به نام خدا

دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده مهندسی برق

پروژه نهایی VHDL درس مدارهای منطقی

دکتر محمدرضا پورفرد

مارال ترابی مهرام ۴۰۲۲۳۰۱۹

کیمیا خودسیانی ۴۰۲۲۳۰۳۰

مرداد ۱۴۰۴

فهرست مطالب

فصل اول – مقدمه

۱-۱. اهمیت تصحیح خطا در سیستم‌های دیجیتال

در دنیای دیجیتال، انتقال و ذخیره‌سازی داده‌ها همیشه با خطر خطا مواجه است. نویز در خطوط انتقال، خطاهای ناشی از نویزهای الکترومغناطیسی، خرابی‌های حافظه یا ناپایداری منابع تغذیه می‌توانند باعث تغییر بیت‌های داده شوند. اگر این خطاها بدون تشخیص باقی بمانند، می‌توانند منجر به خرابی سیستم، از بین رفتن اطلاعات و یا بروز عملکرد نادرست شوند. در همین راستا، استفاده از روش‌هایی برای تشخیص و حتی تصحیح خطا بسیار ضروری است، به‌ویژه در کاربردهایی مانند مخابرات، ذخیره‌سازی داده‌ها، سیستم‌های ایمن و کنترل صنعتی. کدهای تصحیح خطا (Error Correction Codes - ECC) ابزارهایی هستند که با افزودن افزونگی هوشمند به داده، امکان بازیابی داده اصلی حتی در حضور خطا را فراهم می‌سازند.

۱-۲. معرفی سیستم‌های ارسال/دریافت داده

سیستم‌های ارسال و دریافت داده، به عنوان اجزای کلیدی در ارتباطات دیجیتال، وظیفه انتقال مطمئن اطلاعات از یک منبع به یک مقصد را بر عهده دارند. این سیستم‌ها می‌توانند در قالب شبکه‌های کامپیوتری، پروتکل‌های سریال مانند UART یا SPI، یا سامانه‌های نهفته کاربرد داشته باشند. برای افزایش قابلیت اطمینان، استفاده از سازوکارهایی نظیر کدگذاری و رمزگشایی داده، بررسی صحت (Checksum) و ذخیره‌سازی موقت در حافظه ضروری است. در این پروژه، یک سیستم کامل ارسال و دریافت داده طراحی شده که در آن، اطلاعات ابتدا توسط یک ماژول Encoder به کد Hamming تبدیل شده، سپس ذخیره و پردازش می‌شود، و در نهایت با استفاده از Decoder بازسازی شده و صحت آن بررسی می‌شود.

۱-۳. هدف پروژه و کاربردهای آن

هدف این پروژه طراحی و پیاده‌سازی یک سیستم دیجیتال کامل در زبان VHDL است که بتواند یک داده ۸ بیتی را به‌صورت سریالی دریافت کرده، آن را به کد Hamming تبدیل کند، در RAM ذخیره نماید، روی آن عملیات منطقی/حسابی انجام دهد، سپس دوباره آن را به صورت کد شده ارسال یا در صورت نیاز بازیابی کند. کاربردهای چنین سیستمی در حوزه‌هایی نظیر مخابرات امن، پردازش داده در سامانه‌های توزیع‌شده، رابط‌های سریال صنعتی، و حافظه‌های با قابلیت تصحیح خطا بسیار گسترده است.

۱-۴. ساختار کلی سیستم طراحی‌شده

سیستم طراحی‌شده از چندین ماژول اصلی تشکیل شده است که به صورت سلسله‌وار و تحت کنترل یک واحد کنترلی مرکزی (Control Unit) با یکدیگر در ارتباط هستند .اجزای کلیدی عبارتند از:

Encoder (Hamming) : تولید کد ۱۳ بیتی از داده ۸ بیتی

Decoder (Hamming) : بازیابی داده و بررسی خطا

RAM : حافظه ۳۲×۸ بیتی برای ذخیره داده‌ها

ALU : انجام عملیات Add/Sub/OR/AND

Control Unit : مدیریت ترتیب اجرای عملیات

Packet Format : ساختار ارسال/دریافت داده با Checksum

این سیستم به گونه‌ای طراحی شده که بتواند پکت‌های مختلف با عملکردهای متفاوت (مثل Immediate، Operand، Array، Indirect) را پردازش کند.

فصل دوم – مبانی نظری و پایه‌ای

۲-۱. کدگذاری Hamming و کاربرد آن

کد Hamming روشی مؤثر برای تشخیص و تصحیح خطاهای تک‌بیتی است که با استفاده از بیت‌های توازن، موقعیت بیت خراب را شناسایی و در صورت امکان آن را اصلاح می‌کند. در کدHamming ، بیت‌های توازن در موقعیت‌هایی از داده قرار می‌گیرند که توان 2هستند (۱، ۲، ۴، ۸، ...). در این پروژه، از نسخه ۱۳ بیتی استفاده شده که شامل:

۸ بیت داده (در موقعیت‌های غیر توانی ۲)

۴ بیت توازن(P1 ، P2، P4، P8)

۱ بیت توازن کلی(Overall Parity - P_total)

مزیت مهم این روش، قابلیت تصحیح خطا به‌صورت کاملاً سخت‌افزاری با هزینه پایین منطقی است.

۲-۲ .قالب پکت (Packet) و اجزای آن

در این سیستم برای انتقال اطلاعات بین بخش‌های مختلف مانندEncoder ، RAM، ALUو Decoder از ساختار استانداردی به نام **پکت (Packet)** استفاده می‌شود.

