مجله علمی پژوهشی «علوم و فناوری‌های پدافند غیرعامل»
سال سوم، شماره۳، پاییز ۱۳۹۱؛ ص ۲۱۱-۱۹۹

پیاده‌سازی سخت‌افزاری یک پردازنده رمزنگاری خم بیضوی کارآمد در میدان $GF(2^{163})$

مسعود معصومی[۱*]، حسین مهدیزاده[۲]
۱- استادیار و ۲- کارشناس ارشد دانشگاه آزاد اسلامی واحد اسلامشهر
(دریافت: ۱۳۹۰/۱۰/۲۳، پذیرش: ۰۸/ ۱۳۹۱/۰۹)

چکیده

در این مقاله، پردازنده ضرب عددی خم بیضوی کارآمد در میدان باینری $GF(2^{163})$ طراحی و با استفاده از کدهای قابل سنتز VHDL پیاده‌سازی شده است. طراحی معماری‌های جدید و کارآمد برای واحدهای محاسبات میدان و به‌ویژه واحد محاسباتی ضرب میدان منجر به کاهش طول مسیر بحرانی پردازنده شد. همچنین استفاده از اجرای موازی عملیات ضرب میدان در الگوریتم ضرب عددی Lopez-Dahab و جدا نمودن مسیر جمع دو نقطه از کلید باعث بهبود عملکرد پردازنده در مقایسه با بهترین پیاده‌سازی‌های تاکنون گزارش شده در ادبیات مربوطه شد. نتیجه‌های پیاده‌سازی نشان داد که ضرب عددی در این پردازنده با طول رقم G = ۴۱ در زمان ۱۱/۹۲ μs با حداکثر فرکانس ۲۵۱ MHz بر روی تراشه Xilinx – XC4VLX200 با اشغال ۱۹۶۰۶ اسلایس اجرا می‌شود، جایی که G طول کلمه در ضرب‌کننده میدان سریال- موازی است.

کلیدواژه‌ها: رمزنگاری خم بیضوی، محاسبات میدانی، ضرب عددی نقطه، پیاده‌سازی سخت‌افزاری، FPGA.

The FPGA Implementation of an Efficient Elliptic Curve Cryptographic Process over $GF(2^{163})$

M. Masoumi[*], H. Mahdizadeh
Islamic Azad University, Islamshahr Branch
(Received: 13/01/2012; Accepted: 28/11/2012)

Abstract

A new and highly efficient architecture for elliptic curve scalar point multiplication is presented. To achieve the maximum architectural and timing improvements, the critical path of the Lopez-Dahab scalar point multiplication architecture has been reorganized and reordered such that logic structures are implemented in parallel and operations in the critical path are diverted to noncritical paths. The results show that with G= 41, the proposed design is able to compute $GF(2^{163})$ elliptic curve scalar multiplication in 11.92μs with the maximum achievable frequency of 251 MHz on Xilinx Virtex-4 (XC4VLX200), where G is the digit size of the underlying digit-serial finite field multiplier. The results of synthesis show that in this implementation 19606 slices or 22% of the chip area is occupied.

Keywords: Elliptic Curve Cryptography, Scalar Point Multiplication, FPGA Implementation.

[*] Corresponding Author Email: m_masoumi@eetd.kntu.ac.ir

۱. مقدمه

مأموریت پدافند غیرعامل انجام اقدامات غیرمسلحانه برای محافظت از سامانه‌های شهری و نظامی در مقابل هرگونه اقدام آفندی دشمن است. از این‌رو بالا بردن ضریب امنیت و کارایی سامانه‌های مختلف نظامی و غیرنظامی از جمله مقولات بسیار مهم در بحث پدافند غیرعامل محسوب می‌شود، زیرا وجود هر نوع رخنه یا آسیب‌پذیری امنیتی ممکن است موجب وارد آمدن خسارات جبران‌ناپذیری شود. یکی از مهم‌ترین مقولات در زمینه امنیت بحث رمزنگاری، طراحی و پیاده‌سازی تراشه‌های رمزکننده است. رمزنگاری کلید عمومی مبتنی بر خم بیضوی[1] اولین بار در سال ۱۹۸۵ میلادی مطرح شد. امنیت این مکانیزم رمزنگاری بر دشواری مسئله لگاریتم گسسته بر روی خم‌های بیضوی یا ECDLP[2] استوار است و الگوریتم‌هایی که برای حل مسئله ECDLP وجود دارد، زمان اجرای نمایی دارند. مشخصه بارز روش رمزنگاری خم بیضوی، یعنی طول کلید کوتاه آن، سبب شده است که به یک سطح امنیتی یکسان در مقایسه با کلیدهای بزرگ روش RSA برسد. به‌طور مثال کلید ۱۶۳ بیتی رمزنگاری خم بیضوی سطح امنیتی یکسانی با کلید ۱۰۲۴ بیتی RSA دارد [۲ و ۱]. در میان بسترهای پیاده‌سازی، تراشه‌های FPGA به‌خاطر دارا بودن مزیت‌هایی از قبیل زمان، هزینه طراحی و پیاده‌سازی کم، انعطاف پذیری، قابلیت پیکربندی مجدد و کارایی بالا در طراحی و پیاده‌سازی الگوریتم‌های رمزنگاری مورد توجه هستند که بسترهای پرکاربرد هستند [۳].

