

گزارش پروژه‌ی درس معماری کامپیوتر

پیاده‌سازی جمع‌کننده و ضرب‌کننده‌ها

علی بهمنیار - ۹۸۲۳۰۱۸



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)



دانشکده‌ی مهندسی برق

دانشگاه صنعتی امیرکبیر

دی ماه ۱۴۰۱

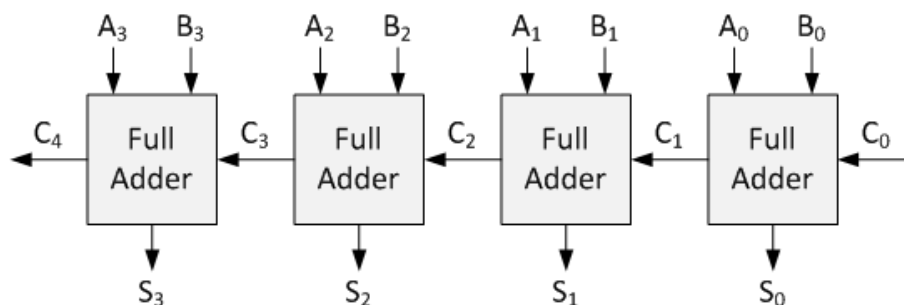
فهرست مطالب

۲	۱ جمع‌کننده‌ها
۲ Ripple Adder ۱.۱
۳ Carry-Lookahead Adder ۲.۱
۴ Carry Select Adder ۳.۱
۵	۲ ضرب‌کننده‌ها
۵ Shift-Add Multiplier ۱.۲
۵ Array Multiplier ۲.۲
۶ Carry-Save Multiplier ۳.۲
۸	پیوست‌ها
۸	آ نتایج شبیه‌سازی‌ها

۱ جمع‌کننده‌ها

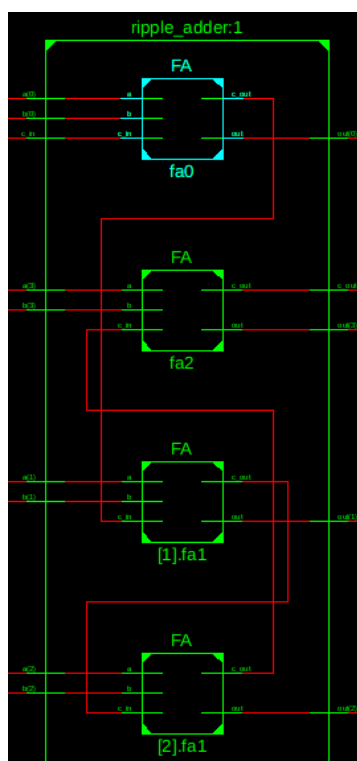
۱.۱ Ripple Adder

این جمع‌کننده، 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 Adder 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 Adder ها عبور کند. به این دلیل این جمع‌کننده سرعت عملکرد نسبتاً پایینی دارد. پس از پیاده‌سازی این جمع‌کننده، شماتیک RTL آن به صورت شکل ۲ می‌باشد:

