ارائه معماری نوین ضرب‌کننده با استفاده از مدل سیستم Multi-expert برای کاربردهای سریع

علی ذاکرالحسینی[*]، کیوان ناوی [*]، امید کاوه‌ای [†]

چکیده

در این مقاله معماری نوینی برای ضرب‌کننده‌ها ارائه شده است. این معماری بر پایه سیستم Multi-Expert و بر اساس مدل موازی پیاده‌سازی شده است. در این مدل Expert های مختلف بصورت موازی فعالیت می‌نمایند. این Expertها پیاده‌سازی الگوریتم‌های مختلف کدگذار Booth می‌باشند. بر این اساس نتایج حاصله از این Expertها (حاصل‌ضرب‌های جزئی) پس از عبور از شبکه جمع این ضرب‌کننده وارد یک ماژول تصمیم‌گیر می‌شوند. وظیفه این بخش از سیستم ضرب‌کننده انتخاب بهینه نتایج در راستای رسیدن به بالاترین سرعت ممکنه می‌باشد. هدف از ارائه این مدل برای ضرب‌کننده‌ها دستیابی به سرعت بیشتر در مقایسه با سایر طرح‌های امروزی است. معماری ارائه شده برای، بر اساس نتایج سنتز و شبیه‌سازی موفق شده است به بهبودی درحدود ۱۴٪ تا ۲۱،۵٪ در ۱۰ بیت اول و ۴٪ تا ۶٪ در ۵۴ بیت بعدی حاصل جمع حاصلضرب‌های جزئی دست پیدا کند.

کلمات کلیدی

ضرب‌کننده، الگوریتم Booth، Multi-Expert System.

A Novel Multiplier Architecture By Using Multi-Expert System Model for High-Speed Applications

Ali Zakeralhosseini, Keivan Navi, Omid kavehie
Department of Electrical and Computer Engineering
Shahid Beheshti University, Tehran, IRAN

Abstract

In this paper a novel architecture for multipliers is presented. This architecture is implemented based on a parallel multi-expert system. That is, the experts are placed in parallel. Each expert has been implemented to execute a particular booth algorithm. The outputs of the expert (particular product) are passed to a summation and finally a decision making models. The decision making model select the optimum result based on speed constraint. The novel multiplier architecture, implements fast multiple compare to known approach. The simulation and synthesis result, represents an increase in performance of 14% to 21.5% for the first 10 bits and 4% to 6% for the next 54 bits of partial product summation.

Keywords

Multiplier, Booth algorithm, Multi-Expert System.

[*] استادیار دانشکده مهندسی برق و کامپیوتر دانشگاه شهیدبهشتی، {zaker,navi}@sbu.ac.ir.
[†] دانشجوی دکتری معماری کامپیوتر دانشکده مهندسی برق و کامپیوتر دانشگاه شهیدبهشتی، kavehie@sbu.ac.ir.

۱- مقدمه

ضرب یکی از اصلی‌ترین اعمال حسابی می‌باشد. بنا بر نتایج بدست آمده در [۲]، [۳]، [۱۱]، [۱۵]، [۱۸]، ۸،۷۲٪ از کل دستورالعمل‌های یک برنامه با عمل ضرب درگیر می‌باشند. با توجه به این نکته که، عمل ضرب، یکی از زمان‌گیرترین اعمال حسابی محسوب می‌شود، در نتیجه، ساخت ضرب‌کننده‌های سریع، برای بهبود کارآیی پروسسورها موضوعی کاملا جدی محسوب می‌شود. پیشرفت‌های اخیر در زمینه تکنولوژی، امکان پیاده‌سازی‌های، سریع‌تر و کوچک‌تر را فراهم نموده‌اند. با توجه به این مطلب که، می‌توان عمل ضرب را، به فرم‌های مختلف، سخت‌افزاری، و یا نرم‌افزاری، پیاده‌سازی نمود، امروزه پیاده‌سازی سخت‌افزاری، به علت سرعت بیشتر و نیز کاهش یافتن قیمت سخت‌افزار، بیشتر مورد توجه می‌باشد.

