایت‌های یک بردار است که طول آن 60 بیت (15 نایبل یا 15 گروه 4 بیتی) است. ورودی input_vector یک بردار 60 بیتی است که شامل داده‌های ورودی به صورت گروه‌های 4 بیتی (هر نایبل معادل یک رقم BCD یا بخشی از داده) است. خروجی output_vector بردار 60 بیتی دیگری است که داده‌های ورودی را به صورتی بازگردانده که ترتیب بایت‌ها (گروه‌های 4 بیتی) کاملاً معکوس شده باشد.

عملکرد کلی این ماژول در فرآیندهای مربوط به Card Verifier، مانند اعتبارسنجی شماره کارت یا پردازش داده‌های رمزگذاری شده، اهمیت دارد. معکوس کردن بایت‌ها ممکن است در الگوریتم‌های مرتبط با پردازش کارت (مانند الگوریتم Luhn یا سایر الگوریتم‌های بررسی) استفاده شود. این کار می‌تواند برای انطباق با پروتکل‌های خاص، ساختار داده‌ها، یا نیازهای پردازش مستقیم انجام شود.

برای مثال، اگر ورودی به صورت ABCDEF123456789 باشد (هر رقم معادل یک نایبل 4 بیتی)، خروجی به صورت FEDCBA987654321 خواهد بود. این معکوس‌سازی می‌تواند در فرآیندهایی که ترتیب داده‌ها در پردازش یا مقایسه تأثیر دارد، نقشی کلیدی ایفا کند.

ماژول compute_sum

این کد یک ماژول VHDL است که مجموع مقادیر 15 نایبل (گروه‌های 4 بیتی) را از بردار ورودی 60 بیتی محاسبه می‌کند. ورودی input_vector شامل 15 رقم 4 بیتی به صورت std_logic_vector(59 downto 0) است و خروجی sum_out یک عدد صحیح (integer) است که مجموع این ارقام را نمایش می‌دهد.

در این ماژول، سیگنال sum_value برای ذخیره مجموع موقت مقادیر تعریف شده و در داخل یک فرآیند (Process) مقداردهی و به‌روزرسانی می‌شود. در حلقه for، هر گروه 4 بیتی از بردار ورودی استخراج شده و به مقدار عدد صحیح تبدیل می‌شود (to_integer(unsigned(...))). سپس این مقدار به متغیر sum_value اضافه می‌شود. در پایان، نتیجه در خروجی sum_out قرار می‌گیرد.

کاربرد در Card Verifier

این ماژول می‌تواند برای محاسبه مجموع ارقام یک شماره کارت یا سایر داده‌های عددی استفاده شود. در الگوریتم‌هایی مانند Luhn، محاسبه مجموع ارقام بخشی از فرآیند اعتبارسنجی است. این ماژول مجموع ارقام را فراهم می‌کند که ممکن است به‌عنوان یکی از مراحل پردازش کارت برای بررسی صحت شماره کارت به کار رود.

برای مثال، اگر ورودی input_vector = "000100100011010000101"... باشد (که هر گروه 4 بیتی معادل یک رقم است)، این ماژول مجموع همه ارقام را محاسبه کرده و به صورت عدد صحیح در sum_out قرار می‌دهد.

ماژول Luhn_Validator

این کد یک ماژول VHDL به نام Luhn Validator است که برای بررسی صحت شماره کارت اعتباری بر اساس الگوریتم Luhn طراحی شده است. در ادامه، عملکرد کلی و منطق ماژول توضیح داده شده است.

هدف کلی ماژول

ماژول بررسی می‌کند که آیا یک شماره کارت ورودی (64 بیتی) معتبر است یا خیر. الگوریتم Luhn بر اساس محاسبات خاص روی ارقام کارت کار می‌کند و نتیجه‌اش تعیین می‌کند که کارت معتبر است یا خیر. خروجی valid_out نشان می‌دهد که آیا شماره کارت معتبر ('1') یا نامعتبر ('0') است.

عملکرد گام‌به‌گام

1. جداسازی ارقام کارت

- آخرین رقم کارت (end_digit) به‌صورت جداگانه در سیگنال end_digit ذخیره می‌شود. این رقم به‌عنوان رقم کنترلی (Check Digit) در الگوریتم Luhn استفاده می‌شود.