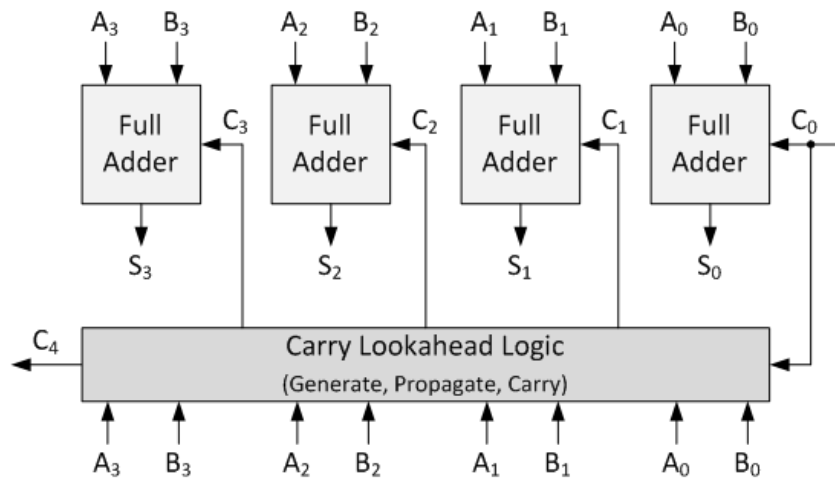


شکل ۲: شماتیک Ripple Adder

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

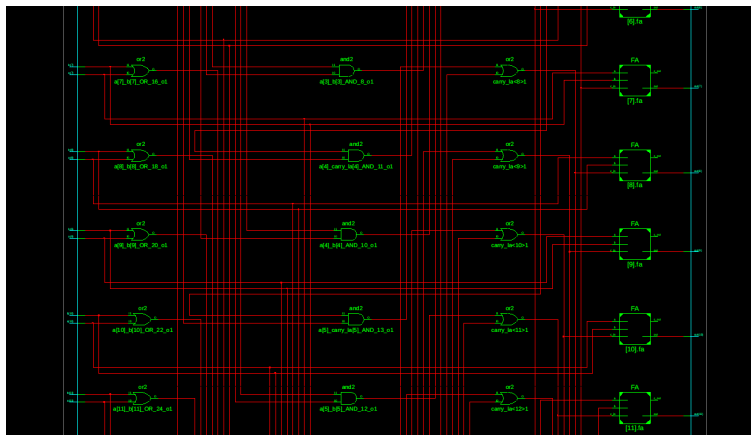
۲.۱ Carry-Lookahead Adder

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.

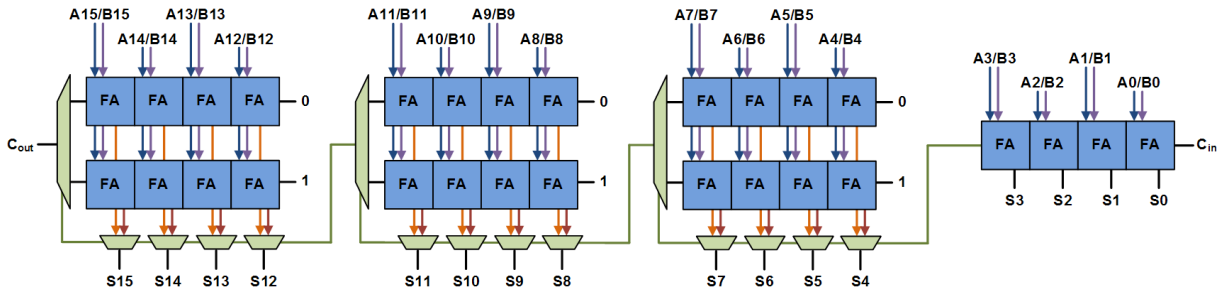
```
1 assign carry_la[j] = (a[j-1] & b[j-1]) | ((a[j-1] | b[j-1]) & carry_la[j-1]);
```

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.

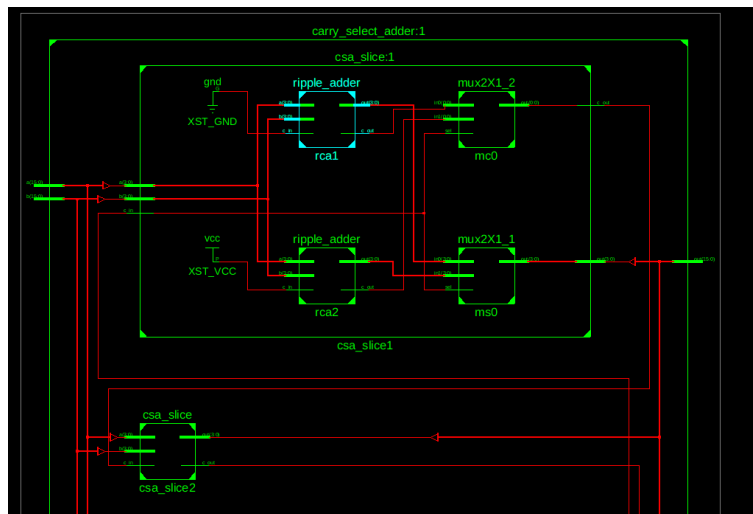
۳.۱ Carry Select Adder

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 کرای 0 و 1 انجام دهیم، سپس بعد از مشخص شدن بیت کرای 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 هر بار 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:

```

1 always @(posedge clk) begin
2     if (bn < N) begin
3         finished = 0;
4         cb = b[bn];
5
6         if (bn == 0) begin
7             case (cb)
8                 0: res[2*N-1:N] = (a & zero);
9                 1: res[2*N-1:N] = (a & one);
10            endcase
11        end else begin
12            case (cb)
13                0: res[2*N:N] = res[2*N-1:N] + (a & zero);
14                1: res[2*N:N] = res[2*N-1:N] + (a & one);
15            endcase
16
17        end
18        res = res >> 1;
19        bn = bn + 1;
20    end else begin
21        finished = 1;
22    end
23
24    out = res;
25 end