در این مقاله، معماری جدیدی برای پیاده‌سازی رمزنگاری خم بیضوی بر روی FPGA ارائه کرده‌ایم که در مقایسه با سایر کارهای گزارش شده در ادبیات مربوطه تاکنون از نظر سرعت و سطح اشغالی برتر است. برای انجام محاسبات میدانی الگوریتم‌های مؤثری انتخاب شده است که با طراحی معماری‌های مناسب در پیاده‌سازی آنها، مسیر بحرانی به‌طور چشمگیری کاهش یافته است. مهم‌ترین و تأثیرگذارترین واحد محاسباتی در پردازنده ضرب‌کننده میدانی است که در این کار با استفاده از ضرب‌کننده سریال- موازی LSD و استفاده از ایده گراف درخت در XOR کردن دو رشته بیت، طول مسیر بحرانی این واحد محاسباتی کاهش یافته است. استفاده از واحد محاسباتی ضرب‌کننده بهبود یافته، استفاده از دو واحد مربع‌کننده برای جدا کردن مسیر مربع‌هایی که چندین مرتبه باید تکرار شوند و نیز استفاده از حداکثر موازی‌سازی منجر به افزایش کارایی واحد محاسباتی معکوس‌کننده میدان Itoh-Tsuji[3] شده است. استفاده از الگوریتم ضرب عددی Lopez-Dahab (LD) و پیاده‌سازی سه ضرب‌کننده هم‌زمان به‌طور موازی باعث کاهش تأخیر انجام محاسبات مختصات تصویری[4] شد. کاهش یافتن تأخیر در انجام محاسبات مختصات تصویری و کاهش سطح پیاده‌سازی در محاسبات مربوط به تبدیل مختصات تصویری به آفینی، منجر به نتیجه‌های بهبود یافته‌ای در مقایسه با سایر کارها شد، به‌طوری که ضرب عددی با طول رقم G =۴۱ در این پردازنده در زمان ۱۱/۹۲ μs با حداکثر فرکانس ۲۵۱ MHz بر روی تراشه Xilinx – XC4VLX200 با اشغال ۱۹۶۰۶ اسلایس[5] اجرا می‌شود که G طول کلمه در ضرب‌کننده میدانی سریال- موازی است. همچنین ضرب عددی با طول رقم G = ۲۴ در ۱۵/۶ μs با فرکانس ۲۵۴ MHz اجرا می‌شود، درحالی‌که سطح اشغالی در این حالت ۱۵۹۵۳ اسلایس است. می‌توان با انتخاب طول کلمه G = ۴۱ از پردازنده برای کاربردهای فرکانس بالا و سریع استفاده کرد و با انتخاب طول کلمه G = ۲۴ می‌توان سطح پیاده‌سازی را کاهش داده، به یک مصالحه در سرعت و مساحت رسید.

از آنجا که در این مقاله هدف پیاده‌سازی، افزایش سرعت و کاهش مساحت اشغالی بر روی بستر FPGA است، از نمایش چند جمله‌ای در میدان باینری به‌علت امکان پیاده‌سازی مؤثر استفاده شده است. همچنین به‌علت استفاده از الگوریتم ضرب عددی مونتگمری، خم بیضوی نا ابر منفرد[6] در میدان باینری طبق پیشنهاد NIST[7] انتخاب شده است که کوچک‌ترین میدان باینری پیشنهادی $GF(2^{163})$ است [۸-۴] در ادامه مقاله، در قسمت ۲ به‌طور مختصر یک پیش‌زمینه ریاضی از محاسبات مربوط به خم‌های بیضوی را بیان می‌کنیم. در قسمت ۳ برخی از کارهای قبلی انجام شده درباره پیاده‌سازی ضرب عددی خم بیضوی بر روی FPGA را مرور خواهیم کرد. در قسمت ۴، الگوریتم‌های مورد استفاده در محاسبات مربوط به میدان‌های متناهی را با جزئیات بیشتر بررسی کرده و ساختار پیاده‌سازی واحدهای محاسبات میدانی را بر روی سخت‌افزار بیان می‌کنیم. در قسمت ۵ معماری پردازنده ضرب عددی خم بر روی FPGA را پیشنهاد می‌دهیم. در قسمت ۶ نتیجه‌های پیاده‌سازی را ارائه کرده و با سایر کارهای مشابه قبلی مقایسه می‌کنیم. درنهایت در قسمت نتیجه‌گیری به جمع‌بندی موضوع پرداخته و نتیجه‌های نهایی را ارائه می‌دهیم.

