این فایل شامل تعریف ماژول رجیستر داده است که برای ذخیره‌سازی و بارگذاری داده‌ها استفاده می‌شود.

```
module final_project (
    input clk,
    input clr,
    input ld,
    input inr,
    input [15:0] data_in,
    output reg [15:0] ac
);
```

module final_project: تعریف ماژول با نام **final_project** .

input clk: سیگنال ساعت (clock) که ورودی مدار است.

input clr: سیگنال پاک‌سازی (clear) که ورودی مدار است و برای ریست کردن مقدار **ac** به کار می‌رود.

input ld: سیگنال بارگذاری (load) که ورودی مدار است و برای بارگذاری مقدار **data_in** به **ac** استفاده می‌شود.

input inr: سیگنال افزایش (increment) که ورودی مدار است و برای افزایش مقدار **ac** به کار می‌رود.

input [15:0] data_in: ورودی 16 بیتی که داده‌ای را که باید در **ac** بارگذاری شود، نگهداری می‌کند.

output reg [15:0] ac: خروجی 16 بیتی که مقدار انباشته (accumulator) را نگهداری می‌کند.

```
reg prev_clk;
```

reg prev_clk: یک رجیستر داخلی برای نگهداری وضعیت قبلی سیگنال ساعت (**clk**). این رجیستر برای شبیه‌سازی لبه بالارونده ساعت به کار می‌رود.

```
10
11      always @(*) begin
12          if (clr)
13              ac <= 16'h0000;
14          else if (clk && !prev_clk) begin
15              if (ld)
16                  ac <= data_in;
17              else if (inr)
18                  ac <= ac + 1;
19          end
20      end
21
```

بلوک اول always

این بلوک برای به‌روزرسانی مقدار **ac** استفاده می‌شود.

شروع بلوک `always @(*)` :

این بلوک به تمامی سیگنال‌ها حساس است، یعنی با هر تغییر در هر یک از سیگنال‌های ورودی (`clk` , `data_in` , `inr` , `ld` , `clr`)، این بلوک دوباره ارزیابی می‌شود.

بررسی سیگنال `clr` :

اگر سیگنال `clr` فعال باشد (`clr == 1`)، مقدار `ac` به `h0000'16` تنظیم می‌شود. این کار بدون توجه به سایر سیگنال‌ها انجام می‌شود.

بررسی وضعیت `clk` و `prev_clk` :

اگر `clr` غیر فعال باشد (`clr == 0`) و سیگنال `clk` بالا باشد (`clk == 1`) و `prev_clk` پایین باشد (`prev_clk == 0`)، این شرط درست است. این شرط عملاً بررسی می‌کند که آیا `clk` از `0` به `1` تغییر کرده است (لبه بالا رونده کلاک).

بررسی سیگنال `ld` :

اگر شرط لبه بالا رونده کلاک درست باشد و سیگنال `ld` فعال باشد (`ld == 1`)، مقدار `ac` به مقدار `data_in` تنظیم می‌شود.

بررسی سیگنال `inr` :

اگر شرط لبه بالا رونده کلاک درست باشد و سیگنال `ld` غیر فعال باشد (`ld == 0`) و سیگنال `inr` فعال باشد (`inr == 1`)، مقدار `ac` یک واحد افزایش می‌یابد (`ac <= ac + 1`).

```
21
22      always @(*) begin
23          if (clr)
24              prev_clk <= 0;
25          else
26              prev_clk <= clk;
27      end
28  endmodule
29
```

بلوک دوم `always`

این بلوک برای به‌روزرسانی مقدار `prev_clk` استفاده می‌شود.

شروع بلوک `always @(*)` :

این بلوک نیز به تمامی سیگنال‌ها حساس است، یعنی با هر تغییر در هر یک از سیگنال‌های ورودی (`data_in` , `inr` , `ld` , `clr` , `clk`)، این بلوک دوباره ارزیابی می‌شود.

بررسی سیگنال `clr` :

اگر سیگنال `clr` فعال باشد (`clr == 1`)، مقدار `prev_clr` به `0` تنظیم می‌شود. این کار بدون توجه به سایر سیگنال‌ها انجام می‌شود.

بررسی سیگنال `clk` :

اگر سیگنال `clr` غیر فعال باشد (`clr == 0`)، مقدار `prev_clr` به مقدار `clk` تنظیم می‌شود.

نحوه عملکرد کلی

ریست کردن (`clr`):

وقتی `clr` فعال شود، هر دو رجیستر (`ac` و `prev_clk`) به `0` تنظیم می‌شوند.

لبه بالا رونده کلاک:

وقتی `clr` غیر فعال باشد و سیگنال `clk` از `0` به `1` تغییر کند (لبه بالا رونده)، بلوک اول بررسی می‌کند که آیا `ld` یا `inr` فعال هستند:

اگر `ld` فعال باشد، مقدار `ac` به `data_in` تنظیم می‌شود.

اگر `ld` غیر فعال و `inr` فعال باشد، مقدار `ac` یک واحد افزایش می‌یابد.