```

کد ۲: پیاده‌سازی Shift-Add Multiplier

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

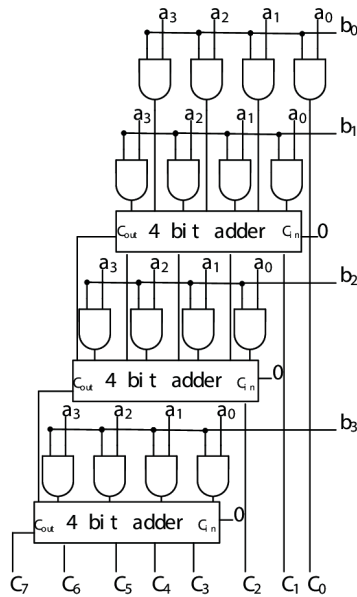
۲.۲ Array Multiplier

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

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

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 صورت شکل ۸ خواهد بود:



شکل ۸: شماتیک ضرب‌کننده‌ی آرایه‌ای

این ضرب‌کننده به صورت ۴ بیتی پیاده‌سازی شده است.

۳.۲ Carry-Save Multiplier

در ضرب‌کننده‌ی آرایه‌ای می‌بایست همواره منتظر انتشار نتیجه‌ی بیت‌های carry بمانیم اما می‌توان عملیات جمع را به صورت دیگری نیز انجام داد، مثال زیر را در نظر می‌گیریم:

$$\begin{array}{r} 111 \\ \times 11 \\ \hline 10101 \end{array}$$

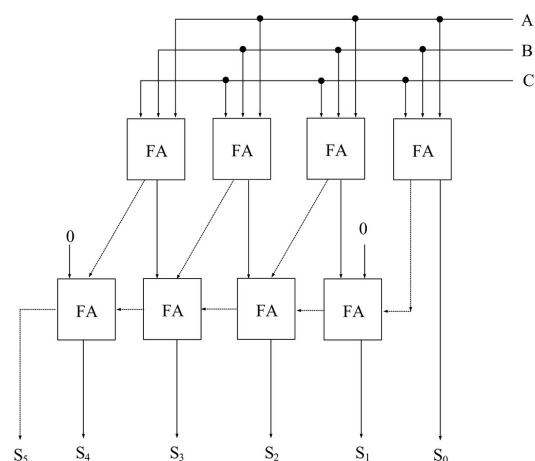
راه دیگر برای انجام ضرب فوق به صورت زیر است:

$$\begin{array}{r} 111 \\ \times 11 \\ \hline 1221 \end{array}$$

این نمایش نامتداول اما قابل فهم است؛ در واقع این‌جا ۲ نشان دهنده‌ی ۱۰ است یعنی یک بیت ۰ با بیت کری ۱. بعد از به دست آمدن این نتیجه حال می‌توانیم بیت‌های کری به دست آمده را در مرحله‌ی بعدی جمع کرده و نتیجه‌ی نهایی را به صورت باینری متداول به دست آوریم:

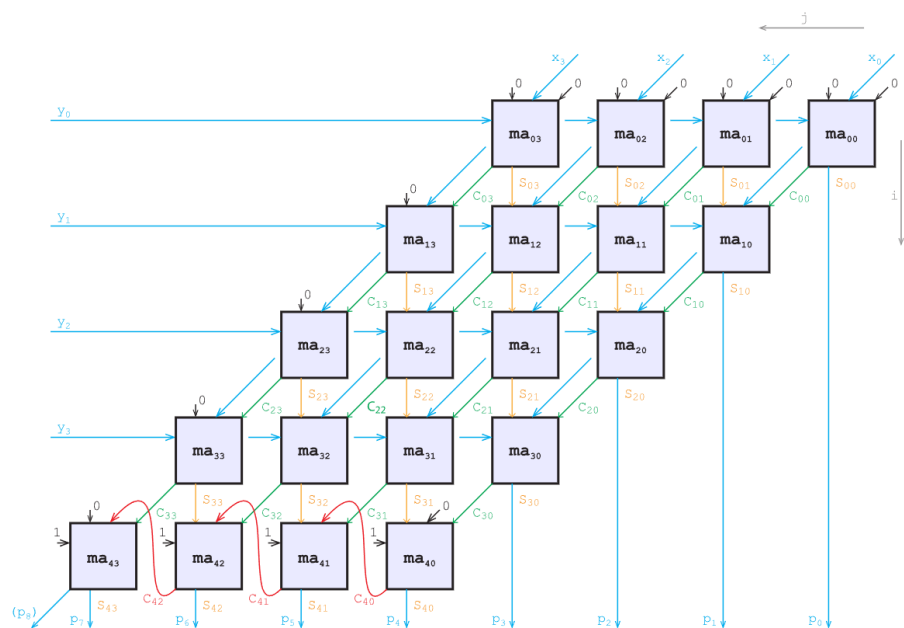
$$\begin{array}{r} 1001 \\ + 11 \\ \hline 10101 \end{array}$$

