به نام خدا

پروژه پایانی درس اجزا کامپیوتر

نام استاد : دکتر شاه حسینی

دانشجویان:

مهدیس رژه 400412193

پریا تیموریان 400411315

آرین احمدی 400411027

تاریخ : ۱۴۰۳/۱۱

روش‌های پیاده‌سازی ضرب‌کننده‌ها

1.ضرب‌کننده دودویی(Binary Multiplier) :

این روش بر اساس اصول پایه‌ای ضرب دودویی کار می‌کند.

- **روش کار:**
  - اعداد به صورت دودویی در نظر گرفته می‌شوند.
  - هر بیت از یکی از اعداد (مضروب) با بیت‌های عدد دیگر (ضارب) ضرب شده و نتایج در مکان‌های مناسب جمع می‌شوند.
- **ویژگی‌ها:**
  - ساده‌ترین روش پیاده‌سازی.
  - به مدارهای جمع‌کننده نیاز دارد.
  - سرعت پایین‌تر نسبت به سایر روش‌ها (وابسته به تعداد بیت‌ها).

```
//Binary multiplier
module Multiplier (
    input wire [7:0] operand_1,
    input wire [7:0] operand_2,
    output reg [15:0] product
);
    integer i;
    always @(*) begin
        product = 16'b0;
        for (i = 0; i < 8; i = i + 1) begin
            if (operand_2[i]) begin
                product = product + (operand_1 << i);
            end
        end
    end
endmodule
```

2.ضرب‌کننده آرایه‌ای(Array Multiplier) :

این روش با استفاده از یک آرایه منظم از جمع‌کننده‌ها و گیت‌های AND پیاده‌سازی می‌شود.

- **روش کار:**
  - هر بیت از یکی از ورودی‌ها با هر بیت از ورودی دیگر ضرب می‌شود) با استفاده از گیت AND).
  - نتایج به صورت آرایه‌ای مرتب شده و با استفاده از مدارهای جمع‌کننده جمع می‌شوند.
- **ویژگی‌ها:**
  - طراحی سخت‌افزاری ساده و منظم.
  - سرعت نسبتاً بالاتر به دلیل عملیات موازی.
  - نیاز به فضای بیشتری در سخت‌افزار (هزینه بیشتر).

```
//Array Multiplier

module Multiplier (
    input wire [7:0] operand_1,
    input wire [7:0] operand_2,
    output reg [15:0] product
);
    reg [15:0] partial_products[7:0];
    integer i;

    always @(*) begin
        // تولید حاصلضرب‌های جزئی
        for (i = 0; i < 8; i = i + 1) begin
            partial_products[i] = (operand_2[i]) ? (operand_1 << i) : 16'b0;
        end

        // جمع حاصلضرب‌ها
        product = partial_products[0] + partial_products[1] +
                  partial_products[2] + partial_products[3] +
                  partial_products[4] + partial_products[5] +
                  partial_products[6] + partial_products[7];
    end
endmodule
```

3.ضرب‌کننده با استفاده از شیفت رجیستر + جمع‌کننده(Shift-and-Add Multiplier)

این روش مبتنی بر جمع‌های متوالی و شیفت است.

- **روش کار:**
  - ضارب (یکی از ورودی‌ها) بیت به بیت بررسی می‌شود.
  - اگر بیت فعلی برابر ۱ باشد، مضروب (ورودی دیگر) به نتیجه فعلی اضافه می‌شود.
  - مضروب در هر مرحله یک بیت به چپ شیفت داده می‌شود.
- **ویژگی‌ها:**
  - استفاده بهینه از منابع سخت‌افزاری (کوچک‌تر).
  - سرعت کمتر به دلیل عملیات متوالی.
  - مناسب برای طراحی‌های ساده‌تر.

```
//ضربکننده با استفاده از شیفت رجیستر + جمعکننده

module Multiplier (
    input wire [7:0] operand_1,
    input wire [7:0] operand_2,
    output reg [15:0] product
);
    reg [7:0] multiplicand;
    reg [7:0] multiplier; // متغیر کمکی برای نگهداری operand_2
    reg [15:0] accumulator;
    reg [3:0] count;

    always @(*) begin
        multiplicand = operand_1;
        multiplier = operand_2; // کپی operand_2 برای انجام عملیات
        accumulator = 16'b0;
        count = 4'b1000; // 8 بیت

        while (count > 0) begin
            if (multiplier[0]) begin
                accumulator = accumulator + multiplicand;
            end
            multiplicand = multiplicand << 1;
            multiplier = multiplier >> 1; // عمل شیفت روی کپی ورودی
            count = count - 1;
        end

        product = accumulator;
    end
endmodule
```

4.ضرب‌کننده بوت(Booth Multiplier)

این روش بهینه‌سازی شده برای اعداد علامت‌دار است و تعداد عملیات را کاهش می‌دهد.