- باقی‌مانده ارقام کارت (از بیت 63 تا 4) در سیگنال digits ذخیره می‌شوند. این ارقام وارد فرآیند پردازش می‌شوند.

2. معکوس کردن ارقام (Reverse Bytes)

- ماژول Reverse_Bytes (تعریف شده در کد قبلی) ترتیب ارقام کارت (به جز رقم کنترلی) را معکوس می‌کند و نتیجه را در سیگنال reversed_number قرار می‌دهد. این گام ضروری است زیرا در الگوریتم Luhn، پردازش از انتهای شماره کارت شروع می‌شود.

3. دو برابر کردن ارقام در موقعیت‌های فرد (Double Odd Positions)

- ماژول Double_Odd_Positions مقادیر ارقام در موقعیت‌های فرد (با توجه به ترتیب معکوس شده) را دو برابر می‌کند. اگر نتیجه دو برابر کردن یک عدد از 9 بیشتر شود، عدد به صورت BCD اصلاح می‌شود (برای حفظ فرمت). خروجی در سیگنال doubled_values ذخیره می‌شود.

4. محاسبه مجموع ارقام (Compute Sum)

- ماژول Compute_Sum مجموع تمام ارقام (پس از پردازش مرحله قبل) را محاسبه می‌کند و نتیجه را در سیگنال sum_result قرار می‌دهد. این مجموع برای مرحله نهایی اعتبارسنجی استفاده می‌شود.

5. بررسی اعتبار کارت

- در فرآیند اصلی، با هر پالس ساعت (clk) و در صورتی که سیگنال reset فعال نباشد:

- مجموع محاسبه‌شده (sum_result) را بر 10 تقسیم می‌کند و باقی‌مانده آن را با رقم کنترلی (end_digit) مقایسه می‌کند.

- اگر باقی‌مانده با رقم کنترلی برابر باشد، کارت معتبر است ('valid_out = '1) و در غیر این صورت نامعتبر است ('valid_out = '0).

6. بازنشانی سیستم

- اگر سیگنال reset فعال شود ('reset = '1)، خروجی valid_out صفر می‌شود، یعنی کارت نامعتبر در نظر گرفته می‌شود.

کاربرد در Card Verifier

این ماژول بخش نهایی اعتبارسنجی شماره کارت در یک سیستم Card Verifier است. از توابع Reverse_Bytes، Double_Odd_Positions، و Compute_Sum به صورت سلسله‌مراتبی استفاده می‌کند تا الگوریتم Luhn را به‌طور کامل پیاده‌سازی کند. خروجی نهایی نشان می‌دهد که آیا

شماره کارت ورودی معتبر است یا خیر، که برای سیستم‌های پرداخت، بانکداری، و احراز هویت اهمیت دارد.

ماژول Double_Odd_Position

این کد یک ماژول VHDL به نام Double_Odd_Positions است که در الگوریتم Luhn و سایر فرآیندهای اعتبارسنجی کارت‌ها به کار می‌آید. وظیفه این ماژول دو برابر کردن ارقام در موقعیت‌های فرد (از سمت راست) و عبور دادن ارقام در موقعیت‌های زوج بدون تغییر است.

عملکرد کلی ماژول

ماژول Double_Odd_Positions برای پردازش بردار ورودی 60 بیتی (input_vector) طراحی شده است که شامل 15 نایبل (گروه 4 بیتی) است. این ماژول هر نایبل را بررسی کرده و در موقعیت‌های فرد (با توجه به شمارهگذاری از راست به چپ) آن را دو برابر می‌کند. برای دو برابر کردن ارقام، از ماژول BCD_Multiplier_2 استفاده می‌شود که وظیفه ضرب در 2 را برای هر نایبل به عهده دارد.

جزئیات پردازش

1. استفاده از ماژول BCD_Multiplier_2

- برای دو برابر کردن هر نایبل (رقم 4 بیتی)، از ماژول BCD_Multiplier_2 استفاده می‌شود که در کدهای قبلی توضیح داده شده است. این ماژول ورودی یک نایبل 4 بیتی را می‌گیرد و آن را در 2 ضرب می‌کند.

- ماژول BCD_Multiplier_2 برای هر نایبل که در موقعیت‌های فرد (0، 2، 4 و ...) قرار دارد، فراخوانی می‌شود.

2. حلقه جنریک (Generate Loop)

- در این ماژول، یک حلقه generate از 0 تا 14 وجود دارد که 15 بار تکرار می‌شود (چون ورودی 60 بیتی شامل 15 نایبل است).

