گزارش پروژه‌ی درس معماری کامپیوتر

پیاده‌سازی جمع‌کننده و ضرب کننده‌ها

علی بهمنیار - ۹۸۲۳۰۱۸

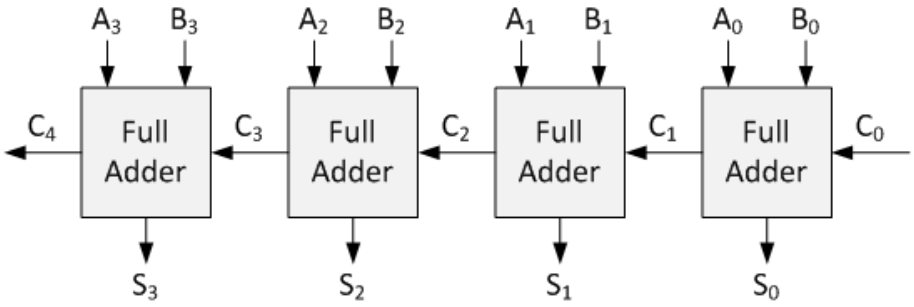
دانشکده‌ی مهندسی برق
دانشگاه صنعتی امیرکبیر
دی ماه ۱۴۰۱

فهرست مطالب

۱ جمع‌کننده‌ها

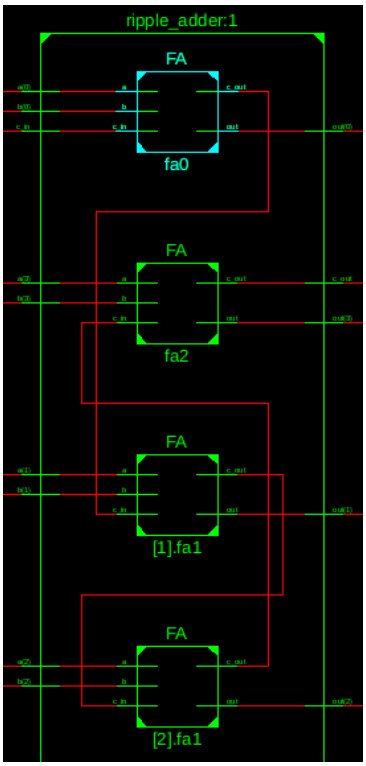
۱.۱ Ripple Adder

این جمع کننده، ساده‌ترین نوع جمع‌کننده می‌باشد و طراحی نسبتاً ساده‌ای دارد. شمای کلی این طراحی در شکل ۱ مشخص است:



شکل ۱: شماتیک کلی جمع‌کننده‌ی Ripple Adder

در این طراحی هر دو بیت توسط یک Full Adeer با یکدیگر جمع شده، سپس بیت carry حاصل به عنوان بیت carry ورودی به Full Adder بعدی داده شده تا دو بیت بعدی با یکدیگر جمع شده و این فرآیند تکرار می‌شود تا حاصل جمع نهایی ایجاد شود.
در این جمع‌کننده هر Full Adder باید منتظر جواب واحد قبلی خود بماند و بنابراین برای ایجاد پاسخ نهایی سیگنال باید به ترتیب از تمامی Full Adderها عبور کند. به این دلیل این جمع‌کننده سرعت عملرد نسبتاً پایینی دارد.
پس از پیاده‌سازی این جمع‌کننده، شماتیک RTL آن به صورت شکل ۲ می‌باشد:

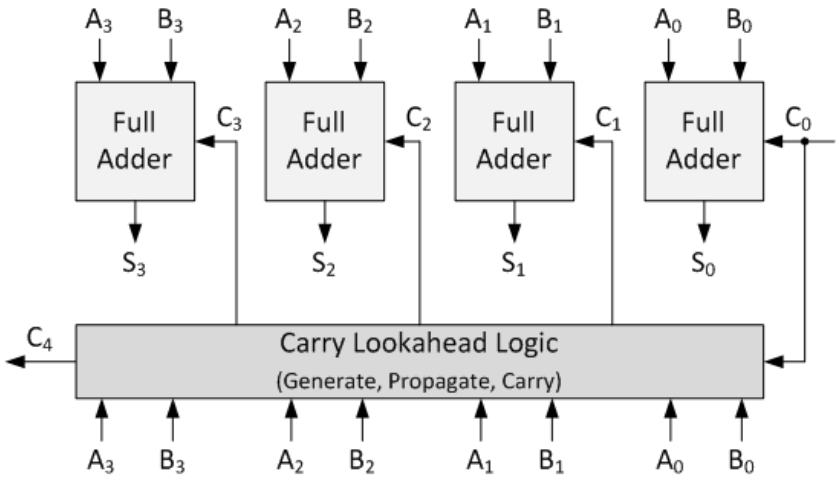


شکل ۲: شماتیک Ripple Adder

این جمع‌کننده به صورت پارامتری برای N بیت پیاده‌سازی شده و تعداد بیت‌های آن توسط پارامتر N ماژول قابل تنظیم است.