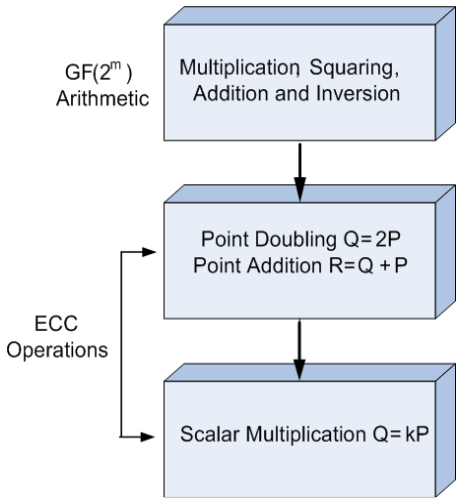
۲. رمزنگاری خم بیضوی

عملیات ضرب عددی خم بیضوی، ضرب یک مقدار عددی در یک نقطه روی خم است. این ضرب تنها در گروه نقاط خم بیضوی تعریف می‌شود و برخلاف سایر ضرب‌ها از عمل جمع دو نقطه[8] و دو برابر کردن یک نقطه[9] براساس یک یا صفر بودن بیت‌های مقدار عددی به‌دست می‌آید. معادله $Q = kP$ را درنظر بگیرد، به‌طوری که $P, Q \in E(F_q)$ که E معادله خم و q یک عدد اول و یا توانی از یک از عدد اول است که $k < p$. اگر معادله خم E و میدان q براساس استاندارد NIST انتخاب شوند آن‌گاه با در اختیار داشتن P و Q به‌دست آوردن k به مسئله لگاریتم گسسته بر روی خم بیضوی

[1] Elliptic Curve Cryptography
[2] Elliptic Curve Discrete Logarithm Problem (ECDP)
[3] Itoh-Tsuji Multiplicative Inverse (ITIMA)
[4] Projective Computations

[5] Slice
[6] Non-Supersingular
[7] National Institute of Standards and Technology
[8] Point Addition
[9] Point Doubling

منتهی می‌شود. هرم محاسبات مورد نیاز برای ضرب عددی نقطه خم بیضوی در شکل (۱) نشان داده شده است.



شکل۱. مدل سه لایه‌ای برای ضرب عددی خم بیضوی

نمایش نقاط خم بیضوی بر مبنای مختصات دوبعدی است. اما علاوه بر مختصات دوبعدی برای نمایش نقاط، مختصات تصویری[1] سه‌بعدی بعدی نیز وجود دارد که استفاده از آن منجر به محاسبات کم هزینه‌تر خواهد شد، زیرا برای انجام عمل جمع و دو برابر کردن نقطه نیازی به عمل معکوس میدانی نیست. بنابراین نقطه P در مختصات آفینی به‌صورت دوبعدی (x,y) نمایش داده می‌شود که همین نقطه در مختصات تصویری به‌صورت سه‌بعدی (X,Y,Z) نشان داده می‌شود. مطابق با آنچه در ادبیات و منابع مربوطه گزارش شده است، مختصات تصویری LD به‌دلیل پیچیدگی کمتر در محاسبات، برای پیاده‌سازی سخت‌افزاری مناسب‌تر از سایر الگوریتم‌ها است. پس از تعیین مختصات مناسب برای انجام محاسبات، لازم است تا یک الگوریتم ضرب عددی کارآمد انتخاب شود. در ارزیابی الگوریتم‌های ضرب عددی پارامترهای زمان اجرای الگوریتم و حافظه اشغالی براساس تعداد عملیات ضرب میدانی و تعداد نقاط ذخیره شده برای پیش محاسبات بررسی می‌شوند.

در این مقاله، الگوریتم ضرب عددی LD در مختصات تصویری [۹] که دارای عملکرد بالایی بوده، استفاده شده است، این روش یک پیاده‌سازی مؤثر از الگوریتم ضرب عددی مونتگمری ارائه شده در مرجع [۱۰] است. در شکل (۲) الگوریتم ضرب عددی LD نشان داده شده است. این الگوریتم را می‌توان به سه مرحله تقسیم کرد. محاسبات ابتدای الگوریتم که تبدیل مختصات نقطه دریافتی به‌عنوان ورودی از آفینی به تصویری است، محاسبات حلقه اصلی که براساس بیت‌های کلید عملیات جمع و دو برابر کردن نقطه در مختصات تصویری انجام می‌شود و محاسبات انتهای الگوریتم که شامل تبدیل مختصات تصویری به آفینی برای تعیین حاصل kP در مختصات آفینی است، زیرا در مختصات آفینی فضای کمتری اشغال می‌شود و چون حاصل نهایی به بیرون پردازنده ارسال می‌شود، در مختصات آفینی پهنای باند کمتری اشغال می‌شود.