به عبارت دیگر در جمع carry-save ابتدا جمع را حساب کرده سپس نتیجه بیت‌های carry را با نتیجه‌ی به دست آمده جمع می‌کنیم. شمای یک جمع‌کننده‌ی carry-save در شکل ۹ مشخص است:



شکل ۹: شماتیک carry-save adder

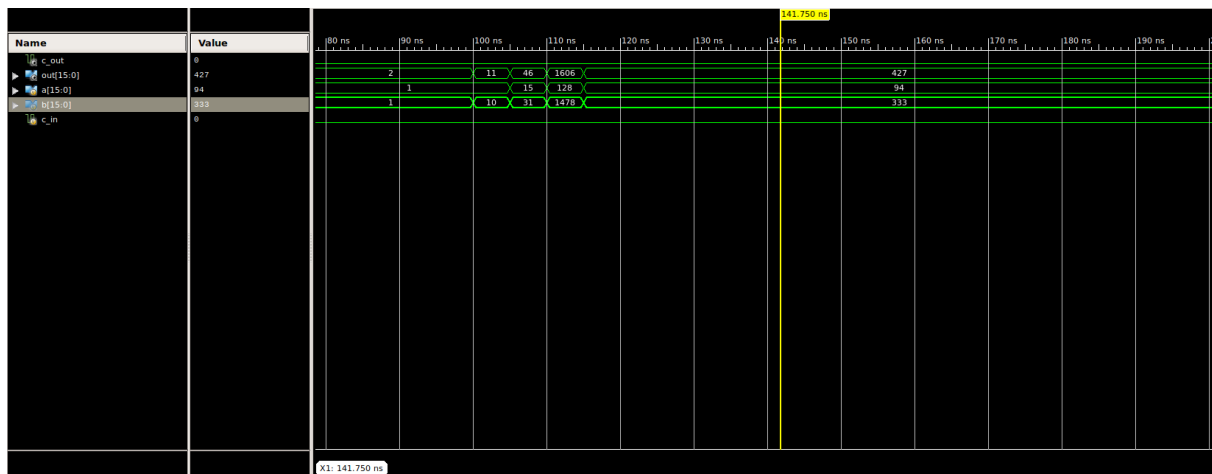
حال می‌توانیم با استفاده از این جمع کننده یک سلول ضرب کننده طراحی کنیم که چهار بیت ورودی می‌گیرد، دو تای آن‌ها را در یکدیگر ضرب کرده نتیجه را با دو بیت دیگر جمع کرده و دو بیت خروجی می‌دهد:



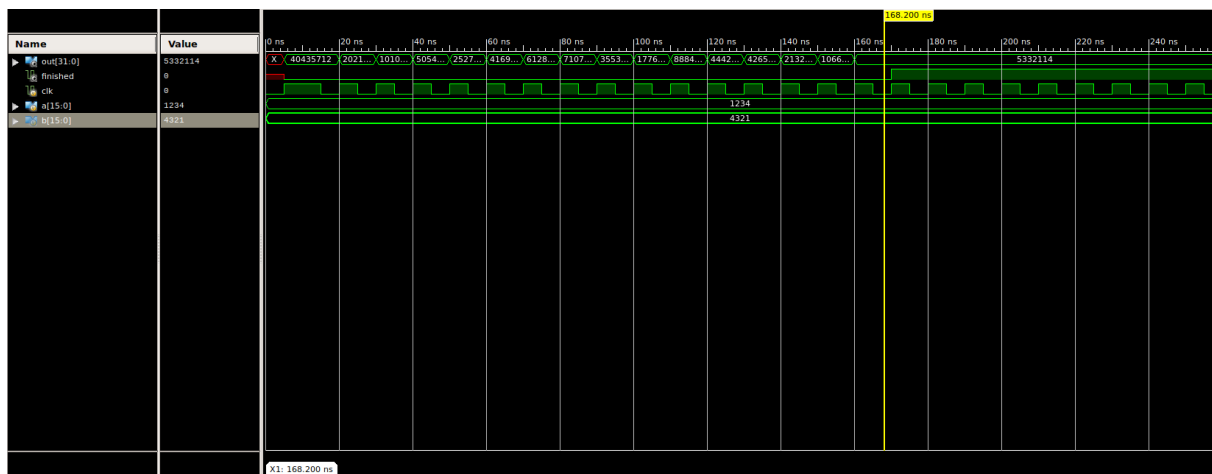
شکل ۱۰: شماتیک carry-save multiplier

این ضرب کننده به صورت ۴ بیتی پیاده‌سازی شده است.