در بخش دوم این مقاله مروری بر روش‌های مختلف کدگذاری خواهیم داشت. در بخش سوم به معرفی مدل سیستمی Multi-Expert [۱] و معماری نوین ضرب‌کننده خواهیم پرداخت. بخش بعدی در این مقاله حاوی نتایج حاصله از شبیه‌سازی و سنتز می‌باشد. در انتها نیز نتیجه‌گیری نهایی ارائه خواهد شد.

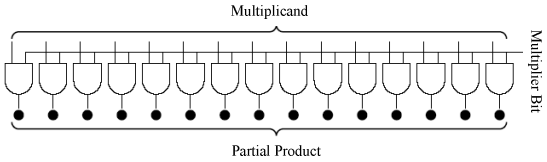
۲- روش‌های کدگذاری

در این بخش چندین روش متفاوت، به‌منظور تولید حاصل‌ضرب‌های‌جزئی، مورد بررسی قرار می‌گیرد. یکی از بهترین گونه‌های الگوریتم ضرب، الگوریتم کدگذاری Booth است [۱۰] که در سال ۱۹۵۱ توسط Booth ارائه شده است. این الگوریتم اجازه کاهش تعداد حاصل‌ضرب‌های‌جزئی را داده و در نتیجه باعث افزایش سرعت فرآیند ضرب می‌شود. این روش بیت‌های مضروب‌فیه را در گروه‌های دوتایی دوباره‌کدگذاری* می‌نماید. به‌علاوه، کاهش بیشتر در تعداد حاصل‌ضرب‌های‌جزئی با بسط ساده الگوریتم Booth با کدگذاری دوباره تعداد بیشتری از ورودی‌ها، امکان پذیر می‌باشد، اما انجام این کار تعدادی عمل جمع زمان‌بر را در پی خواهد داشت [۶]-[۱۱].

۲-۱- Non-Booth

ساده‌ترین روش برای کدگذاری non-Booth نامیده می‌شود. یک ردیف از سخت‌افزار این کدگذار در شکل ۱ برای یک ضرب‌کننده ۱۶×۱۶ بیتی نشان داده شده است. دلیل شیفت حاصل‌ضرب‌های‌جزئی وجود وزن‌های محاسباتی متفاوت حاصل‌ضرب‌های‌جزئی می‌باشد. در واقع، این الگوریتم به صورت ساده، نمایشی از الگوریتم شیفت و جمع می‌باشد. تعداد حاصل‌ضرب‌های‌جزئی تولید شده توسط این الگوریتم برابر است با اندازه مضروب‌فیه. این الگوریتم حاصل‌ضرب‌های‌جزئی را از مجموعه {0,M} انتخاب می‌کند (M همان مضروب است).

*- recode

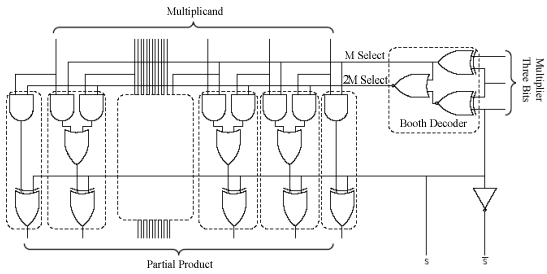


شکل ۱ یک ردیف از کدگذار non-Booth

بطور کلی در الگوریتم‌های خانواده Booth و الگوریتم‌های مشابه، دو مرحله برای کدگذاری انجام می‌شود: کدگذاری و انتخاب†. هدف از کدگذاری بررسی کردن گروه‌های چند بیتی از ضرب‌شونده (مضروب) و تعیین عملیاتی است که باید بر روی آن انجام شود (جدول ۱) [۹]-[۱۵].