پکت‌ها مانند بسته‌های اطلاعاتی هستند که فیلدهای مشخصی دارند و هر فیلد وظیفه‌ای خاص را بر عهده دارد. پکت‌ها بسته به نوع عملکردشان، ساختار متفاوتی دارند اما معمولاً شامل فیلدهای زیر هستند:

| Function | Address1 | Address2 / Data | Destination Address | Length | ChecksumH | ChecksumL |

- Function: تعیین می‌کند چه عملیاتی باید انجام شود (مثلاً جمع، نوشتن در حافظه، خواندن و ...)
- Address1 / Address2: آدرس‌هایی از حافظه برای استخراج یا ذخیره داده
- Data: در پکت‌هایی که Immediate هستند، این فیلد داده ورودی را مشخص می‌کند
- Destination Address: محل نهایی ذخیره‌سازی نتیجه
- Length: در عملیات‌هایی مثل Array ALU مشخص می‌کند عملیات باید روی چند خانه حافظه انجام شود
- ChecksumH / ChecksumL: بررسی صحت داده در طول انتقال

این طراحی انعطاف‌پذیری زیادی ایجاد می‌کند تا بتوان عملیات‌های مختلف را مانند موارد زیر انجام داد:

- انجام عملیات ALU با دو ورودی(Operand-based)

- انجام عملیات ALU با داده فوری (Immediate)
- پردازش آرایه‌ای در حافظه (Array)
- خواندن/نوشتن داده در حافظه
- استفاده از آدرس‌دهی غیرمستقیم (Indirect Addressing)

۲-۳. Checksum و روش محاسبه آن

برای اطمینان از صحت پکت دریافتی، از مکانیزم **Checksum** استفاده می‌شود.
در مرحله ارسال پکت، ابتدا مجموع تمام بایت‌های پکت (به‌جز فیلدهای Checksum خودش) محاسبه می‌شود. سپس این مقدار ۱۶ بیتی به دو قسمت ۸ بیتی تقسیم می‌شود:

- **CheckSumL**: ۸ بیت پایین‌تر مجموع
- **CheckSumH**: ۸ بیت بالاتر مجموع

در سمت گیرنده، همان جمع مجدداً انجام می‌شود و با مقادیر Checksum دریافتی مقایسه می‌گردد.
اگر مجموع با Checksum دریافتی برابر باشد، داده معتبر شناخته می‌شود. در غیر این صورت، سیگنال خطا (Error) فعال شده و اجرای عملیات متوقف می‌شود.
این کار باعث افزایش اطمینان در سیستم و جلوگیری از اجرای عملیات روی داده‌های خراب می‌شود.

۲-۴. معرفی ALU ، RAM و واحد کنترل (FSM)

- **ALU (واحد حساب و منطق)**:
  ALU بخشی از سیستم است که عملیات‌های منطقی و حسابی مانند جمع (Add) ، تفریق (Sub) ، یا (OR) و و (AND) را انجام می‌دهد.
  نوع عملیات از طریق فیلد Function موجود در پکت مشخص می‌شود و ALU بسته به نوع دستور، داده‌های مربوطه را از حافظه خوانده و عملیات را انجام می‌دهد.
- **RAM (حافظه موقت)**:
  یک حافظه با ظرفیت ۳۲ خانه ۸ بیتی است که داده‌ها در آن ذخیره می‌شوند.
  قابلیت خواندن و نوشتن دارد و در هنگام فعال شدن سیگنال Reset (Rst) ، تمام خانه‌های حافظه پاک‌سازی

می‌شوند.

خواندن و نوشتن داده در RAM با تأخیر مشخصی انجام می‌شود که در طراحی لحاظ شده.(Latency)

- **واحد کنترل–FSM (ماشین حالت متناهی):**

این ماژول نقش مغز سیستم را دارد FSM .با بررسی پکت دریافتی و سیگنال‌هایی مانند InputRdy ، Function وOutputRdy مشخص می‌کند در هر لحظه کدام بخش سیستم باید فعال شود.

برای مثال اگر پکتی با Function مربوط به ALU دریافت شود، FSMابتدا داده‌ها را از RAM خوانده، سپس ALU را فعال کرده و در نهایت خروجی را در مقصد ذخیره می‌کند.

این ماژول باعث می‌شود تمام اجزای سیستم به‌صورت هماهنگ و دقیق عمل کنند.

فصل سوم- طراحی و پیاده‌سازی سیستم

۳-۱. ماژول Hamming Encoder

هدف این ماژول این است که داده ۸ بیتی را به صورت سریالی دریافت کرده و پس از محاسبه بیت‌های توازن (Parity Bits)و یک بیت توازن کل(Overall Parity) ، آن را به صورت ۱۳ بیتی کد Hamming تبدیل کرده و سریالی ارسال نماید. این کد قابلیت تصحیح یک بیت خطا و تشخیص دو بیت خطا را دارد.

این ماژول دارای ورودی‌ها و خروجی‌های زیر است:

- Clk: سیگنال کلاک که وظیفه هماهنگی و زمان‌بندی بین مراحل مختلف را برعهده دارد.
- In_Start: سیگنالی برای مشخص کردن شروع دریافت داده. با فعال شدن آن، شمارش داده‌های ورودی آغاز می‌شود.
- In_Data: ورودی سریالی داده ۸ بیتی. در هر سیکل کلاک، یک بیت از داده وارد می‌شود.
- Rst: سیگنال ریست برای بازنشانی کلیه سیگنال‌های داخلی و شروع دوباره ماژول.
- Out_Data: خروجی سریالی ۱۳ بیتی حاصل از کدگذاری همینگ. در هر کلاک یک بیت از آن ارسال می‌شود.
- Out_Rdy: سیگنالی که نشان می‌دهد خروجی آماده است و داده در حال ارسال است.
- Open_In: سیگنالی دوطرفه که وضعیت ماژول را مشخص می‌کند. اگر مقدار آن ۱ باشد، ماژول در حالت دریافت داده است؛ اگر ۰ باشد، در حالت ارسال داده قرار دارد.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity HammingEncoder is
    Port ( clk : in STD_LOGIC;          -- Receiving sequential data
           In_Start : in STD_LOGIC;     -- Marking In_Data LSB
           In_Data : in  STD_LOGIC;     -- Input data
           Rst : in  STD_LOGIC;         -- Setting everything to the default
           Out_Data : out  STD_LOGIC;   -- Hamming coded data
           Out_Rdy : out  STD_LOGIC;    -- Is true if the output is calculated
           Open_In : inout  STD_LOGIC); -- No calculation is being done
end HammingEncoder;

architecture Behavioral of HammingEncoder is

        signal code : STD_LOGIC_VECTOR (12 downto 0);

```

درون معماری Behavioral این ماژول، یک process تعریف شده که با لبه بالارونده کلاک (rising_edge(clk))فعال شده و بخش‌های عملکردی آن به صورت زیر است:

```
begin

        process (clk, Rst, In_Start, Open_In)
                variable cnt : integer := 0;    -- cnt is 0 at the begining of simulation
        begin
                if rising_edge(clk) then
                        if (Rst = '1' or In_Start = '1') then
                                Out_Data <= '0';
                                Out_Rdy <= '0';
                                Open_In <= '1';
                                code <= "0000000000000";
                                cnt := 0;
                        end if;
                        if Open_In = '1' then   -- Input gate is open, we are in INPUT MODE
                                case cnt is
                                        when 0 =>
                                                code(2) <= In_Data;
                                        when 1 =>
                                                code(4) <= In_Data;
                                        when 2 =>
                                                code(5) <= In_Data;
                                        when 3 =>
                                                code(6) <= In_Data;
                                        when 4 =>
                                                code(7) <= In_Data;
                                        when 5 =>
                                                code(8) <= In_Data;
                                        when 6 =>
                                                code(9) <= In_Data;
                                        when 7 =>
                                                code(10) <= In_Data;
                                                cnt := 0;
                                                Open_In <= '0';
                                        when others =>
                                                -- Impossible!
                                end case;
```

۱. بررسی اولیه Reset

در ابتدای هر لبه‌ی کلاک، ابتدا بررسی می‌شود که آیا سیگنال Rst یا In_Start فعال است. در این صورت، ماژول به حالت اولیه باز می‌گردد.

این مرحله شامل ریست شدن متغیرها و سیگنال‌های داخلی مانند code، cntو Open_In است تا آماده دریافت داده جدید باشد.

۲. حالت دریافت داده – Input Mode (زمانی که Open_In = '1')

در این حالت، ماژول وارد مرحله دریافت داده به صورت سریالی می‌شود. ورودی In_Data در هر سیکل کلاک، یک بیت از داده را وارد می‌کند. با استفاده از متغیر شمارنده cnt، این بیت‌ها به ترتیب در موقعیت‌های خاصی از بردار codeقرار می‌گیرند.

جایگاه‌های بیت‌های داده به صورت زیر در بردار ۱۳ بیتی codeتعیین شده‌اند:

- $\text{cnt} = 0 \rightarrow \text{code}(2)$
- $\text{cnt} = 1 \rightarrow \text{code}(4)$
- $\text{cnt} = 2 \rightarrow \text{code}(5)$
- $\text{cnt} = 3 \rightarrow \text{code}(6)$
- $\text{cnt} = 4 \rightarrow \text{code}(7)$
- $\text{cnt} = 5 \rightarrow \text{code}(8)$
- $\text{cnt} = 6 \rightarrow \text{code}(9)$

- $cnt = 7 \rightarrow code(10)$

در اینجا پس از دریافت هشتمین بیت داده (در $cnt = 7$) و ثبت آن در $code(10)$، مقدار cnt ریست می‌شود ($cnt := 0$)و سیگنال $Open_In$ برابر با صفر می‌گردد تا ماژول وارد فاز ارسال شود.

۳. حالت ارسال داده – Output Mode (زمانی که $Open_In = '0'$)

در این مرحله، ابتدا بیت‌های توازن محاسبه شده و در جایگاه‌های مربوطه در بردار $code$ذخیره می‌شوند. بیت‌های توازن مطابق روابط استاندارد Hamming به شرح زیر هستند:

- $code(0)$ = XOR بیت‌های ۲، ۴، ۶، ۸، ۱۰ $(P1)$
- $code(1)$ = XOR بیت‌های ۲، ۵، ۶، ۹، ۱۰ $(P2)$
- $code(3)$ = XOR بیت‌های ۴، ۵، ۶، ۱۱ $(P4)$
- $code(7)$ =XOR بیت‌های ۸، ۹، ۱۰، ۱۱ $(P8)$

سپس، توازن کلی نیز به صورت XOR تمام بیت‌های $code(0)$تا $code(11)$محاسبه شده و در موقعیت $code(12)$ قرار می‌گیرد.

پس از آماده‌سازی بردار $code$، در هر سیکل کلاک، یکی از بیت‌های آن از طریق خروجی Out_Data ارسال می‌شود. شمارنده cnt از ۰ تا ۱۲ در این مرحله افزایش می‌یابد و برای هر مقدار از آن، یکی از بیت‌های $code$ خروجی داده می‌شود. در واقع فاز ارسال دقیقاً **۱۳ سیکل کلاک** طول می‌کشد.

۴. بازگشت به حالت دریافت

در زمانی که $cnt = 12$ و آخرین بیت خروجی ($code(12)$) ارسال می‌شود:

- شمارنده cnt به صفر ریست می‌شود.
- سیگنال $Open_In$ مجدداً برابر ۱ قرار می‌گیرد تا ماژول وارد مرحله‌ی دریافت داده جدید شود.
- سیگنال Out_Rdy نیز خاموش می‌شود.

۵. نکته کلیدی درباره چرخه دریافت و ارسال

چرخه دریافت و ارسال به‌صورت پشت‌سرهم انجام می‌شود. هر بار که ۸ بیت ورودی دریافت شد، بلافاصله ماژول وارد فاز ارسال ۱۳ بیت خروجی می‌شود. پس از اتمام ارسال، دوباره آماده دریافت سری بعدی داده‌ها خواهد بود.

رفتار سیگنال‌های Open_In و شمارنده cnt به گونه‌ای طراحی شده است که هماهنگی کامل بین دو فاز ورودی و خروجی فراهم شود و هیچ‌گونه همپوشانی یا ناسازگاری زمانی ایجاد نگردد.