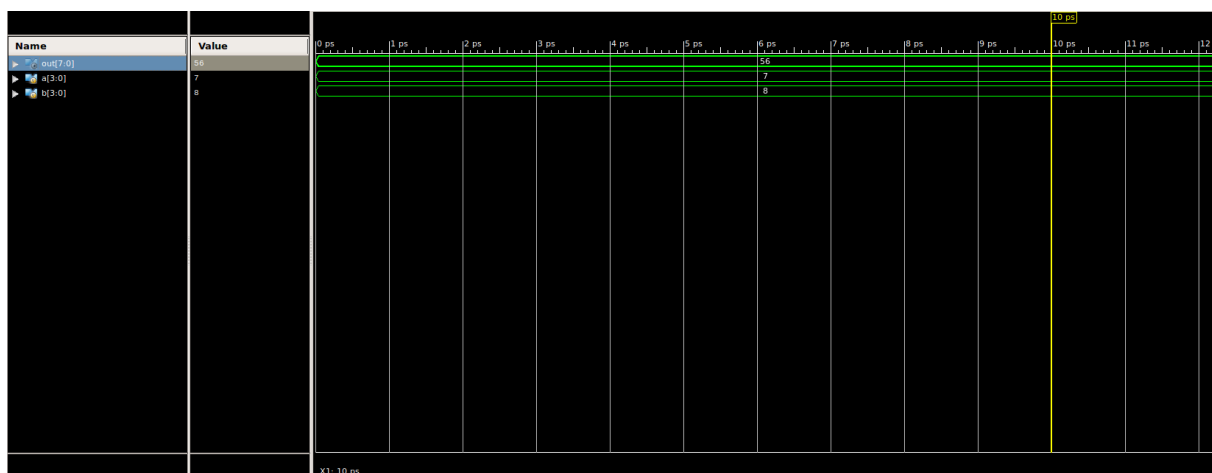
آ نتایج شبیه‌سازی‌ها



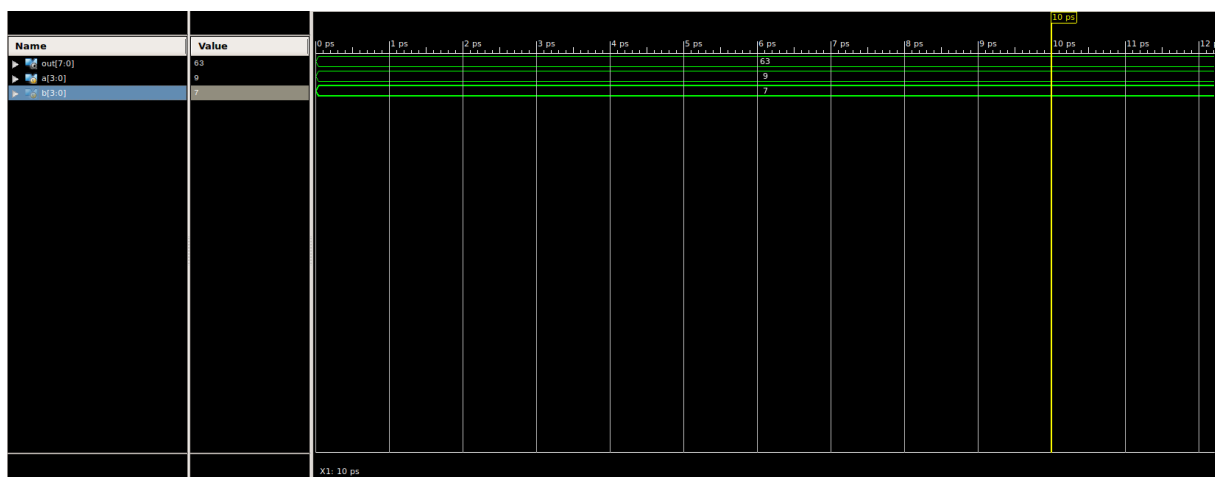
شکل ۱۱: شبیه‌سازی جمع‌کننده‌ها



شکل ۱۲: شبیه‌سازی Shift-Add Multiplier



شکل ۱۳: شبیه‌سازی Array Multiplier



شکل ۱۴: شبیه‌سازی Carry-Save Multiplier