۲-۲- Booth2

الگوریتم Booth تعداد حاصل‌ضرب‌های‌جزئی را با کدگذاری‌دوباره مضروب‌فیه و با استفاده از خاصیت رشته‌ای‡ کاهش می‌دهد؛ این خاصیت با بیان، رشته‌های صفر و یک متوالی، تعداد کمتری حاصل‌ضرب‌جزئی را تولید می‌نماید. الگوریتم Booth، کدگذاری را به صورت سریال انجام می‌دهد [۱۰]، از این‌رو "الگوریتم Booth بهبودیافته§" [۲۱] که کدگذاری را به صورت موازی و هم‌زمان انجام می‌دهد، توسط MacSorley ارائه گردید. در الگوریتم Booth بهبودیافته، که از این به بعد برای سادگی، با نام الگوریتم Booth2 نامیده می‌شود، مضروب‌فیه به گروه‌های سه بیتی هم‌پوش** تقسیم می‌شود. در شکل ۲ سلول پایه این کدگذار نشان داده شده است. این روش تعداد حاصل‌ضرب‌های جزئی را بدون نیاز به هیچ پیش‌جمعی†† با نرخ ۲ کاهش می‌دهد. مثلا بر طبق محاسبات در این روش (برای یک ضرب‌کننده ۱۶×۱۶ بیتی)، تعداد حاصل‌ضرب‌های‌جزئی از ۱۶ به ۹ کاهش یافته است.



شکل ۲ سلول پایه کدگذار Booth2

† - selection
‡- string property
§- modified Booth algorithm
**- overlapping groups
††- preadd

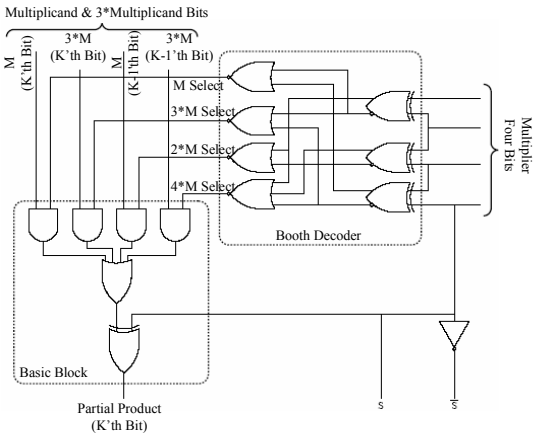
تعداد کل نقاط نیز از ۲۵۶ به ۱۷۷ رسیده است. هر حاصل‌ضرب‌جزئی در مقایسه با دیگری دو بیت شیفت دارد. مضارب مضروب از مجموعه {±2M,±M,0} انتخاب می‌شود (جدول ۱). در شکل ۲، S برابر صفر است، اگر حاصل‌ضرب‌جزئی مثبت باشد (۴ سطر بالایی جدول ۲)؛ و S برابر یک خواهد بود، اگر حاصل‌ضرب‌جزئی منفی باشد (۴ سطر پایینی جدول ۲) [۲]، [۳]، [۱۵]، [۱۸]، [۱۹].

جدول ۱ انتخاب حاصل‌ضرب‌های‌جزئی در روش Booth2

بیت‌های ضرب‌کننده	انتخاب
000	+0
001	+M
010	+M
011	+2M
100	-2M
101	-M
110	-M
111	-0

۲-۳- Booth3

خاصیت رشته‌ای که در الگوریتم Booth مورد استفاده قرار گرفت، فقط به گروه‌های سه‌تایی محدود نمی‌شود. در Booth3 مضروب‌فیه به گروه‌های چهاربیتی هم‌پوش تقسیم خواهد شد که هر گروه، به صورت هم‌زمان برای انتخاب یک حاصل‌ضرب‌جزئی مشخص، کدگشایی خواهد شد. هر حاصل‌ضرب‌جزئی چنان‌چه در شکل ۳ نشان داده شده است، متعلق به مجموعه {±4M,±3M,±2M,±M,0} خواهد بود (جدول ۲). با استفاده از این روش تعداد حاصل‌ضرب‌های‌جزئی از ۱۶ به ۶ و تعداد نقاط از ۲۵۶ به ۱۲۶ کاهش یافته است [۹]-[۲۱]. جدول مربوطه در جدول ۲، آورده شده است. مشکل بزرگ این الگوریتم، پیچیدگی منطق انتخاب حاصل‌ضرب‌جزئی و کدگشای Booth می‌باشد (شکل ۳).