```
                                cnt := cnt + 1;
                        elsif Open_In = '0' then   -- Input gate is closed, we are in OUTPUT MODE
                                -- Assigning parities
                                code(0) <= (((code(2) xor code(4)) xor code(6)) xor code(8)) xor code(10);
                                code(1) <= (((code(2) xor code(5)) xor code(6)) xor code(9)) xor code(10);
                                code(3) <= ((code(4) xor code(5)) xor code(6)) xor code(11);
                                code(7) <= ((code(8) xor code(9)) xor code(10)) xor code(11);
                                code(12) <= (((((code(0) xor code(1)) xor code(2)) xor code(3)) xor code(4)) xor code(5)) xor
                                                        (((((code(6) xor code(7)) xor code(8)) xor code(9)) xor code(10)) xor code(11));
                                -- Outputing
                                Out_Rdy <= '1';
                                case cnt is
                                        when 0 =>
                                                Out_Data <= code(0);
                                        when 1 =>
                                                Out_Data <= code(1);
                                        when 2 =>
                                                Out_Data <= code(2);
                                        when 3 =>
                                                Out_Data <= code(3);
                                        when 4 =>
                                                Out_Data <= code(4);
                                        when 5 =>
                                                Out_Data <= code(5);
                                        when 6 =>
                                                Out_Data <= code(6);
                                        when 7 =>
                                                Out_Data <= code(7);
                                        when 8 =>
                                                Out_Data <= code(8);
                                        when 9 =>
                                                Out_Data <= code(9);
                                        when 10 =>
                                                Out_Data <= code(10);
                                        when 11 =>
                                                Out_Data <= code(11);
                                        when 12 =>
                                                Out_Data <= code(12);
                                                cnt := 0;
                                                Open_In <= '1';
                                                Out_Rdy <= '0';
                                        when others =>
                                                -- Impossible!
                                end case;
                                cnt := cnt + 1;
                        end if;
                end if;
        end process;