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$ with $k_{t-1} = 1$, $P = (x_P, y_P) \in E(F_{2^m})$.
OUTPUT: kP.
1. $X_1 \leftarrow x_P$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x_P^4 + b$, $Z_2 \leftarrow x_P^2$. {Compute $(P, 2P)$}
2. For i from $t-2$ downto 0 do
  2.1 If $k_i = 1$ then
    $T \leftarrow Z_1$, $Z_1 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_1 \leftarrow x_P Z_1 + X_1 X_2 T Z_2$.
    $T \leftarrow X_2$, $X_2 \leftarrow X_2^4 + bZ_2^4$, $Z_2 \leftarrow T^2 Z_2^2$.
  2.2 Else
    $T \leftarrow Z_2$, $Z_2 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_2 \leftarrow x_P Z_2 + X_1 X_2 Z_1 T$.
    $T \leftarrow X_1$, $X_1 \leftarrow X_1^4 + bZ_1^4$, $Z_1 \leftarrow T^2 Z_1^2$.
3. $x_3 \leftarrow X_1/Z_1$.
4. $y_3 \leftarrow (x_P + X_1/Z_1)[(X_1 + x_P Z_1)(X_2 + x_P Z_2) + (x_P^2 + y)(Z_1 Z_2)](x_P Z_1 Z_2)^{-1} + y_P$.
5. Return (x_3, y_3)

شکل ۲. الگوریتم ضرب عددی LD روی میدان $GF(2^m)$ [۹]

۳. پیشینه تحقیق

در این قسمت چند نمونه از بهترین پیاده‌سازی‌های ضرب عددی خم بیضوی بر روی سخت‌افزار FPGA را که در سال‌های اخیر انجام شده را بررسی می‌کنیم. در این کارها به‌منظور تسریع در اجرای عملیات ضرب عددی از تکنیک‌هایی همچون طراحی موازی معماری یا پیش محاسبات استفاده شده است.

پیاده‌سازی انجام شده توسط اورلاند با استفاده از الگوریتم LD بوده که روی میدان $GF(2^{167})$ انجام شده و ضرب‌کننده میدانی در آن با تأخیر یازده پریود اجرا می‌شود [۱۱]. عملیات ضرب عددی این کار در ۲۱۰ ms اجرا می‌شود. در گزارش دیگری برای چند طول میدان مختلف ضرب عددی برای بسترهای سخت‌افزاری و نرم‌افزاری اجرا شده است [۱۲]. در این کار ضرب‌کننده میدانی با تأخیر سه سیکل اجرا می‌شود. برای میدان $GF(2^{163})$ عمل ضرب عددی در ۱۴۴ μs اجرا می‌شود.

جاروینن و همکارانش از دو ضرب‌کننده موازی به‌صورت هم‌زمان در الگوریتم ضرب عددی LD استفاده می‌کنند، به‌طوری که این ضرب‌کننده دارای چندین رجیستر در مسیر بحرانی برای افزایش فرکانس کاری مدار است، اما به‌علت اینکه در طراحی آنها از خط لوله[2] استفاده نشده است، پیاده‌سازی آنها حجیم و دارای تأخیر زیادی است [۱۳].

یک پیاده‌سازی FPGA گزارش شده که جمع و دو برابر کردن را به‌صورت موازی اجرا کرده است و از چند تابع محاسباتی دیگر نیز به‌طور هم‌زمان استفاده می‌کند [۱۴]. البته به‌نظر می‌رسد که در نتیجه‌های ارائه شده هزینه تبدیل مختصات و معکوس میدانی درنظر گرفته نشده است. به‌هر حال این کار یکی از سریع‌ترین کارهایی است که در مقالات موجود است، اما همچنان فرکانس کارکرد مدار به‌علت پیچیده بودن مسیر بحرانی ضرب‌کننده پایین و برابر ۴۶/۵ MHz است.

لوتز و حسن از تکنیک پیش محاسبات برای اجرای ضرب عددی

[1] Projective Coordinates

[2] Pipelining

استفاده می‌کنند [۱۵]. ضرب میدانی مورد استفاده در این کار ضرب‌کننده سریال- موازی LSD است. در این کار برای اجرای معکوس برای تبدیل مختصات از تکرار مربع کردن به‌گونه‌ای استفاده شده که هزینه معکوس کاهش یابد.

ضرب عددی در دو خانواده خم‌های ابرمنفرد و نا ابر منفرد مقایسه شده است [۱۶]. نتیجه‌های این پیاده‌سازی روی بستر Virtex II Pro 30 زمان ۲۸۰μs با فرکانس ۱۰۰ MHz و اشغال ۸۴۵۰ اسلایس را نشان می‌دهد. در گزارش دیگری پردازنده ضرب عددی خم بر مبنای مجموعه دستورالعمل‌های خاص خط لوله شده، ارائه شده است [۱۷]. این کار بر روی Xilinx Virtex-4 (XC4VLX200) در زمان μs ۱۹/۵۵ با فرکانس ۱۵۹/۳MHz و اشغال ۱۶۲۰۹ اسلایس اجرا می‌شود.

انصاری و حسن یک معماری رمزنگاری خم بیضوی براساس الگوریتم مونتگمری پیشنهاد داده‌اند [۱۸]. در این کار از یک ضرب‌کننده سریال موازی شبه لوله‌گذاری[1] شده استفاده شده است و عملیات ضرب عددی در میدان $GF(2^{163})$ بر روی Xilinx XC2V2000 در μs ۴۶/۵ با حداکثر فرکانس ۱۰۰ MHz اجرا می‌شود. ویژگی قابل ملاحظه این طراحی فشرده بودن آن است، به‌طوری که LUTs[2] ۷۵۵۹ را اشغال می‌کند.