شکل ۳ سلول پایه کدگذار Booth3

در شکل ۳، S برابر صفر است، اگر حاصل‌ضرب‌جزئی مثبت باشد (۸ سطر سمت چپ جدول ۳)؛ و S برابر یک خواهد بود، اگر حاصل‌ضرب‌جزئی منفی باشد (۸ سطر سمت راست جدول ۳) [۲]، [۳]، [۱۵]، [۱۸]، [۱۹]. تعداد حاصل‌ضرب‌های‌جزئی را می‌توان، با استفاده از مقادیر بیشتر شیفت، کاهش داد، بطور مثال در الگوریتم Booth4 مضارب:

{±8M,±7M,±6M,±5M,±4M,±3M,±2M,±M,0}

را تولید می‌کند. مضارب‌سخت 3M (6M با شیفت 3M بدست می‌آید) 5M و 7M هر یک، به یک جمع‌کننده مجزا نیاز دارند. علاوه بر افزایش نمایی تعداد جمع‌کننده‌های مورد نیاز، کدگذارهای* Booth4 پیچیده‌تر می‌باشند (جدول ۴). از این رو Booth4 و مرتبه‌های بالاتر آن غیر عملی‌اند. پژوهش‌های گوناگون و متنوعی در [۶]، [۸]، [۹]، [۱۴]، [۱۷]، [۲۰]، [۲۱]، [۲۲] در این حیطه انجام، شده است. در این میان تمرکز [۸]، [۹]، بیشتر بر الگوریتم‌های Booth افزونه می‌باشد. در [۱۹]، بررسی میزان اثرگذاری الگوریتم Booth در عملکرد ضرب‌کننده‌های موازی که از ساختار کمپرسور، برای کاهش حاصل‌ضرب‌های‌جزئی استفاده می‌کنند، صورت گرفته است. به روش مشابه می‌توان مراتب بالاتری از الگوریتم Booth را تعریف نمود. راه حل بهتری که می‌توان بکار برد، استفاده از چندین الگوریتم بطور همزمان است. این راه‌حل که بحث اصلی این مقاله می‌باشد در قسمت بعدی شرح داده شده است.

جدول ۲ انتخاب حاصل‌ضرب‌های‌جزئی در روش Booth3

انتخاب	بیت‌های ضرب‌کننده	انتخاب	بیت‌های ضرب‌کننده
-4M	1000	+0	0000
-3M	1001	+M	0001
-3M	1010	+M	0010
-2M	1011	+2M	0011
-2M	1100	+2M	0100
-M	1101	+3M	0101
-M	1110	+3M	0110
-0	1111	+4M	0111