- **روش کار:**
  - اعداد دودویی به شکل خاصی کدگذاری می‌شوند) کدگذاری.(Booth
  - به جای جمع یا تفریق برای هر بیت، عملیات بر اساس گروه‌های چند بیتی انجام می‌شود.
- **ویژگی‌ها:**
  - کاهش تعداد مراحل عملیات برای اعداد با بیت‌های صفر زیاد.
  - مناسب برای اعداد علامت‌دار.
  - پیچیدگی بیشتر نسبت به روش‌های دیگر.

```
//Booth Multiplier

module Multiplier (
    input wire [7:0] operand_1,
    input wire [7:0] operand_2,
    output reg [15:0] product
);
    reg [15:0] accumulator;
    reg [8:0] multiplier; // شامل بیت اضافی برای Booth
    integer i;

    always @(*) begin
        accumulator = 16'b0;
        multiplier = {operand_2, 1'b0};

        for (i = 0; i < 8; i = i + 1) begin
            case (multiplier[1:0])
                2'b01: accumulator = accumulator + (operand_1 << i);
                2'b10: accumulator = accumulator - (operand_1 << i);
                default: ; // هیچ عملیاتی انجام نمی‌شود
            endcase
            multiplier = multiplier >> 1;
        end

        product = accumulator;
    end
endmodule
```

روش	سرعت	پیچیدگی سخت‌افزار	مزایا	معایب
ضرب‌کننده دودویی	کم	کم	ساده‌ترین روش پیاده‌سازی	سرعت پایین برای بیت‌های بیشتر
ضرب‌کننده آرایه‌ای	بالا	زیاد	سرعت بالا به دلیل موازی بودن	هزینه سخت‌افزاری بالا
شیفت + جمع‌کننده	متوسط	کم	استفاده کم از منابع سخت‌افزاری	کندتر از روش آرایه‌ای
ضرب‌کننده بوت	بالا	متوسط	بهینه‌سازی برای اعداد علامت‌دار	نیاز به مدارهای پیچیده‌تر

- اگر هدف سرعت بالا است و هزینه سخت‌افزاری اهمیت کمتری دارد، روش آرایه‌ای بهترین گزینه است.
- اگر بهینه‌سازی منابع سخت‌افزاری مدنظر است، روش شیفت + جمع‌کننده انتخاب مناسبی است.
- برای اعداد علامت‌دار یا کاهش تعداد عملیات، روش بوت مناسب است.

در این پروژه روش های مختلف را انجام داده ایم اما نتایج نهایی را برای روش ضرب کننده آرایه‌ای در این گزارش آورده‌ایم.

```
//Array Multiplier

module Multiplier (
    input wire [7:0] operand_1,
    input wire [7:0] operand_2,
    output reg [15:0] product
);
    reg [15:0] partial_products[7:0];
    integer i;

    always @(*) begin
        // تولید حاصلضرب‌های جزئی
        for (i = 0; i < 8; i = i + 1) begin
            partial_products[i] = (operand_2[i]) ? (operand_1 << i) : 16'b0;
        end

        // جمع حاصلضرب‌ها
        product = partial_products[0] + partial_products[1] +
                  partial_products[2] + partial_products[3] +
                  partial_products[4] + partial_products[5] +
                  partial_products[6] + partial_products[7];
    end
endmodule
```

کد فوق یک **ضرب‌کننده آرایه‌ای (Array Multiplier)** در زبان Verilog است. این کد دو عدد ۸ بیتی را به عنوان ورودی دریافت کرده و یک حاصل‌ضرب ۱۶ بیتی تولید می‌کند. این طراحی ساده و مبتنی بر روش محاسبه حاصل‌ضرب‌های جزئی (Partial Products) است. در ادامه، بخش‌های مختلف کد را توضیح می‌دهیم:

Operand_1: عدد اول ۸ بیتی به عنوان multiplicand

Operand_2: عدد دوم ۸ بیتی به عنوان multiplier

Product: خروجی ۱۶ بیتی که نتیجه حاصل ضرب است.

Partial_products[7:0]: آرایه‌ای که برای ذخیره حاصل‌ضرب‌های جزئی استفاده می‌شود. هر عنصر یک حاصل‌ضرب شیفت داده شده از `operand_1`است.

always @(*) begin: این بلوک **حساس به هر تغییر در ورودی‌ها** است و هر زمان که یکی از ورودی‌ها تغییر کند، محاسبات به‌روز می‌شود.

تولید حاصل‌ضرب‌های جزئی

- حلقه `for` از ۰ تا ۷ اجرا می‌شود (به ازای هر بیت از `operand_2`)
- شرط `(operand_2[i])` بررسی می‌کند که آیا بیت `i`-ام `operand_2` برابر با ۱ است یا خیر:

  اگر ۱ باشد: مقدار `operand_1` به اندازه `i` بیت به چپ شیفت داده می‌شود و به عنوان یک حاصل‌ضرب جزئی ذخیره می‌شود.