- برای هر تکرار حلقه، اگر موقعیت نایبل فرد باشد (یعنی اندیس i زوج باشد)، نایبل با استفاده از BCD_Multiplier_2 دو برابر می‌شود و نتیجه در bcd_out_signal قرار می‌گیرد.

- اگر موقعیت نایبل زوج باشد (اندیس i فرد باشد)، نایبل بدون تغییر در temp_vector ذخیره می‌شود.

3. ساخت خروجی

- در پایان حلقه، سیگنال temp_vector که شامل نایبل‌های دو برابر شده در موقعیت‌های فرد و نایبل‌های بدون تغییر در موقعیت‌های زوج است، به خروجی output_vector منتقل می‌شود.

خروجی نهایی

در نتیجه، خروجی output_vector برداری است که نایبل‌های در موقعیت‌های فرد را دو برابر کرده و نایبل‌های در موقعیت‌های زوج را بدون تغییر نگه می‌دارد.

کاربرد در الگوریتم Luhn

در الگوریتم Luhn، این ماژول به‌ویژه برای مرحله‌ای مفید است که در آن باید ارقام در موقعیت‌های فرد از انتها دو برابر شوند. در نتیجه، برای اعتبارسنجی شماره کارت‌ها یا دیگر داده‌ها، این ماژول کمک می‌کند تا پردازش‌ها به‌درستی انجام شود.

برای مثال، اگر ورودی به شکل زیر باشد:

... input_vector = 0110 0101 1000 1101 0011

پس از پردازش توسط این ماژول، نایبل‌هایی که در موقعیت‌های فرد (از سمت راست) قرار دارند، دو برابر خواهند شد و خروجی به این صورت خواهد بود.

نتیجه‌گیری

ماژول Double_Odd_Positions وظیفه دو برابر کردن ارقام در موقعیت‌های فرد از 15 نایبل ورودی را بر عهده دارد و از ماژول BCD_Multiplier_2 برای انجام این کار استفاده می‌کند. این پردازش در الگوریتم‌های اعتبارسنجی مانند Luhn برای بررسی صحت شماره‌های کارت به کار می‌آید.

ماژول Card_Verifier

این کد VHDL مربوط به ماژول Card Verifier است که برای بررسی اعتبار شماره کارت استفاده می‌شود. در این ماژول، داده‌ها (که به صورت پیاپی از طریق ورودی chunk_in وارد می‌شوند) به تدریج در یک ثبت جابجایی (shift register) ذخیره می‌شوند و در نهایت پس از دریافت تمام داده‌ها، اعتبار شماره کارت با استفاده از الگوریتم Luhn بررسی می‌شود. خروجی این بررسی به‌طور نهایی در valid_out قرار می‌گیرد.

عملکرد کلی ماژول

ماژول به ورودی‌های مختلفی نیاز دارد:

- clk: پالس ساعت برای هماهنگ‌سازی عملیات.
- rst: سیگنال بازنشانی که برای بازنشانی وضعیت‌های داخلی استفاده می‌شود.
- enable: برای فعال کردن ورود داده‌ها و فرآیند بررسی اعتبار کارت.
- chunk_in: هر باری که ورودی داده به صورت 4 بیت (یک نایبل) وارد می‌شود.
- valid_out: نشان می‌دهد که آیا شماره کارت معتبر است یا خیر.

جزئیات عملکرد

1. ثبت جابجایی (Shift Register)

- سیگنال shift_reg یک ثبت جابجایی 64 بیتی است که برای ذخیره‌سازی 16 نایبل کارت طراحی شده است.

- هر بار که ورودی chunk_in جدید وارد می‌شود، 4 بیت داده به سمت راست جابجا می‌شود و 4 بیت جدید در سمت چپ قرار می‌گیرد. این عملیات تا زمانی ادامه می‌یابد که تمام 16 نایبل کارت وارد شوند.

2. شمارش تعداد چانک‌ها (Chunk Counter)

- سیگنال chunk_counter برای شمارش تعداد نایبل‌های واردشده استفاده می‌شود. این شمارش از 0 تا 15 ادامه می‌یابد (چون 16 نایبل در کارت داریم).