end Behavioral;
```

۳-۲. ماژول Hamming Decoder

هدف این ماژول دریافت کد Hamming ۱۳ بیتی به‌صورت سریالی، بررسی صحت آن، شناسایی و تصحیح یک بیت خطا (در صورت وجود) و تولید خروجی ۸ بیتی تصحیح‌شده است. برخلاف نسخه قبلی، در این طراحی خروجی به‌صورت کامل (۸ بیت هم‌زمان) از طریق out_data ارائه می‌شود.

این ماژول دارای ورودی‌ها و خروجی‌های زیر است:

- **Clk**: سیگنال ساعت اصلی سیستم برای هماهنگی اجرای ترتیبی کد.
- **RST**: سیگنال ریست که باعث بازنشانی تمام سیگنال‌ها و شمارنده‌ها می‌شود.
- **in_start**: شروع دریافت پکت جدید را مشخص می‌کند؛ در لبه فعال آن کد آماده دریافت است.
- **in_data**: ورودی سریالی که بیت‌های کد Hamming به‌صورت یکی‌یکی وارد سیستم می‌شوند.
- **out_rdy**: خروجی دوطرفه که نشان می‌دهد خروجی محاسبه و آماده ارسال است.
- **out_data**: خروجی ۸ بیتی تصحیح‌شده، به‌صورت کامل و هم‌زمان ارائه می‌شود.
- **valid_out**: در صورتی که کد دریافتی حداکثر یک بیت خطا داشته باشد و تصحیح شده باشد، این سیگنال برابر با ۱ خواهد شد.

۱. بررسی اولیه Reset

در ابتدای هر لبه‌ی بالارونده‌ی کلاک، ابتدا بررسی می‌شود که آیا سیگنال‌های RST یا in_start فعال شده‌اند یا نه. در صورت فعال بودن هرکدام:

- مقدار cnt (شمارنده موقعیت بیت) به صفر بازنشانی می‌شود.
- بردار inp_enc_data که برای ذخیره ۱۳ بیت دریافتی به کار می‌رود، صفر می‌شود.
- سیگنال‌های خروجی valid_out و out_rdy خاموش می‌شوند.
- سیگنال out_data نیز به مقدار صفر تنظیم می‌گردد.

هدف از این بخش، آماده‌سازی کامل ماژول برای دریافت یک پکت جدید است.

۲. حالت دریافت داده (Input Mode)

بعد از ریست، ماژول وارد حالت دریافت داده می‌شود. در این مرحله:

- در هر لبه‌ی کلاک، یک بیت از ورودی in_data خوانده می‌شود و در یکی از موقعیت‌های inp_enc_data(cnt) قرار می‌گیرد.
- شمارنده cnt از ۰ تا ۱۲ افزایش می‌یابد.
- به ازای هر مقدار cnt، یکی از بیت‌های ۱۳تایی کد Hamming در بردار inp_enc_data ذخیره می‌شود.

در پایان این فاز (زمانی که $cnt = 13$):

- دریافت داده کامل شده و شمارنده cnt مجدداً صفر می‌شود.
- سیستم وارد مرحله بررسی و تصحیح می‌شود.

۳. حالت بررسی و تصحیح (Processing Mode)

- چهار بیت توازن داخلی با استفاده از XOR بین بیت‌های مشخص‌شده محاسبه شده و در marker(3 downto 0) قرار می‌گیرند.
- موقعیت احتمالی خطا با تبدیل marker به عدد صحیح (ind) به‌دست می‌آید.
- یک بیت توازن کلی (overall parity) با استفاده از XOR تمام بیت‌های ۰ تا ۱۱ محاسبه می‌شود و با بیت inp_enc_data(12) مقایسه می‌گردد.

شرایط تصمیم‌گیری به این صورت است:

- اگر $ind = -1$ (یعنی $marker = 0000$) و $extention = inp_enc_data(12)$ ← داده بدون خطاست.
- اگر $ind \neq -1$ و extention مخالف inp_enc_data(12) ← یک بیت خطا وجود دارد و اصلاح می‌شود.
- در سایر حالات ← داده معتبر نیست و valid_out برابر با صفر می‌ماند.

در صورت تشخیص داده‌ی معتبر:

- بیت خراب (در صورت وجود) اصلاح می‌شود.
- valid_out روشن می‌شود.
- out_rdy نیز برابر ۱ می‌شود و خروجی آماده است.

۴ .حالت تولید خروجی (Output Mode)

پس از بررسی و تصحیح داده:

- داده اصلی ۸ بیتی از بین ۱۳ بیت موجود استخراج شده و به ترتیب در out_dataقرار می‌گیرد.
- ترتیب قرارگیری بیت‌های داده اصلی در خروجی به صورت زیر است:

$$out_data = inp_enc_data(11)\ \&\ inp_enc_data(10)\ \&\ inp_enc_data(9)\&\ inp_enc_data(8)\ \&\ inp_enc_data(6)\ \&\ inp_enc_data(5)\ \&\ inp_enc_data(4)\ \&\ inp_enc_data(2);$$

این ترتیب دقیقاً همان است که در انکودر برای قرار دادن بیت‌های داده در موقعیت‌های خاص استفاده شده بود.

پس از تولید خروجی:

- ماژول آماده‌ی دریافت داده جدید خواهد بود.
- لازم است سیگنال in_start یا RST برای آغاز دوباره فاز دریافت فعال شوند.

۳-۳. فایل Packages

فایل Packages به‌عنوان یک کتابخانه پشتیبان، انواع داده و توابع موردنیاز ماژول‌های اصلی سیستم را تعریف می‌کند:

کاربرد	تعریف	نام نوع داده
یک واحد داده ۸ بیتی	STD_LOGIC_VECTOR(7 downto 0)	byte
پکت اصلی ۷بایتی	array (0 to 6) of byte	data_packet
برای دسته‌بندی دستورات	نوعی از عملیاتهاALU ، RAMو...	packet_type
حافظه اصلی سیستم	array (0 to 31) of byte	ram_matrix
پکت خروجی ازRAM	array (0 to 3) of byte	ram_resp_pack
برای ذخیره موقتی داده‌ها	بافر خواندن در عملیات آرایه‌ایALU	alu_read_cash_array

این فایل شامل چند تابع مهم نیز می‌شود:

- ByteSum: مجموع ۵ بایت اول پکت را محاسبه می‌کند و خروجی ۱۶ بیتی می‌دهد.
- CheckSumH: ۸ بیت بالایی حاصل جمع را برمی‌گرداند.
- CheckSumL: ۸ بیت پایینی حاصل جمع را برمی‌گرداند.
- Validate: بررسی می‌کند که آیا مقادیر Checksum ذخیره‌شده در پکت با مقدار واقعی جمع مطابقت دارد یا نه.

۴-۳. ماژول Control Unit