یانگ و همکارانش روی پیاده‌سازی سخت‌افزاری سرعت بالای رمزنگاری خم بیضوی بر روی FPGA متمرکز شده‌اند [۱۹]. آنها الگوریتم ضرب عددی را با اجرای موازی آن و طراحی یک معماری جدید بهینه کرده‌اند. این کار بر روی تراشه Xilinx XC2V6000 در میدان $GF(2^{163})$ با تأخیر μs ۳۴/۱۱ و فرکانس ۹۳/۳ MHz اجرا می‌شود. یک نسخه موازی شده از الگوریتم LD ارائه شده است[۲۰]. ضرب‌کننده میدانی در مبنای نرمال گوسی با سه کلمه ۵۵ بیتی به‌کار رفته است. نکته قابل توجه در این کار پیاده‌سازی بسیار ساده مربع با توجه به مبنای نمایش نرمال است. زمان اجرای ضرب عددی در این کار μs ۱۰ است اما سطح مورد نیاز پیاده‌سازی به‌نسبت بزرگ و برابر ۲۴۳۶۳ اسلایس است.

۴. معماری سخت‌افزاری محاسبات میدان متناهی روی $GF(2^m)$

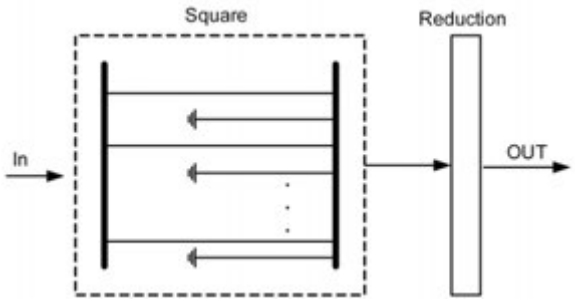
در این قسمت الگوریتم‌ها و تکنیک‌هایی را که برای پیاده‌سازی مؤثر محاسبات در میدان‌های متناهی $GF(2^m)$ به‌کار می‌رود، به‌طور مختصر مرور می‌کنیم. واحدهای محاسباتی طراحی شده در این قسمت در معماری پردازنده ضرب عددی خم بیضوی به‌کار می‌رود. جمع و تفاضل دو عنصر متعلق به میدان باینری ساده‌ترین عملیات ریاضی میدان است، زیرا بیت‌های هر عنصر با بیت‌های مشابه (از نظر ارزش مکانی) عنصر دیگر XOR می‌شوند و نیازی به کاهش در میدان نیست [۱۹ و ۲].

۴-۱. مربع میدان متناهی[3]

یکی از واحدهای محاسباتی که چندین مرتبه از آن در پیاده‌سازی پردازنده ضرب عددی LD استفاده می‌شود، مربع‌کننده است. از محاسن مربع در میدان‌های باینری پیاده‌سازی ساده آن است. فرض کنید چند جمله‌ای $A = \sum_{i=0}^{m-1} a_i x^i$ در میدان $GF(2^m)$ تعریف شده باشد، مربع A به‌صورت زیر نمایش داده می‌شود:

$$D(x) = A(x)\,A(x) = A^2(x) = \left(\sum_{i=0}^{m-1} a_i\right)\left(\sum_{i=0}^{m-1} a_i\right) = \sum_{i=0}^{m-1} a_i x^{2i} \quad (۱)$$

از این معادله می‌توان فهمید که $k<m$ بیت اول A، $2k$ بیت اول A^2 را تعیین می‌کنند. نکته قابل توجه آنکه باعث سهولت پیاده‌سازی مربع‌کننده می‌شود، صفر بودن نیمی از بیت‌های A^2 است؛ زیرا بیت‌های A به بیت‌ها با اندیس زوج A^2 نگاشته می‌شوند و کلیه جمله‌ها با اندیس فرد A^2 صفر است. مرحله بعد این است که جمله‌های بزرگ‌تر از اندازه میدان در A^2 را با استفاده از کاهش به اندازه میدان مورد نظر یعنی $GF(2^m)$ بنگاریم. حال اگر عمل مربع و کاهش را با همدیگر ترکیب کرده به‌گونه‌ای که مربع میدانی در یک سیکل انجام شود و در عین حال کاهش تأخیر و منابع مصرفی رعایت شود، یک واحد محاسباتی کارآمد پیاده‌سازی شده است. واحد محاسباتی مربع‌کننده در این مقاله به‌صورتی ترکیبی و در یک سیکل، مطابق با کار انجام شده در گزارش دیگری [۱۹] طراحی شده است (شکل ۳).