- هر بار که یک نایبل وارد می‌شود، شمارش یک واحد افزایش می‌یابد. وقتی که 16 نایبل (تمام داده کارت) وارد شوند، data_ready به '1' تنظیم می‌شود که به‌معنای آماده بودن داده‌ها برای پردازش است.

3. پردازش داده‌های ورودی

- در هر پالس ساعت (clk) و زمانی که enable برابر با '1' باشد، 4 بیت جدید از ورودی chunk_in به ثبت جابجایی shift_reg اضافه می‌شود.

- وقتی که تمام داده‌ها وارد می‌شوند (یعنی وقتی chunk_counter = 15 می‌شود)، سیگنال data_ready فعال می‌شود که نشان‌دهنده آماده بودن داده‌ها برای پردازش اعتبارسنجی است.

4. استفاده از ماژول Luhn Validator

- پس از اینکه تمام داده‌ها در shift_reg ذخیره شدند و data_ready فعال شد، داده‌ها به ماژول Luhn_Validator ارسال می‌شوند.

- ماژول Luhn_Validator داده‌ها را پردازش کرده و بررسی می‌کند که آیا شماره کارت معتبر است یا خیر. نتیجه اعتبارسنجی در سیگنال valid_tmp قرار می‌گیرد.

5. نتیجه‌گیری اعتبار

- در پایان، وقتی که داده‌ها آماده باشند (یعنی 'data_ready = '1)، سیگنال valid_tmp که نتیجه اعتبارسنجی توسط Luhn_Validator است، به valid_out منتقل می‌شود.

- اگر کارت معتبر باشد، 'valid_out = '1 خواهد شد و در غیر این صورت 'valid_out = '0 خواهد بود.

6. بازنشانی سیستم

- در صورت فعال بودن سیگنال rst (بازنشانی)، تمام سیگنال‌ها به حالت اولیه خود برمی‌گردند. به‌ویژه، shift_reg صفر می‌شود، شمارشگر به صفر باز می‌گردد، و valid_out به '0' تنظیم می‌شود.

چگونگی عملکرد ماژول

1. در هر پالس ساعت، 4 بیت از ورودی chunk_in به ثبت جابجایی وارد می‌شود.
2. هنگامی که 16 نایبل وارد ثبت جابجایی شدند، داده‌ها آماده برای پردازش اعتبارسنجی هستند.
3. سپس، داده‌ها به ماژول Luhn_Validator ارسال می‌شوند که بررسی می‌کند آیا شماره کارت با استفاده از الگوریتم Luhn معتبر است یا خیر.
4. نتیجه اعتبارسنجی در valid_out قرار می‌گیرد که می‌تواند به عنوان ورودی به سیستم‌های دیگر مانند سیستم پرداخت یا احراز هویت استفاده شود.

کاربرد در کارت‌های اعتباری

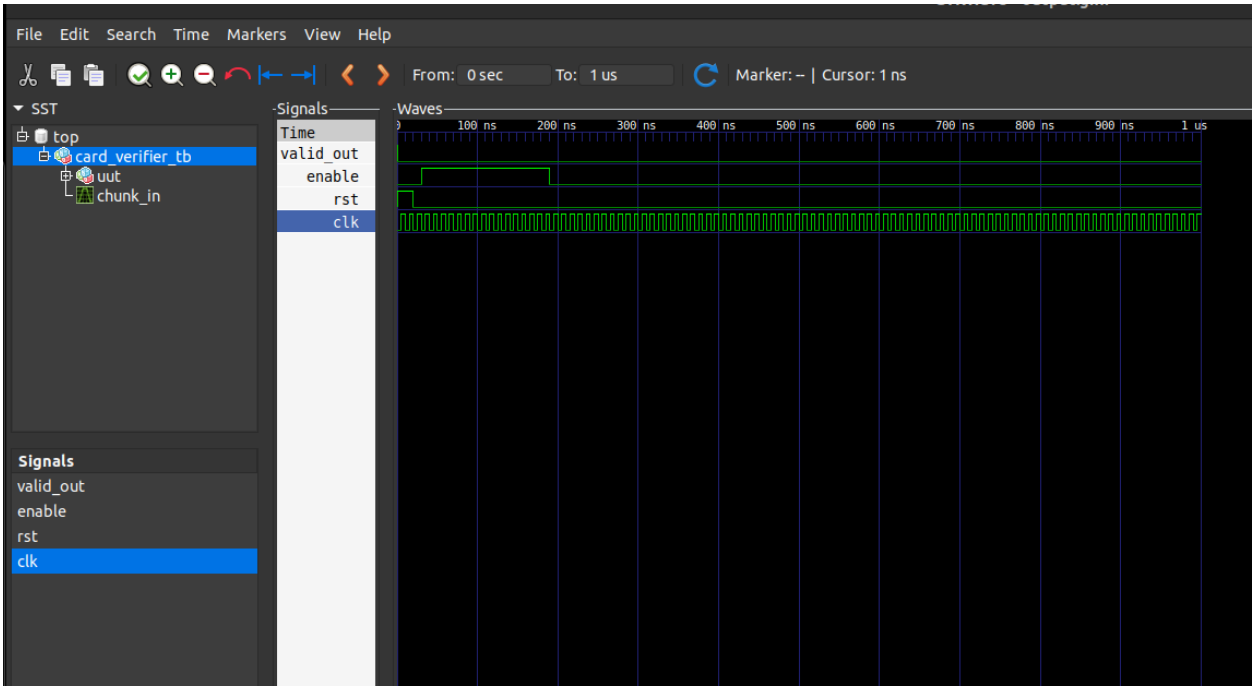
این ماژول می‌تواند در سیستم‌های کارت‌خوان یا دستگاه‌های مشابه برای اعتبارسنجی شماره کارت‌های اعتباری استفاده شود. وقتی که کاربر شماره کارت را وارد می‌کند، این ماژول به‌طور خودکار اعتبار کارت را بررسی می‌کند و مشخص می‌کند که آیا کارت معتبر است یا خیر.