هدف ماژول ControlUnit این است که داده‌های ۸ بیتی ورودی را دریافت کرده، آن‌ها را به پکت ۷ بایتی تبدیل می‌کند، نوع آن را تشخیص می‌دهد و مسیر مناسب پردازش(RAMیاALU) را مشخص می‌سازد. این ماژول در کنار فایل Packages پایه‌گذار مرحله‌ی «تحلیل ورودی» در سیستم هستند.

ورودی‌ها:

توضیح	نوع	سیگنال

InByte	byte	داده دریافتی از دیکودر
clk	STD_LOGIC	سیگنال ساعت
RST	STD_LOGIC	ریست سیستم

خروجی‌ها:

سیگنال	نوع	توضیح
Packet	data_packet	پکت نهایی ۷ بایتی
PackMode	packet_type	نوع عملیات مشخص‌شده
Switch	STD_LOGIC	تعیین مسیر RAM (0) یا ALU (1)
Validation	STD_LOGIC	مشخص می‌کند که پکت ورودی معتبر بوده یا نه

نحوه عملکرد ماژول Control Unit:

در این ماژول داده‌های ورودی به صورت بایت‌های ۸ بیتی به ماژول وارد می‌شوند. در هر سیکل کلاک یک بایت جدید دریافت شده و داخل آرایه PackHold ذخیره می‌شود. به محض دریافت اولین بایت، نوع پکت با بررسی مقدار آن تعیین می‌شود. در ادامه بسته به نوع پکت، تعدادی بایت دیگر نیز دریافت می‌شود.

اگر نوع پکت از نوع Rea_d باشد، پس از دریافت دو بایت ورودی، دریافت متوقف شده و ماژول بلافاصله وارد مرحله پردازش می‌شود. برای پکت‌های Writ_e این مقدار سه بایت است.

در صورتی که پکت از نوع ALU باشد (Immediate، Operand یا Indirect)، دریافت تا چهار بایت ادامه می‌یابد. برای حالت Array_Alu پنج بایت مورد نیاز است. بعد از آن نیز دو بایت برای Checksum دریافت شده و پکت تکمیل می‌شود. در انتهای مرحله دریافت، ماژول با استفاده از تابع Validate، صحت پکت را بررسی کرده و نتیجه آن را از طریق سیگنال Validation گزارش می‌دهد.

پکت نهایی نیز از طریق خروجی Packet به ماژول‌های بعدی ارسال می‌شود. در کنار آن، سیگنال PackMode نوع پکت را مشخص می‌کند و سیگنال Switch مسیر جریان داده را تعیین می‌کند. اگر مقدار Switch صفر باشد، پکت برای RAM ارسال می‌شود. اگر مقدار آن یک باشد، پکت به ALU هدایت خواهد شد.

۳-۵. ماژول RAM

ماژول RAM به‌عنوان حافظه اصلی سیستم، وظیفه‌ی **ذخیره و بازیابی داده‌ها** را دارد. این ماژول می‌تواند بسته به نوع عملکرد، یک مقدار مشخص را در یک آدرس مشخص از حافظه بنویسد یا از آن بخواند. همچنین در فرآیند خواندن، پاسخ را به‌صورت یک **پکت استاندارد ۷ بایتی** به همراه Checksum باز می‌گرداند تا در مراحل بعدی مورد استفاده قرار گیرد.

ورودی‌ها و خروجی‌ها:

این ماژول دارای دو پکت ورودی است؛ یکی از طرف **کنترل یونیت (CtrlReq)** و دیگری از طرف **واحد ALU (AluReq)**. یک سیگنال کنترلی به نام InChoose تعیین می‌کند که کدام یک از این دو ورودی باید در سیکل کلاک فعلی بررسی و پردازش شود.

همچنین این ماژول دارای خروجی‌های زیر است:

- یک پکت خروجی استاندارد (ReadResp) که در صورت اجرای عملیات خواندن از RAM، حاوی داده برگشتی است.
- یک سیگنال خطا (Error) که در صورت آدرس خارج از محدوده یا عملکرد نامعتبر فعال می‌شود.

ساختار حافظه:

درون ماژول، ساختاری به نام Memory تعریف شده است که یک آرایه ۳۲ خانه‌ای از نوع byte (یعنی ۸ بیتی) است. به هر خانه از این حافظه از طریق آدرس‌دهی مستقیم با مقدار InPack(1) دسترسی پیدا می‌شود.

نحوه عملکرد ماژول RAM:

در هر لبه بالارونده کلاک:

۱. **اگر سیگنال Reset فعال باشد،** تمام خانه‌های حافظه با صفر مقداردهی مجدد می‌شوند و سیستم به حالت اولیه باز می‌گردد.

۲. **در حالت عادی:**

- ابتدا مشخص می‌شود که ورودی از طرف ALU خوانده شود یا از طرف ControlUnit، و در نتیجه InPack با مقدار مناسب مقداردهی می‌شود.

- بررسی می‌شود که آیا بایت اول InPack(0) معادل "00001111" (برای Read) یا "11110000"(برای Write) است.
- اگر نباشد، نوع عملکرد zero تلقی شده و سیگنال Error فعال می‌شود.

اگر نوع عملکرد Writ_e باشد:

- آدرس از InPack(1) استخراج شده و به عدد صحیح بین ۰ تا ۳۱ تبدیل می‌شود.
- مقدار InPack(2) در خانه مشخص‌شده از حافظه نوشته می‌شود.
- اگر آدرس از ۳۱ بیشتر باشد، عملیات انجام نمی‌شود و Error فعال می‌شود.

اگر نوع عملکرد Rea_d باشد:

- داده‌ی ذخیره‌شده در آدرس مورد نظر از حافظه استخراج می‌شود.
- سپس یک پکت خروجی ۷ بایتی ساخته می‌شود:
  ۱. بایت اول برابر "11001111"به‌عنوان نشانه‌ی پاسخ حافظه
  ۲. بایت دوم مقدار خوانده‌شده از حافظه
  ۳. سه بایت میانی cash(2) تا cash(4) با صفر مقداردهی می‌شوند
  ۴. سپس Checksum بالا (cash(5)) و پایین (cash(6)) محاسبه می‌شود.
- این پکت در خروجی ReadResp قرار می‌گیرد تا به ماژول‌های بعدی منتقل شود.

Checksumها:

برای تضمین صحت داده‌های خروجی از حافظه، از توابع CheckSumH و CheckSumL که در فایل Packages تعریف شده‌اند استفاده می‌شود. این توابع مجموع ۵ بایت اول پکت را محاسبه کرده و به‌ترتیب ۸ بیت بالا و پایین آن را در بایت‌های ۵ و ۶ ذخیره می‌کنند.

۳-۶. ماژول ALU و Error Detection

هدف ماژول ALU این است که عملیات‌های منطقی و حسابی متنوعی مانند **جمع، تفریق، AND و OR** را روی داده‌هایی که از حافظه خوانده شده‌اند یا در پکت ورودی موجودند، انجام دهد و نتیجه را به حافظه برگرداند. این ماژول به‌طور مستقیم به RAM متصل است و بسته به نوع عملکرد پکت، از حافظه می‌خواند، عملیات را انجام می‌دهد و نتیجه را ذخیره می‌کند.