۳- ارائه معماری نوین ضرب‌کننده

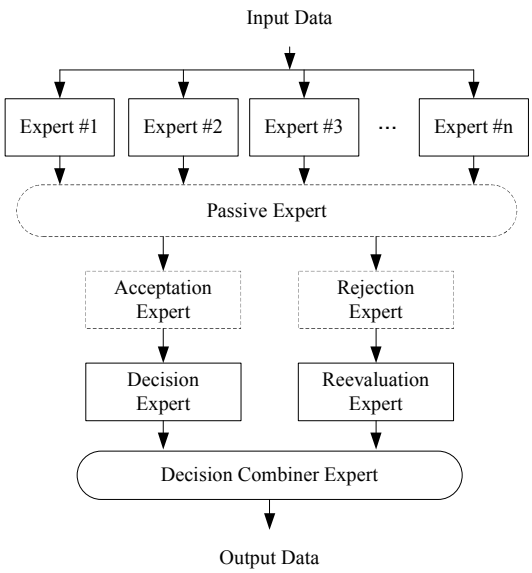
۱-۳- معرفی سیستم Multi-Expert

همانطور که در قسمت قبلی نیز اشاره شد راه حل بسیار هوشمندانه‌تر استفاده از رهیافت Multi-Expert Systemها می‌باشد [۳]. این سیستم‌ها مزیت‌های بسیاری را نسبت به روش‌های معمول دارند. مزیت ابتدایی آنها سرعت پاسخ است. در روش معمول شما باید پارامترهای زیادی را استخراج کرده و بر اساس آنها تصمیم‌گیری نمائید. اما این روش جنبه‌های مختلف مسئله توسط expertهای مختلف (که می‌توانند بسیار ساده نیز باشند) سنجیده شده و جواب‌های میانی بدست می‌آیند و سپس یک سیستم تصمیم گیرنده

*- encoder

(Decision Making) جواب نهایی را استخراج می‌کند [۴]، [۵]. مزیت دوم، تنوع بسیار زیاد expertها است. بعنوان مثال برای یک ضرب‌کننده هم می‌توان از مدل‌های مختلف رفتاری استفاده نمود و هم مدل‌های گوناگون ساختاری را در نظر گرفت و همه را در تصمیم‌گیری نهایی دخالت داد. سیستمهای Multi-Expert از لحاظ پیاده‌سازی سخت‌افزاری نیز بر الگوریتم‌های پیچیده برتری دارند زیرا با توجه به زیاد بودن الگوریتم‌های حل مسئله ، آزادی بیشتری در انتخاب expertهایی که از لحاظ پیاده‌سازی مناسب‌تر هستند وجود دارد. فلسفه حل یک مسئله بصورت Multi-Expert در واقع استفاده از الگوریتم‌های مختلف و متنوع و مستقل از هم و ترکیب نتایج آنها برای بدست آوردن جواب بهینه نهایی می‌باشد. اگر هرکدام از الگوریتم‌ها را یک expert بنامیم سه روش برای ترکیب آنها وجود دارد: روش سری، موازی و ترکیبی [۱].

در روش سری، اطلاعات ورودی توسط یک expert تحت محدودیت‌هایی پردازش شده و غربال می‌شوند. اما در روش موازی همه expertها همزمان و مستقل از هم عمل می‌نمایند و معمولا دو نوع expert در نظر گرفته می‌شود، فعال و غیرفعال. در نوع فعال هر expert کاملا مستقل عمل نموده و روی جواب نهایی اثر می‌گذارد؛ ولی در نوع غیرفعال ورودی از لایه قبلی گرفته می‌شود. در این روش ساده یا پیچیده بودن expertها چندان مطرح نبوده ولی اگر پیاده‌سازی سخت‌افزاری و سرعت مهم باشد باید تا حد امکان زمان اجرای expertها برابر باشد زیرا تاخیر سیستم برابر با تاخیر کندترین الگوریتم (expert) خواهد بود [۱]، [۴]، [۵]. شکل ۴ روش موازی را نشان میدهد.



شکل ۴ مدل موازی سیستم Multi-Expert

expertهای Acceptation و Rejection اطلاعات خروجی هر expert را بازرسی کرده و آنرا تائید یا رد می‌کنند. expert Reevaluation میتواند طوری طراحی شود که در صورت لزوم expertها را تصحیح نموده و یا expertی را که عملکرد خوبی ندارد از لیست حذف نماید. در نهایت expert ترکیب (Decision Combiner) خروجی‌های expertهای مختلف را ترکیب کرده و خروجی بهینه را تولید می‌نماید. وظیفه این بخش از سیستم ضرب‌کننده انتخاب بهینه نتایج در راستای رسیدن به بالاترین سرعت ممکنه می‌باشد.

در روش ترکیبی نیز از ترکیب روش‌های سری و موازی استفاده می‌شود. استفاده از مدل ترکیبی بعلت افزایش چشمگیر پیچیدگی زیاد مورد توجه نمی باشد [۵].