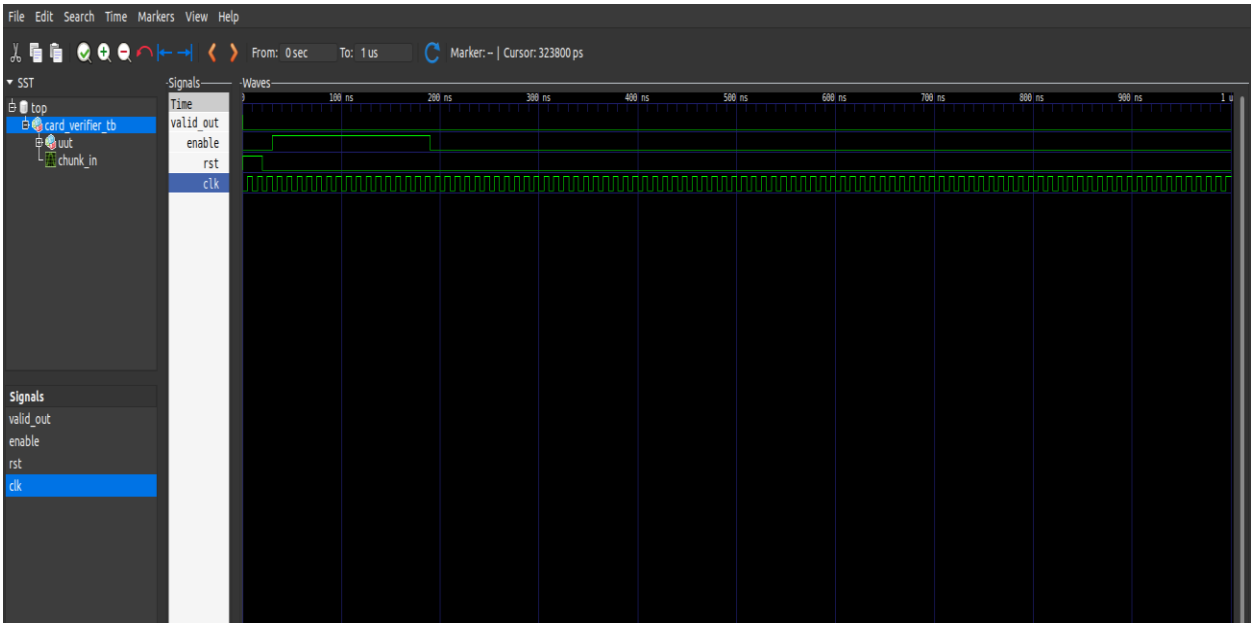
نتیجه‌گیری

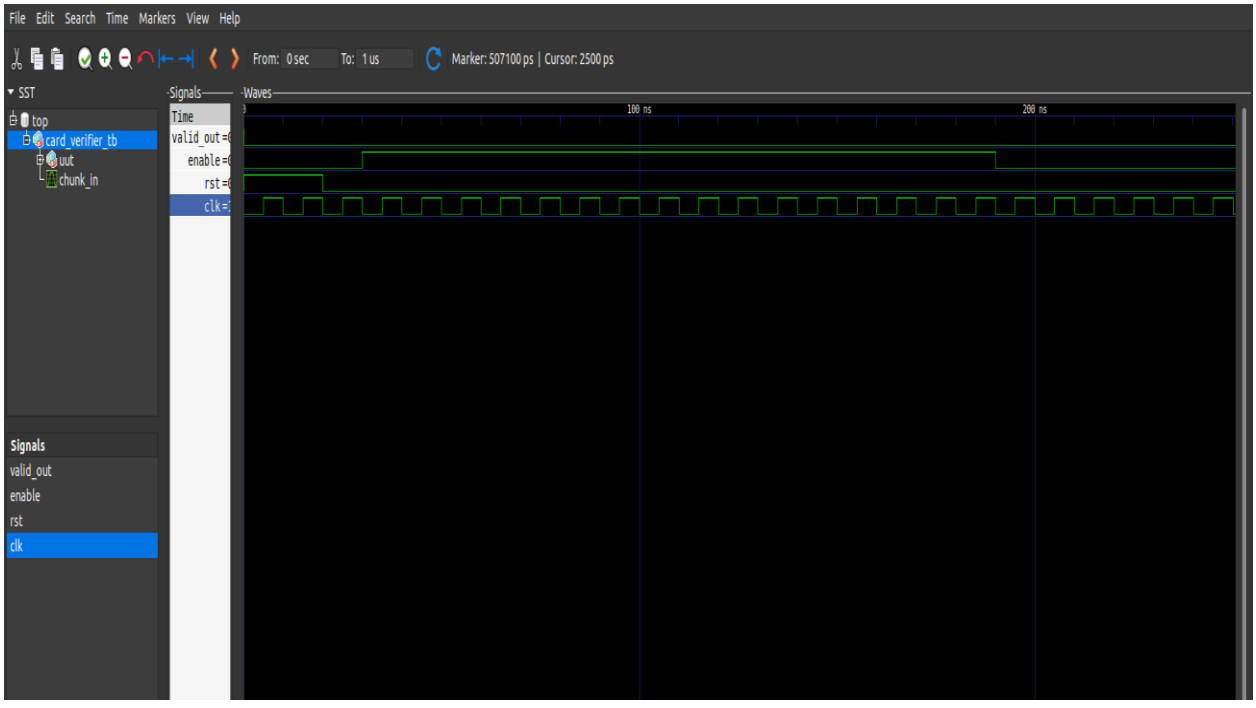
ماژول Card_Verifier به‌عنوان یک واحد برای جمع‌آوری داده‌ها از ورودی‌های پیاپی و سپس بررسی اعتبار شماره کارت با استفاده از الگوریتم Luhn عمل می‌کند. پس از وارد شدن تمام داده‌ها، نتیجه اعتبارسنجی کارت به صورت یک سیگنال خروجی به نام valid_out نمایش داده می‌شود.

خروجی کد

ما این کد را با استفاده از یک فایل make و دستورات ghdl در vs code سنتز کرده و خروجی موج های آن را در gtkwave به نمایش گذاشتیم




File Edit Search Time Markers View Help
From: 0 sec
To: 1 us
Marker: – | Cursor: 323800 ps
SST
top
card_verifier_tb
uut
chunk_in
Signals
valid_out
enable
rst
clk
Time
Waves
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns


File Edit Search Time Markers View Help
From: 0 sec
To: 1 us
Marker: 507100 ps | Cursor: 2500 ps
SST
top
card_verifier_tb
uut
chunk_in
Signals
valid_out
enable
rst
clk
Time
Waves
100 ns
200 ns

منابع

https://www.aparat.com/v/y8813ym

https://www.aparat.com/v/x165rk6

https://faradars.org/courses/fvee9601-vhdl-programming-in-ise

https://faradars.org/courses/fvee9601s03-basic-concepts-of-vhdl

https://maktabkhooneh.org/course/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D9%85%D9%82%D8%AF%D9%85%D8%A7%D8%AA%DB%8Cactive-vhdl-mk252