در کنار ALU ، ماژول کوچکی به نام **ErrorDetection** نیز تعریف شده که وظیفه ترکیب و گزارش نهایی هرگونه خطا در سیستم را بر عهده دارد. این خطاها می‌توانند از دیکودر، کنترل یونیت، RAM یا ALU منشأ گرفته باشند.

ورودی‌ها و خروجی‌های ALU

ورودی‌ها:

- InPack: پکت دریافتی از کنترل یونیت
- PackMode: نوع عملکرد پکت (مانند Operand_Alu ، Immediate_Alu و...)
- ReadResponse: داده‌هایی که از RAM برای خواندن دریافت شده‌اند
- Enable: فعال‌سازی ALU فقط زمانی که پکت مربوط به ALU باشد
- RST: ریست تمام مقادیر داخلی
- Clk: سیگنال ساعت

خروجی‌ها:

- SentToRam: پکت ۷بایتی خروجی که برای نوشتن به حافظه ارسال می‌شود
- Finish: سیگنالی برای اعلام پایان عملیات
- Error: سیگنال گزارش خطا

نحوه عملکرد ماژول ALU:

ماژول ALU دارای حالت‌های مختلف است که بسته به نوع پکت در PackMode رفتار متفاوتی از خود نشان می‌دهد. در همه حالت‌ها، روند کلی به شکل زیر است:

۱. بررسی نوع عملیات بر اساس ۲ بیت آخر InPack(0)
۲. استخراج آدرس‌ها، داده‌ها یا پارامترها از پکت
۳. خواندن داده‌ها از RAM از طریق ReadResponse
۴. انجام عملیات منطقی/ریاضی
۵. ساخت پکت خروجی و ارسال آن برای نوشتن درRAM
۶. گزارش پایان عملیات از طریق سیگنال Finish

۱.عملیات با دو داده حافظه (Operand_Alu)

- از دو آدرس InPack(1) و InPack(2) داده‌ها را از حافظه می‌خواند.
- آن‌ها را با عملیات مشخص‌شده ترکیب کرده و نتیجه را در آدرس InPack(3) می‌نویسد.
- در این حالت عملیات طی **۳ سیکل کلاک** انجام می‌شود.

۲.عملیات با داده ثابت (Immediate_Alu)

- یکی از داده‌ها مستقیماً در پکت (InPack(2)) قرار دارد.
- داده دیگر از حافظه خوانده می‌شود.
- نتیجه عملیات در آدرس InPack(3)ذخیره می‌شود.
- طی **۲ سیکل کلاک** انجام می‌شود.

۳.عملیات روی آرایه از حافظه (Array_Alu)

- از آدرس InPack(1) داده‌ها را پشت سر هم می‌خواند.
- داده دوم ثابت است (InPack(2)) .

- به تعداد InPack(3) خانه حافظه پیمایش و روی آن عملیات انجام می‌شود.
- نتایج در خانه‌های پشت سر هم از InPack(4) ذخیره می‌شوند.
- اگر طول آرایه بیشتر از ۳۲ باشد، سیگنال Error فعال می‌شود.

۴.آدرس غیر مستقیم (Indirect_Addressing)

- ابتدا از InPack(1) داده‌ای خوانده می‌شود.
- سپس از InPack(2) آدرسی خوانده می‌شود که حاوی آدرس دوم واقعی است.
- از این آدرس دوم داده خوانده شده و عملیات انجام می‌شود.
- نتیجه در InPack(3) ذخیره می‌شود.

نتیجه هر عملیات داخل متغیر Output ذخیره شده و به‌همراه اطلاعات آدرس و نوع عملیات در قالب یک پکت کامل به نام SentToRamPack ساخته می‌شود. در پایان، مقادیر Checksum با صفر مقداردهی شده و پکت خروجی نهایی از طریق خروجی SentToRam ارسال می‌شود.

مدیریت خطا

در موارد زیر سیگنال Error برابر با ۱ قرار می‌گیرد:

- فعال بودن RST
- نوع پکت ناصحیح یا غیرقابل تشخیص
- طول آرایه بزرگ‌تر از ۳۲
- اشکال در آدرس‌دهی غیرمستقیم یا پردازش داخلی

ماژول بسیار ساده‌ای به نام ErrorDetection در پروژه تعریف شده که تمامی سیگنال‌های خطا از ماژول‌های مختلف شامل دیکودر، کنترل یونیت، RAMو ALU را دریافت کرده و در صورتی که هرکدام از آن‌ها فعال باشند، سیگنال نهایی Error را روشن می‌کند. فرمول منطقی این کار:

Error <= (RamError or AluError) or (DecodingError or PacketError)

به این ترتیب، سیستم همیشه از وقوع خطا در هر یک از بخش‌ها مطلع می‌شود.

۳-۷. ماژول PackToByte

هدف ماژول PackToByte این است که داده‌های ۷ بایتی موجود در خروجی RAM (یعنی یک پکت کامل data_packet) را به‌صورت بایت‌به‌بایت استخراج کرده و آن‌ها را برای ارسال به انکودر آماده کند. این فرآیند شامل سازمان‌دهی مجدد پکت به صورت ram_resp_pack است که تنها شامل ۴ بایت است.

ورودی PackIn: پکت ۷بایتی دریافتی از RAM که شامل Function ، داده، و Checksum است.

خروجی ByteOut: بایتی که باید به صورت سریالی برای تبدیل به بیت به ماژول بعدی ارسال شود.

سیگنال clk: سیگنال کلاک برای همگام‌سازی عملیات.

در ابتدا، ماژول یک آرایه کمکی به نام PacketCash از نوع ram_resp_pack(۴ بایتی) را مقداردهی اولیه می‌کند:

- PacketCash(0)برابر "11001111" (کد شناسه پاسخ خواندن از RAM)
- PacketCash(2)برابر "00000000" (رزرو یا مقدار صفر)

سپس در لبه‌ی بالارونده کلاک، با استفاده از یک شمارنده داخلی CellCnt، در هر سیکل یکی از این بایت‌ها را در خروجی قرار می‌دهد.

- در اولین سیکل (CellCnt = 4) داده‌های اصلی از PackIn(1) (داده RAM) و PackIn(6) (CheckSum) به آرایه PacketCash منتقل می‌شوند.
- سپس از CellCnt = 1 تا CellCnt = 3، خروجی‌ها به ترتیب از PacketCash خوانده شده و در ByteOutقرار می‌گیرند.

۳-۸. ماژول ByteToBit

هدف این ماژول این است که یک بایت (۸ بیت) را به‌صورت سریالی و بیت‌به‌بیت در هر سیکل کلاک روی خروجی قرار دهد تا مستقیماً به انکودر همینگ ارسال شود.