شکل ۳. مدار مربع کردن در میدان‌های باینری [۲]

۴-۲. کاهش میدان متناهی[4]

فرض کنید چند جمله‌ای تحویل‌ناپذیر $P(x)$ در میدان متناهی $GF(2^m)$ تعریف شده است، همچنین $A(x)$ و $B(x)$ نیز اعضایی از این میدان باشند یا به‌عبارتی دیگر $A(x), B(x) \in GF(2^m)$. ضرب دو چند جمله $A(x)$ و $B(x)$ به‌صورت $D(x)= A(x)B(x)$ تعریف می‌شود که حاصل این ضرب به پیمانه چند جمله‌ای کاهشی $P(x)$ به‌عنوان یک ضرب میدانی با $C(x)$ در زیر نشان داده شده است.

$$C(x) = D(x) \bmod P(x) \quad (۲)$$

یادآوری می‌شود که حاصل ضرب چند جمله‌ای $D(x)$ و حاصل ضرب پیمانه‌ای $C(x)$ که به‌ترتیب $2m-1$ و m جمله دارند، در زیر آمده است، یا به‌عبارتی:

[1] Psudo Pipeline
[2] Lookup Table
[3] Finite Field Squaring
[4] Finite-Field Reduction

$$D = [d_{2m-2}, d_{2m-3}, \dots, d_{m+1}, d_m, \dots, d_1, d_0];$$
$$C = [c_{m-1}, c_{m-2}, \dots, c_1, c_0]; \quad (۳)$$

$C(x)$ از XOR کردن و جابه‌جایی جملات $D(x)$ براساس موقعیت جملات چند جمله‌ای کاهشی $P(x)$ تعیین می‌شود. چند جمله‌ای کاهشی $P(x)$ به‌عنوان یک نگاشت خطی، $2m\text{-}1$ جمله $D(x)$ را به m جمله $C(x)$ می‌نگارد. این نگاشت به‌شکل ماتریسی زیر نشان داده شده است:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 & r_{0,0} & \dots & r_{0,m-2} \\ 0 & 1 & \dots & 0 & r_{1,0} & \dots & r_{1,m-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & 1 & r_{m-1,0} & & r_{m-1,m-2} \end{bmatrix} \begin{bmatrix} d_0 \\ \vdots \\ d_{m-1} \\ d_m \\ \vdots \\ d_{2m-2} \end{bmatrix} \quad (۴)$$

به‌طوری که،

$$r_{j,i} = \begin{cases} P_j \; ; \; j = 0, \cdots, m-1; i = 0 \\ r_{j-1,i-1} + r_{m-1,i-1} r_{j,0} \; ; \; j = 0, \cdots, m-1; i = 1, \cdots, m-2 \end{cases} \quad (۵)$$

پیاده‌سازی ماتریس بالا برای مقادیر مختلف m باعث افزایش تعداد گیت‌های منطقی مورد نیاز و درنتیجه افزایش سطح پیاده‌سازی می‌شود. یکی از رویکردهای بسیار مؤثر در پیاده‌سازی سخت‌افزاری کاهش در میدان‌های متناهی استفاده از الگوریتم‌های کاهش سریع ویژه چند جمله‌ای کاهشی مورد استفاده در آن میدان است. شکل (۴) پیاده‌سازی کاهش برای چند جمله‌ای کاهشی مورد استفاده در این مقاله را نشان می‌دهد.

$$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$$

در این الگوریتم فرض شده است که حداکثر درجه $D(x)$ برابر $162+G$ است که باید جملات d_i با درجه‌های $162+G \le i \le 163$ به جملات با درجه‌های $i<163$ نگاشته شوند [۲۱].

Input: D $= [d_{162+G}, d_{161+G}, \dots, d_1, d_0]$,
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$;
Output: $C = [c_{162}, c_{161}, \dots, c_1, c_0]$;

if $G = 0$ then
  $C \leftarrow D$;
else
  $[c_{162}, \dots, c_G] \leftarrow [d_{162-G}, \dots, d_0]$;
  $[c_{G-1}, \dots, c_0] \leftarrow 0$; for i from 1 to G do
  $c_{i-1} \leftarrow c_{i-1}$ xor $d_{163+i-1}$;
  $c_{3+i-1} \leftarrow c_{3+i-1}$ xor $d_{163+i-1}$;
  $c_{6+i-1} \leftarrow c_{6+i-1}$ xor $d_{163+i-1}$;
  $c_{7+i-1} \leftarrow c_{7+i-1}$ xor $d_{163+i-1}$;
Return C;