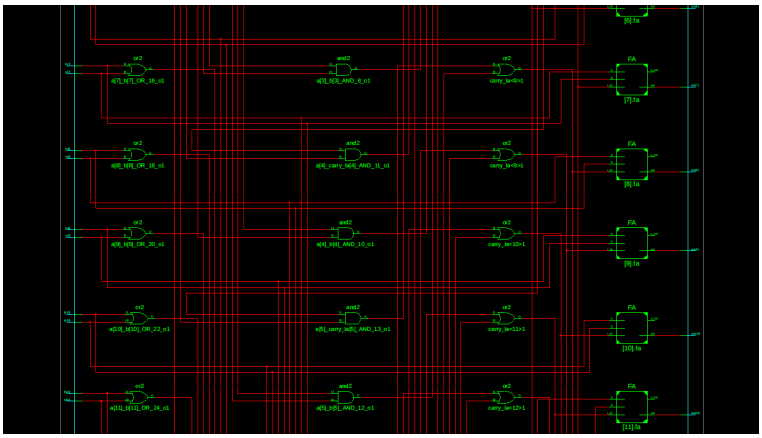
Carry-Lookahead Adder ۲.۱

این جمع‌کننده نسبت به جمع‌کننده‌ی قبلی سرعت بیش‌تری دارد، اما همچنین ساختار آن نیز پیچیده‌تر است. ساختار کلی این جمع‌کننده در شکل ۳ مشخص است:



شکل ۳: شماتیک کلی جمع‌کننده‌ی Carry-Lookahead

در جمع‌کننده‌ی Ripple Carry عامل اصلی تأخیر این است که هر واحد باید منتظر نتیجه‌ی بیت carry واحد قبلی بماند، در این پیاده‌سازی برای برطرف کردن این مشکل می‌توان بیت‌های carry را برای هر Full Adder به صورت جداگانه توسط یک بخش مجزا محاسبه کرد. این کار باعث می‌شود تا پیچیدگی مدار بیش‌تر شود ولی سرعت انجام جمع را به طور قابل ملاحظه‌ای افزایش می‌دهد. پس از پیاده‌سازی این جمع‌کننده، بخشی از شماتیک RTL آن به صورت شکل ۴ می‌باشد، از این شماتیک نیز پیچیدگی بیش‌تر مدار نسبت به راه‌حل قبلی مشخص است:



شکل ۴: شماتیک Carry-Lookahead Adder

این جمع‌کننده به صورت پارامتری برای N بیت پیاده‌سازی شده و تعداد بیت‌های آن توسط پارامتر N ماژول قابل تنظیم است.

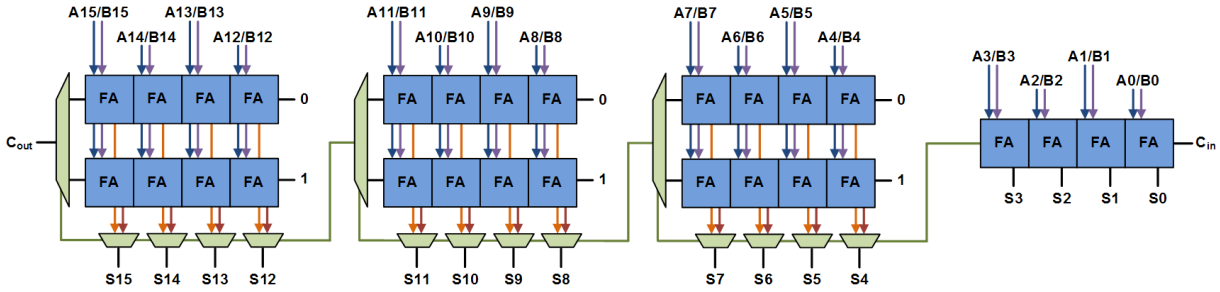
```
assign carry_la[j] = (a[j-1] & b[j-1]) | ((a[j-1] | b[j-1]) & carry_la[j-1]);
```

کد ۱: پیاده‌سازی Carry-Lookahead

پیاده‌سازی Carry-Lookahead در کد ۱ مشخص است؛ در دو صورت بیت carry می‌بایست ۱ باشد: یا هر دو بیت ورودی ۱ باشند، یا یکی از بیت‌های ورودی به همراه بیت carry قبلی ۱ باشند.