۳-۲- ارائه معماری ضرب‌کننده

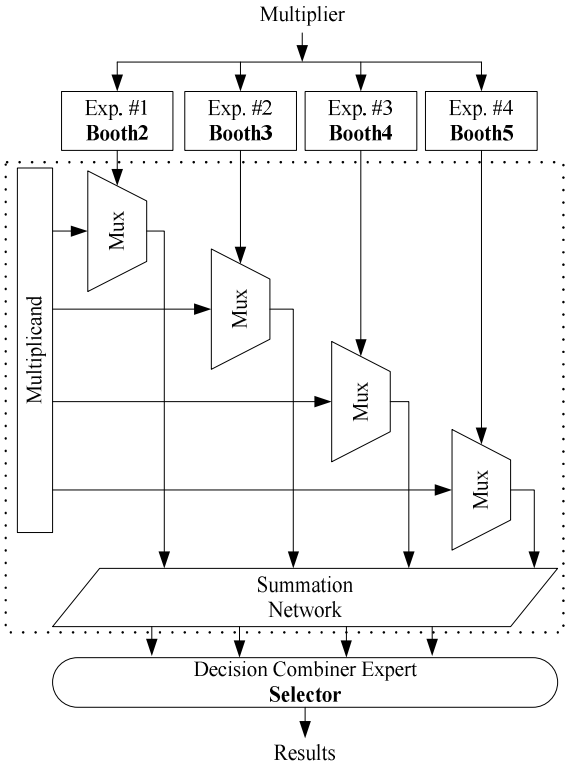
در این معماری از مدل موازی سیستم Multi-Expert استفاده شده است. نوع پیاده‌سازی نیز بر اساس مدل فعال می‌باشد. طبق این پیاده‌سازی (شکل ۵) expertهای موازی همان پیاده‌سازی الگوریتم‌های مختلف Booth می‌باشند. نقطه‌چین نشان داده شده در شکل ۵ بیانگر این است که این قسمت از پیاده‌سازی الگوریتم‌های Booth جدا نبوده و جزء همان‌ها محسوب می‌شود؛ تنها علت رسم جدای آنها بیان جزئیات عملیات انتخاب توسط Multiplicand است.

همانطور که اشاره شد، در این معماری از ۴ expert مختلف Booth استفاده نموده‌ایم که بصورت موازی در کنار یکدیگر قرارگرفته‌اند و در نهایت خروجی خود را به یک انتخاب کننده ارسال می‌کنند. این انتخاب کننده که همان قسمت تصمیم‌گیری نمودار محسوب می‌شود بر اساس تاخیر تولید نتایج تصمیم‌گیری می‌نماید. ورودی همه ماژول‌های Booth کاملا مشابه است. هر ماژول پس از دریافت ورودی‌ها که عبارتند از مضروب و مضروب‌فیه، عمل تولید حاصل‌ضرب‌های جزئی را آغاز نموده و در نهایت نتیجه را به یک ماژول نهایی می‌فرستد. این ماژول نهایی همانطور که اشاره شده یک انتخاب کننده است که وظیفه آن انتخاب سریع‌ترین خروجی ممکن است. با این عمل تضمین می‌شود که بهترین عملکرد از بین عملکردهای مختلف انتخاب خواهد شد. برای پیاده‌سازی این ضرب‌کننده از زبان توصیف‌گر VHDL استفاده نموده‌ایم.