- **ورودی ByteIn**: بایت ۸ بیتی که باید به بیت‌های مجزا تقسیم شود.
- **خروجی BitOut**: خروجی سریالی که در هر کلاک یکی از بیت‌های ByteInرا تولید می‌کند.
- **سیگنال clk**: برای همگام‌سازی عملیات.

نحوه عملکرد ماژول:

- در ابتدا، یک سیگنال کمکی ByteCash مقدار بایت ورودی را ذخیره می‌کند.
- شمارنده داخلی BitCnt مشخص می‌کند که کدام بیت از ByteCash باید در BitOut قرار گیرد.
  - وقتی $BitCnt = 8$باشد (یعنی ابتدای دریافت بایت جدید)، ByteCash مقدار جدیدی می‌گیرد و بیت صفرم آن در BitOut قرار می‌گیرد.
  - در سیکل‌های بعدی (تا $BitCnt = 7$) بیت‌های باقی‌مانده از ByteCash به ترتیب در خروجی قرار می‌گیرند.

ارتباط دو ماژول PackToByte و ByteToBit

۱. PackToByte خروجی RAM را به ترتیب در بایت‌های مجزا قرار می‌دهد.

۲. ByteToBitهر بایت را به ۸ بیت جداگانه تبدیل کرده و برای رمزگذاری Hamming آماده می‌کند.

این دو ماژول در کنار هم، وظیفه دارند داده ۷ بایتی خروجی RAM را به‌صورت سریالی و بیت‌به‌بیت به Hamming Encoderتحویل دهند، تا برای ارسال در سیستم ارتباطی مورد استفاده قرار گیرد.

۳-۹. تاپ ماژول Top Module

ماژول TopModuleبه‌عنوان **واحد تجمیع‌کننده**، تمام بخش‌های طراحی‌شده در سیستم را به یکدیگر متصل می‌کند و فرآیند کلی ارسال و دریافت داده به همراه پردازش منطقی را مدیریت می‌نماید. این ماژول مانند اسکلت اصلی سیستم عمل کرده و ارتباط بین ماژول‌های مختلف را هماهنگ می‌کند.

ورودی‌ها و خروجی‌ها

- Input: ورودی سریالی ۱ بیتی که داده رمزگذاری‌شده (Hamming) را دریافت می‌کند.
- Output: خروجی سریالی ۱ بیتی که داده رمزگذاری‌شده نهایی را ارسال می‌کند.
- Error: سیگنالی برای نشان‌دادن وقوع خطا در هر مرحله از فرآیند.
- RST: سیگنال Reset سراسری برای بازنشانی کل سیستم.
- Clk: سیگنال کلاک برای زمان‌بندی ماژول‌ها.

نحوه عملکرد

ماژول اصلی شامل ۸ ماژول زیر است که به ترتیب متصل شده‌اند و با سیگنال‌های میانی با یکدیگر تعامل دارند:

۱.HammingDecoder

- ورودی سریالی Input را دریافت می‌کند.
- پس از بررسی توازن و تصحیح احتمالی خطا، داده ۸ بیتی خروجی می‌دهد.
- خروجی:
  - DataByte: بایت رمزگشایی‌شده
  - DecValidation: اعتبار داده پس از رمزگشایی

۲. ControlUnit

- داده ۸ بیتی را از DataByte می‌گیرد.
- بسته (Packet) متشکل از چند بایت ساخته و نوع عملکرد آن(ALU یا RAM) را مشخص می‌کند.

- خروجی‌ها:
  - Packet: پکت ساخته‌شده
  - PackMode: نوع پکت (خواندن، نوشتن، عملیات منطقی و ...)
  - Switch: مشخص‌کننده ALU mode یاRAM mode
  - PacketValidation: اعتبار پکت ساخته‌شده) بر اساس(Checksum

۳. ALU (Arithmetic Logic Unit)

- اگر نوع پکت نیازمند عملیات منطقی باشد، فعال می‌شود.
- داده را از RAM می‌گیرد و پس از پردازش، نتیجه را برای نوشتن به RAM آماده می‌کند.
- خروجی‌ها:
  - AluToRam: پکت خروجی شامل نتیجه عملیات
  - AluDone: اتمام عملیات
  - AluError: اگر مشکلی رخ دهد(مثلاً overflow یا دسترسی غیرمجاز)، فعال می‌شود.

۴. RAM

- بسته به Switch، بین ورودی‌های CtrlReq (از ControlUnit) یا AluReq (از ALU) یکی را انتخاب می‌کند.
- اگر عملکرد نوشتن باشد، داده را در حافظه ذخیره می‌کند.
- اگر عملکرد خواندن باشد، داده را از حافظه بازیابی کرده و در قالب پکت ۷بایتی خروجی می‌دهد.
- خروجی‌ها:
  - RamReadResp: پکت حاوی پاسخ خواندن ازRAM
  - RamError: خطای دسترسی به حافظه یا Checksum نادرست

۵. PackToByte

- پکت ۷بایتی دریافتی از RAM را به بایت‌های مجزا تقسیم می‌کند.
- هر بار، یک بایت را در خروجی قرار می‌دهد.

۶. ByteToBit

- بایت خروجی را به بیت‌های مجزا (از بیت ۰ تا ۷) در هر سیکل کلاک تقسیم می‌کند.
- خروجی سریالی تولید می‌کند.

۷. HammingEncoder

- بیت‌های ورودی را دریافت کرده و با محاسبه بیت‌های توازن(Parity) ، آن را به کد Hamming ۱۳ بیتی تبدیل کرده و به صورت سریالی ارسال می‌کند.
- خروجی نهایی از طریق Output خارج می‌شود.

۸. ErrorDetection

- این ماژول بررسی می‌کند آیا در بخش‌های مختلف سیستم خطایی رخ داده است یا خیر:
  - خطای رمزگشایی(DecValidation)
  - خطای صحت پکت(PacketValidation)
  - خطای RAM یاALU
- اگر هر کدام از این خطاها فعال باشند، سیگنال Errorبرابر ۱ می‌شود.

مدیریت حالت ALU

- سیگنال AluEnable تعیین می‌کند که ماژول ALU فعال باشد یا نه.
- با استفاده از: AluEnable <= Switch or (not(AluDone));

این شرط باعث می‌شود ALU تنها زمانی فعال شود که یا در ALU mode باشیم (Switch = 1) یا عملیات قبلی هنوز به اتمام نرسیده باشد.(AluDone = 0)

ماژول TopModule در واقع سیستم کامل انتقال داده رمزگذاری‌شده است که:

۱. داده سریالی را دریافت می‌کند،

۲. آن را رمزگشایی کرده،

۳. در صورت نیاز عملیات منطقی انجام می‌دهد یا در حافظه ثبت می‌کند،

۴. خروجی را مجدداً رمزگذاری کرده و سریالی ارسال می‌کند.

هم‌زمان با این فرآیند، کلیه خطاها به‌صورت سراسری تحت نظارت قرار دارند و سیگنال Error در صورت وقوع مشکل فعال می‌شود.