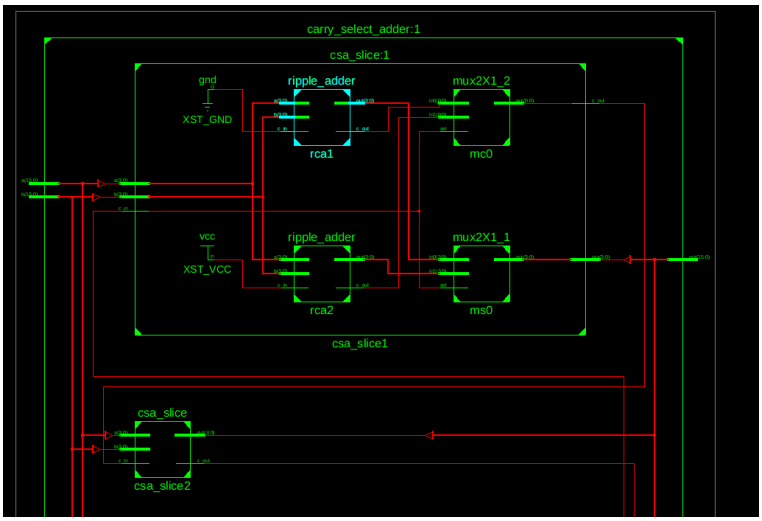
Carry Select Adder ۳.۱

در راهکار قبلی سرعت محاسبه بهبود یافت اما همچنان برای محاسبه‌ی هر یک از بیت‌های carry می‌بایست منتظر خروجی بیت carry قبلی در مدار carry-lookahead بمانیم، به عبارت دیگر انتشار سیگنال در مدار carry-lookahead همچنان زمان‌بر است.
راهکار دیگر این است که کل عملیات جمع را به دسته‌های n بیتی تقسیم کنیم و عملیات جمع را ابتدا به ازای هر دو بیت کری 0 و 1 انجام دهیم، سپس بعد از مشخص شدن بیت کری صحیح با استفاده از یک مالتی‌پلکسر آن را انتخاب کنیم. شمای کلی این راهکار در شکل ۵ مشخص است.



شکل ۵: شماتیک کلی Carry Select Adder

هر یک از بلوک‌های ۴ تایی در واقع یک ripple adder هستند. بهینه‌ترین سایز زیر بلوک برای یک جمع n بیتی نیز برابر با $\lfloor\sqrt{n}\rfloor$ می‌باشد.
پس از پیاده‌سازی این جمع‌کننده، بخشی از شماتیک RTL آن به صورت شکل ۶ می‌باشد، این جمع‌کننده بیش‌ترین مساحت را اشغال خواهد کرد اما سریع‌ترین جمع‌کننده‌ها نیز خواهد بود.



شکل ۶: شماتیک Carry-select-adder

این جمع‌کننده به صورت ۱۶ بیتی پیاده‌سازی شده است.

۲ ضرب‌کننده‌ها

۱.۲ Shift-Add Multiplier

این ضرب‌کننده ساده‌ترین نوع ضرب‌کننده می‌باشد، این ضرب‌کننده با استفاده از کلاک در چندین سیکل هربار عملیات ضرب را برای یک بیت انجام داده و سپس حاصل را با خروجی نهایی جمع می‌کند. منطق اصلی این ضرب‌کننده در کد ۲ مشخص است:

```
always @(posedge clk) begin
        if (bn < N) begin
            finished = 0;
            cb = b[bn];

            if (bn == 0) begin
                case (cb)
                    0: res[2*N-1:N] = (a & zero);
                    1: res[2*N-1:N] = (a & one);
                endcase
            end else begin
                case (cb)
                    0: res[2*N:N] = res[2*N-1:N] + (a & zero);
                    1: res[2*N:N] = res[2*N-1:N] + (a & one);
                endcase

            end
            res = res >> 1;
            bn = bn + 1;
        end else begin
            finished = 1;
        end

        out = res;
    end
```

کد ۲: پیاده‌سازی Shift-Add Multiplier

این ضرب‌کننده به صورت پارامتری برای N بیت پیاده‌سازی شده و تعداد بیت‌های آن توسط پارامتر N ماژول قابل تنظیم است.

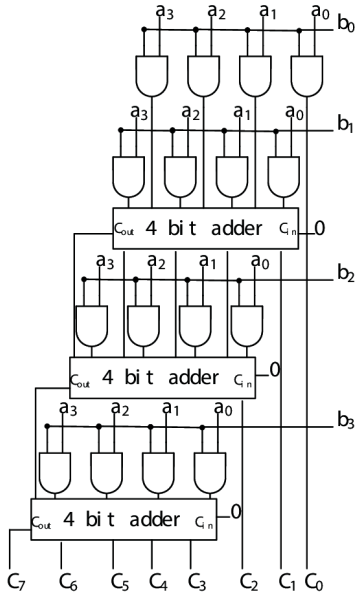
۲.۲ Array Multiplier

جمع‌کننده‌ی shift-Add در چندین کلاک کار می‌کند، اما می‌توانیم بدون استفاده از کلاک و تنها با استفاده از یک مدار ترکیبی نیز عملیات جمع را پیاده‌سازی کنیم، ابتدا می‌توانیم عملیات ضرب را به صورت گسترده مطابق شکل ۷ بنویسیم:

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & p_{30} & p_{20} & p_{10} & p_{00} \\
 & & & p_{31} & p_{21} & p_{11} & p_{01} & \times \\
 & & p_{32} & p_{22} & p_{12} & p_{02} & \times & \times \\
 & p_{33} & p_{23} & p_{13} & p_{03} & \times & \times & \times \\
\hline
s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

شکل ۷: نمای گسترده‌ی ضرب دو عدد چهار بیتی

حال می‌توان تمام عملیات نشان داده‌شده در شکل بالا را توسط یک مدار ترکیبی پیاده‌سازی کرد، شماتیک کلی این مدار به این صورت خواهد بود:



شکل ۸: شماتیک ضرب‌کننده‌ی آرایه‌ای