پیاده‌سازی‌های مختلفی برای انتخاب‌کننده اشاره شده وجود دارد [۱]، [۲]، [۴]، [۵]. برای پیاده‌سازی این ماژول در این مقاله به این نکته خاص توجه شده است که بر اساس وظیفه تعریف شده، این بخش بایستی به ازای هر بیت عملیات زیر را (در یک چرخه) مرتبا اجرا نماید:

در ابتدا باید بیت یا بیت‌های خروجی را از مرحله قبل دریافت نماید. این ماژول قابلیت دریافت ۴ بیت همزمان را دارد و این قابلیت بدلیل وجود ۴ Expert در سیستم می‌باشد. تنها شرط دریافت چندین بیت

از مرحله قبل همزمانی در تشخیص آنهاست، در غیر این‌صورت اولین بیت دریافت شده، سریع‌ترین بیت خروجی در آن جایگاه خواهد بود پس باقی خروجی‌ها برای آن جایگاه bypass خواهند شد. پس از انجام این عمل به ازای بیتی در مکان i ام نتیجه سریعا در خروجی سیستم (برای جایگاه i ام) ظاهر خواهد شد.



شکل ۵ معماری نوین ضرب‌کننده

۴- نتایج شبیه‌سازی و سنتز

برای شبیه‌سازی و سنتز این معماری از ابزار Xilinx ISE 6.1 و Model-Sim Ver. 5.6a XE استفاده نموده‌ایم. پس از شبیه‌سازی ماژول‌های مختلف ضرب بصورت جداگانه به بررسی صحت عملکرد آنها پرداخته و پس از آن تمامی ماژول‌ها را با هم شبیه‌سازی نمودیم تا در نهایت از عملکرد کلی پیاده‌سازی نیز اطمینان حاصل نمائیم. لازم به اشاره است که نوع معماری در پیاده‌سازی، ساختاری است. پس از پیاده‌سازی، بهبود قابل توجهی در سرعت عملکرد معماری نوین مشاهده شد که نتایج آن را در جدول ۳ آورده‌ایم. هر p<i> در این جدول نمایش یک بیت از نتیجه شبکه جمع است که برای ارسال به جمع‌کننده نهایی آماده شده است. همانطور که مشخص است، در بین بیت‌های ۰ تا ۹ نتیجه به بهبودی نزدیک به ۱۴٪ تا ۲۱،۵٪ دست پیدا کرده‌ایم. این بهبود بین بیت‌های ۱۰ تا ۶۳ از نتیجه بین ۴٪ تا ۶٪ بوده است.

۵- نتیجه‌گیری

در این مقاله معماری نوینی برای ضرب‌کننده‌ها ارائه شده است. این Expertها پیاده‌سازی الگوریتم‌های مختلف کدگذار Booth می‌باشند. بر این اساس نتایج حاصله از این Expertها (حاصلضرب‌های جزئی) پس از عبور از شبکه جمع این ضرب‌کننده وارد یک ماژول تصمیم‌گیر می‌شوند. معماری پیشنهادی با استفاده از خصوصیت‌ها و مزایای ماژول‌های مختلف ضرب‌کننده و انتخاب بهترین آنها به سرعت بالاتری در تولید خروجی‌ها دست پیدا کرده است. این معماری بر اساس مدل سیستم Multi-Expert ارائه شده است. در این مدل Expert های بصورت موازی فعالیت می‌نمایند، بعبارت بهتر در این معماری از مدل موازی سیستم Multi-expert استفاده شده است. معماری ارائه شده برای، بر اساس نتایج سنتز و شبیه‌سازی توانست به بهبودی درحدود ۱۴٪ تا ۲۱،۵٪ در ۱۰ بیت اول و ۴٪ تا ۶٪ در ۵۴ بیت بعدی حاصل جمع حاصلضرب‌های جزئی دست پیدا کند.

جدول ۳ بهبود سرعت در معماری نوین نسبت به مدل‌های پیشین (P علامت نتیجه است)

	تاخیر معماری پیشنهادی	تاخیر معماری‌های پیشین	بهبود
P<0~9>	از *ns*۷،۲۸ تا *ns*۲۱،۷۲	از *ns*۸،۴۵ تا *ns*۲۷،۶۷	از ٪۱۳،۸ تا ٪۲۱،۵
P<10~63>	از *ns*۲۳،۸۱ تا *ns*۵۱،۱۳	از *ns*۲۴،۸۶ تا *ns*۵۴،۶۶	از ٪۴ تا ٪۶