شکل ۴ . الگوریتم کاهش برای $C(x) = D(x) \bmod P(x)$

۴-۳. ضرب‌کننده میدان متناهی[1] پیشنهادی

انجام عمل ضرب میدانی مهم‌ترین عملیات در پردازنده ضرب عددی است، به‌طوری که اغلب در تعیین فرکانس کاری مدار و میزان سطح اشغالی تراشه FPGA به‌طور کامل تأثیرگذار است. بنابراین استفاده از یک ضرب‌کننده کارآمد دارای اهمیت است. ضرب‌کننده‌های میدانی مورد استفاده در پیاده‌سازی الگوریتم‌های ضرب عددی به‌طور کلی به سه دسته ضرب‌کننده‌های سری، موازی و سری- موازی تقسیم می‌شوند. ضرب‌کننده‌های سری دارای سطح پیاده‌سازی کوچکی هستند ولی به‌دلیل این که عمل ضرب در m سیکل انجام می‌شود، سرعت آنها پایین است [۲۳-۲۲]. ضرب‌کننده‌های موازی دارای سطح اشغالی بزرگی‌اند اما به‌دلیل انجام عمل ضرب در یک سیکل دارای سرعت بالایی می‌باشند [۲۶-۲۴]. ضرب‌کننده‌های سری- موازی برای کاربردهای رمزنگاری بسیار مناسب‌اند، زیرا بهینه‌سازی مشترک در زمان اجرا و سطح پیاده‌سازی توسط این ضرب‌کننده‌ها امکان‌پذیر است [۲۸-۲۷]. ضرب‌کننده‌های سری- موازی به دو دسته کم ارزش‌ترین کلمه[2] (LSD) و پر ارزش‌ترین کلمه[3] (MSD) تقسیم می‌شوند. نام‌گذاری بر این اساس است که ضرب از کلمه کم ارزش یا پر ارزش ضرب‌شونده شروع شده است. ضرب‌کننده‌ای که برای پیاده‌سازی در این مقاله استفاده شده، ضرب‌کننده LSD است. این ضرب‌کننده در مقایسه با ضرب‌کننده MSD گیت‌های کمتری مصرف کرده و مسیر بحرانی کوتاه‌تری دارد. برای پیاده‌سازی مؤثر ضرب LSD ایده‌های مختلفی وجود دارد. اولین ایده در سال ۱۹۹۸ برای کاربردهای با توان مصرفی پایین در مرجع [۲۷] مطرح شد که این معماری در مرجع [۲۸] بهبود پیدا کرده است.

در میدان $GF(2^m)$ ، یک عضو این میدان که دارای m جمله است را درنظر بگیرید تعداد کلمات با اندازه G جمله در این عضو از میدان برابر است با $n = \lceil m/G \rceil$.

فرض کنید $A = \sum_{j=0}^{m-1} a_j \alpha^j$ و $B = \sum_{j=0}^{m-1} b_j \alpha^j$ به‌طوری که

$$B_i = \begin{cases} \sum_{j=0}^{G-1} b_{G*i+j} \alpha^j & 0 \le i \le n-2 \\ \sum_{j=0}^{m-1-G(n-1)} b_{G*i+j} \alpha^j & i = n-1 \end{cases} \quad (۶)$$

$$C = A * B \bmod P(x) = \sum_{j=0}^{m-1} c_j \alpha^j$$
$$= \begin{pmatrix} B_0 A + B_1 \left(A\alpha^G \bmod f(x) \right) \\ + B_2 \left(A\alpha^G \cdot \alpha^G \bmod P(x) \right) \\ + B_{n-1} \left(A\alpha^{G*(n-2)} \cdot \alpha^G \bmod P(x) \right) \end{pmatrix} \bmod P(x) \quad (۷)$$

الگوریتم LSD در شکل (۵) خلاصه شده است [۲۷].

ضرب کلاسیک دو مرحله‌ای در میدان‌های باینری مستلزم ضرب دو چند جمله‌ای و کاهش به پیمانه چند جمله‌ای تحویل‌ناپذیر است. ضرب چند جمله‌ای‌های $A(x)$ و $B(x)$ برابر $D(x)=A(x)\times B(x)$ است که $D(x)$ یک چند جمله‌ای با حداکثر درجه $2m-2$ است که به‌صورت معادله (۸) نوشته می‌شود.

$$D = \begin{cases} \sum_{i=0}^{k} a_i b_{k-i} \; ; \; k = 0, \cdots, m-1 \\ \sum_{i=k}^{2m-2} a_{k-i+(m-1)} b_{i-(m-1)} \; ; \; k = m, \cdots, 2m-2 \end{cases} \quad (۸)$$

حال به‌منظور این که ضرب LSD را با استفاده از ضرب کلاسیک نمایش دهیم، ضرب دو چند جمله‌ای را به‌صورت ماتریسی نمایش می‌دهیم. مشخص است که در هر مرحله که نیاز به عمل ضرب چند

[1] Finite Field Multiplication

[2] Least-Significant Digit
[3] Most-Significant Digit

جمله‌ای‌هاست، چند جمله‌ای $A(x) \in GF(2^m)$ که دارای m جمله است در کم ارزش‌ترین کلمه $B(x) \in GF(2^m)$ که دارای G جمله بوده، ضرب می‌شود. این ماتریس دارای سه قسمت است. قسمت بالا یک ماتریس بالا مثلثی صفر است، قسمت میانی یک زیر ماتریس $(m-G+1)\times G$ است و قسمت پایین یک زیر ماتریس پایین مثلثی صفر است.