  اگر ۰ باشد: حاصل‌ضرب جزئی برابر با صفر در نظر گرفته می‌شود.

- این فرآیند **روش دستیابی به حاصل‌ضرب‌های جزئی** در ضرب دودویی را شبیه‌سازی می‌کند.

جمع حاصل‌ضرب‌های جزئی

- در این مرحله، تمام حاصل‌ضرب‌های جزئی که در آرایه `partial_products` ذخیره شده‌اند، با یکدیگر جمع می‌شوند.
- جمع این مقادیر، حاصل‌ضرب نهایی دو عدد ورودی را تولید می‌کند و در خروجی `product` ذخیره می‌شود.

مزایا و محدودیت‌ها

مزایا:

۱. **سادگی طراحی** :از روش پایه‌ای ضرب دودویی استفاده می‌کند که پیاده‌سازی آن ساده است.
۲. **ساختار واضح** :محاسبات به صورت مرحله‌ای و شفاف انجام می‌شود.
۳. **انعطاف‌پذیری** :قابل گسترش برای بیت‌های بیشتر است.

محدودیت‌ها:

۱. **کارایی پایین:**
   - جمع همه حاصل‌ضرب‌های جزئی در یک چرخه محاسبه می‌شود که ممکن است زمان‌بر باشد.
۲. **مصرف منابع بیشتر:**

- به علت ذخیره ۸ حاصل‌ضرب جزئی در یک آرایه و جمع کردن آن‌ها، مصرف منابع محاسباتی زیاد است.

۳. **سرعت پایین:**

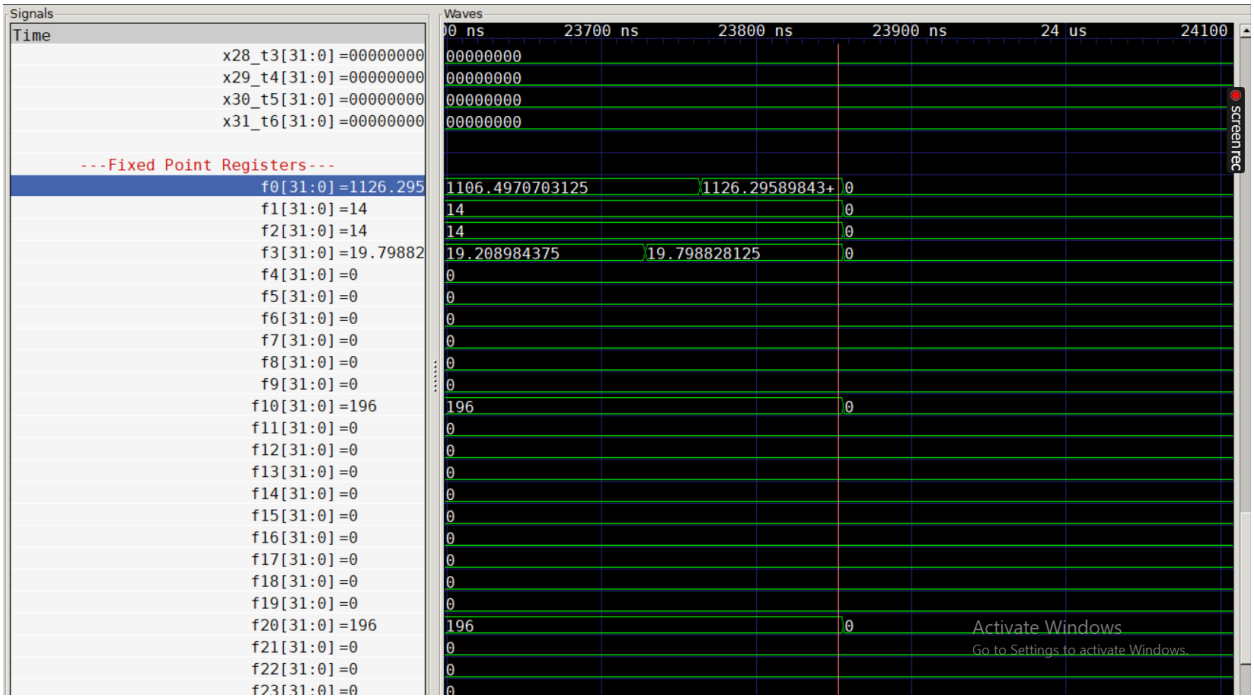
- در مقایسه با روش‌های سریع‌تر مانند ضرب‌کننده بوث یاWallace Tree ، سرعت کمتری دارد.

خروجی:

```
PS C:\Users\VICTUS\Desktop\LUMOS-main\LUMOS-main> vvp LUMOS.vvp
VCD info: dumpfile LUMOS.vcd opened for output.


Execution Finished.
```

شکل موج:



همانطور که مشاهده می‌شود مقدار نهایی f0=1126.295 شده اما به دلیل زیاد بودن رزولوشن تاخیر دیلی ها خرده داشته و دقیق نیستند.