۶- مراجع

[1] M.C. Fairhurst, A.F.R. Rahman, "A generalised approach to the recognition of structurally similar handwritten characters," IEE Proc. Vision Image Signal Process, 144 (1) (1997) 15-22.
[2] A.F. Gonzalez, P. Mazumder, "Redundant arithmetic, algorithms and implementations," VLSI Journal, Elsevier, 30, pp. 13-53, 2000.
[3] Parhami, B., Computer Arithmetic: Algorithms and Hardware Designs, Oxford, 2000.
[4] A.F.R. Rahman, M.C. Fairhurst, "Enhancing multiple expert decision combination strategies through exploitation of a priori information sources," IEE Proc. Vision Image and Signal Process, 146 (1) (1999) 1-10.
[5] A.S. Constantinidis, M.C. Fairhurst, A.F.R. Rahman, "A new multi-expert decision combination algorithm and its application to the detection of circumscribed masses in digital mammograms," Journal of Pattern Recognition, Elsevier, 34 (2001) 1527-1537.
[6] Fayez Elguibaly, "A Fast Parallel Multiplier-Accumulator Using the Modified Booth Algorithm," IEEE

Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 47, no. 9, September 2000.

[7] O.J. Bedrij, "Carry-Select Adder," IRE Transactions on Electronic Computer, Vol.EC-11, pp. 340-346, 1962.

[8] G. Bewick and M. J. Flynn, "Binary multiplication using partially redundant multiples," Technical Report No. CSL-TR-92-528, Computer Systems Laboratory, Stanford University, June 1992.

[9] G. Bewick, "Fast Multiplication: Algorithms and Implementations," Ph.D. Thesis, Stanford University, February 1994.

[10] A. D. Booth, "A signed binary multiplication technique," Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236-240, 1951.

[11] M. Ercegovac and T. Lang، Digital Arithmetic، Morgan Kaufmann، 2004.

[12] R. P. Brent and H. T. Kung, "A regular layout for Parallel Adders," IEEE Transactions on Computers, vol. 31, no. 3, pp. 260-264, March 1982.

[13] L. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol.34, pp.349-356, March 1965.

[14] J. Fadavi-Ardekani, "MxN Booth Encoded Multiplier Generator Using optimized Wallace Trees," IEEE Transactions on Very Large Scale Integration (VLSI) System, vol. 1, no. 2, pp. 120-125, June 1993.

[15] Israel Koren, Computer Arithmetic Algorithms, 2nd Edition, A.K. Peters, Natick, MA ،2002, ISBN 1-56881-160-8.

[16] Shailesh Shah, "Design and Comparison of 32-bit Multipliers for Various Performance Measures," MEng. Project Report, Concordia University, January 2000.

[17] Lakshmanan, Masuri Othman and Mohamad Alauddin M.A., "High Performance Parallel Multiplier using Wallace-Booth Algorithm," ICSE2002 Proc., Penang, Malaysia, 2002.

[18] J. L. Hennessy and D. A. Patterson, Computer Architecture, a Quantitative Approach, Morgan-Kaufmann, 1990.

[19] Z. Huang and M.D. Ercegovac, "Number representation optimization for low-power multiplier design," Proc. SPIE 2002 Advanced Signal Processing Algorithms, Architectures, and Implementations XII, July 2002.

[20] H. Altwaijry, "Area and Performance Optimized CMOS Multipliers," Ph.D. Thesis, Stailford University, August 1997.

[21] T. Kilburn, D. Edwards, and D. Aspinan, "Parallel Addition in Digital Computers: A new Carry Circuit," IEE Proceedings, vol.106, pt. B pp. 464-466, 1959.

[22] O. Kavehie, A. P. Mirbaha, N. Dadkhahi, K. Navi, "Novel Architecture for IEEE-754 Standard," IEEE Intl. Conf. on Info. & Communication Technologies: From Theory to Applications (ICTTA'06), Syria, April 24-28, 2006.