در همین زمان، بلوک دوم مقدار `prev_clk` را به `clk` تنظیم می‌کند تا بتواند لبه کلاک را در چرخه بعدی تشخیص دهد.

Testbench:

```
1   `include "final_project.v"
2   `timescale 1ns / 1ps
```

`include` فایل `final_project.v` را وارد می‌کند که شامل تعریف ماژول `final_project` است.

`timescale` مقیاس زمانی شبیه‌سازی را تنظیم می‌کند. در اینجا، 1 نانوثانیه واحد زمانی است و دقت زمانی 1 پیکوثانیه است. `It stands for 1 picosecond, or 1 trillionth of a second.`

```
module testbench;
    reg clk;
    reg clr;
    reg ld;
    reg inr;
    reg [15:0] data_in;
    wire [15:0] ac;
```

متغیرهای **clk** , **clr** , **ld** , **inr** به صورت رجیستر تعریف شده‌اند که به عنوان سیگنال‌های ورودی به ماژول **final_project** استفاده می‌شوند.

data_in یک رجیستر 16 بیتی است که به عنوان ورودی داده استفاده می‌شود.

ac یک **Wire** 16 بیتی است که خروجی ماژول **final_project** را نشان می‌دهد.

```
    final_project dut (
        .clk(clk),
        .clr(clr),
        .ld(ld),
        .inr(inr),
        .data_in(data_in),
        .ac(ac)
    );
```

final_project dut نمونه‌ای از ماژول **final_project** به نام `dut` (Device Under Test) ایجاد می‌کند و پورت‌های آن را به متغیرهای تعریف شده در تست‌بنچ متصل می‌کند.

```
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
```

بلوک **initial** یک بلوک ابتدایی است که تنها یک بار در شروع شبیه‌سازی اجرا می‌شود.

clk = 0; مقدار اولیه **clk** را 0 تنظیم می‌کند.

دستور **forever #5 clk = ~clk;** یک کلاک با دوره 10 نانوثانیه تولید می‌کند که هر 5 نانوثانیه تغییر می‌کند.

```
    initial begin

        clr = 0;
        ld = 0;
        inr = 0;
        data_in = 16'h0000;


        #5;


        clr = 1;
        #10;
        clr = 0;
        #10;
        $display("After clear: ac = %h", ac);
```

در این بلوک `initial` ، مقادیر اولیه سیگنال‌ها تنظیم می‌شوند.

`data_in = 16'h0000; , ld = 0; , inr = 0; , clr = 0;` مقادیر اولیه سیگنال‌ها را تنظیم می‌کنند.

`#5;` 5 نانوثانیه منتظر می‌ماند.

`clr = 1; #10; clr = 0; #10;` سیگنال `clr` را برای 10 نانوثانیه فعال کرده و سپس غیرفعال می‌کند و 10 نانوثانیه منتظر می‌ماند.

`$display("After clear: ac = %h", ac);` مقدار `ac` را پس از پاک‌سازی نمایش می‌دهد.

```
    ld = 1;
    data_in = 16'h00FF;
    #10;
    ld = 0;
    #10;
    $display("After load: ac = %h", ac);


    inr = 1;
    #10;
    inr = 0;
    #10;
    $display("After increment: ac = %h", ac);
```

`ld = 1; data_in = 16'h00FF;` سیگنال لود (`ld`) را فعال کرده و مقدار `data_in` را تنظیم می‌کند.

`#10; ld = 0; #10;` سیگنال لود را غیرفعال کرده و 10 نانوثانیه منتظر می‌ماند.

`$display("After load: ac = %h", ac);` مقدار `ac` را پس از لود کردن داده نمایش می‌دهد.

`inr = 1; #10; inr = 0; #10;` سیگنال افزایش (`inr`) را فعال کرده، 10 نانوثانیه منتظر می‌ماند و سپس غیرفعال می‌کند.

`$display("After increment: ac = %h", ac);` مقدار `ac` را پس از افزایش نمایش می‌دهد.

```
61
62          ld = 1;
63          data_in = 16'h0F0F;
64          #10;
65          ld = 0;
66          #10;
67          $display("After second load: ac = %h", ac);
68
69
70          inr = 1;
71          #10;
72          inr = 0;
73          #10;
74          $display("After second increment: ac = %h", ac);
75
76
77          clr = 1;
78          #10;
79          clr = 0;
80          #10;
81          $display("After second clear: ac = %h", ac);
82
83
84          #100 $finish;
85      end
```

همانند مراحل قبل، مقادیر سیگنال‌ها تنظیم شده و تست می‌شوند اما با داده‌ها و دستورات جدید.

در پایان، **`#100 $finish;`** شبیه‌سازی را پس از 100 نانوثانیه متوقف می‌کند.

```
87
88      initial begin
89          $dumpfile("final_project.vcd");
90          $dumpvars(0, testbench);
91      end
```

این بلوک **`initial`** یک فایل (**`final_project.vcd`**) VCD برای ذخیره‌سازی موج‌های شبیه‌سازی تولید می‌کند.

`$dumpvars(0, testbench);` تمام متغیرهای ماژول **`testbench`** را در فایل VCD ثبت می‌کند.