Input: $A, B \in GF(2^m)$
Output: $C \in GF(2^m)$, $C = AB$ over $GF(2^m)$
Set: $A^{(0)} = A$, $D^{(0)} = 0$, $n = \lceil m/G \rceil$
for i from 1 to n do
1) $A^{(i)} = A^{(i-1)}\alpha^G \bmod P(x)$,
2) $D^{(i)} = A^{(i-1)} \cdot B_{i-1} + D^{(i-1)}$
Where
$A^{(i)} = \sum_{j=0}^{m-1} A_j^{(i)} \alpha^j$
$D^{(i)} = \sum_{j=0}^{m+G-2} d_j^{(i)} \alpha^j$ and
$B_i = \begin{cases} \sum_{j=0}^{G-1} b_{G*i+j}\alpha^j & 0 \le i \le n-2 \\ \sum_{j=0}^{m-1-G*(n-1)} b_{G*i+j}\alpha^j & i = n-1 \end{cases}$
end for
3) **Return** $C = D^{(n)} \bmod P(x)$

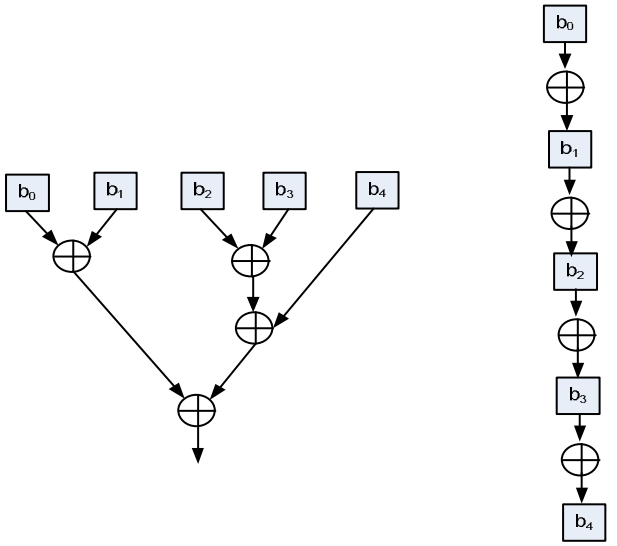
شکل ۵. الگوریتم ضرب *LSD*

$$\begin{bmatrix} a_0 & 0 & 0 & \cdots & 0 \\ a_1 & a_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{G-2} & a_{G-3} & \cdots & a_0 & 0 \\ a_{G-1} & a_{G-2} & \cdots & a_0 & 0 \\ a_G & a_{G-1} & \cdots & a_2 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{m-1} & a_{m-2} & \cdots & a_{m-(G-1)} & a_{m-G} \\ 0 & a_{m-1} & a_{m-2} & \cdots & a_{m-(G-1)} \\ 0 & 0 & a_{m-1} & \cdots & a_{m-(G-2)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{m-1} \end{bmatrix} * \begin{bmatrix} b_0 \\ \vdots \\ b_{G-1} \end{bmatrix}_{G*1} = \begin{bmatrix} d_0 \\ \vdots \\ d_{m+G-2} \end{bmatrix}_{(m+G-1)*1} \quad (۹)$$

نکته قابل توجه در پیاده‌سازی ماتریس بالا این است که چنانچه برای کاهش سطح پیاده‌سازی ضرب، صفرها را کنار بگذاریم، گیت‌های AND کمتری برای انجام محاسبات نیاز می‌شود. اگر ضرب هر کلمه از $B(x)$ را در معادله (۸) براساس ماتریس معادله (۹) انجام دهیم، پیاده‌سازی مناسبی از ضرب دو چند جمله‌ای انجام داده‌ایم. برای بررسی جزئی‌تر ضرب دو چند جمله‌ای G امین جمله چند جمله‌ای $D(x)$ یا d_G براساس معادله (۱۰) به‌صورت زیر بیان می‌شود:

$$d_G = a_{G-1}b_0 + a_{G-2}b_1 + a_0 b_{G-1} \quad (۱۰)$$

که G نشان‌دهنده طول کلمه در ضرب‌کننده LSD است. اگر عبارت‌های معادله (۱۰) را که باید با همدیگر XOR شوند، به‌صورت یک رشته بیت درنظر بگیریم، برای رسیدن به d_G می‌توان جملات این رشته بیت را به‌صورت زنجیره‌وار با همدیگر XOR کرد. استفاده از XOR کردن به‌روش مستقیم همان‌گونه که در شکل (۶- ب) مشاهده می‌شود منجر به مسیر بحرانی به‌طول $(G-1)\Delta_{XOR}$ می‌شود که Δ_{XOR} تأخیر یک گیت XOR است. حال اگر برای XOR کردن این رشته بیت از گراف درخت مطابق آنچه که در شکل (۶- سمت چپ) آمده، استفاده شود؛ با قرار گرفتن رشته بیت در سرشاخه‌ها می‌توان به پاسخ نهایی که همان ریشه درخت بوده، رسید. استفاده از این روش منجر به شکل‌گیری مسیر بحرانی با طول $\lceil log_2(G) \rceil . \Delta_{XOR}$ می‌شود که در مقایسه با XOR کردن به‌روش مستقیم که در شکل (۶- سمت راست) آمده، مسیر بحرانی کوچک‌تری را ایجاد کرده است.

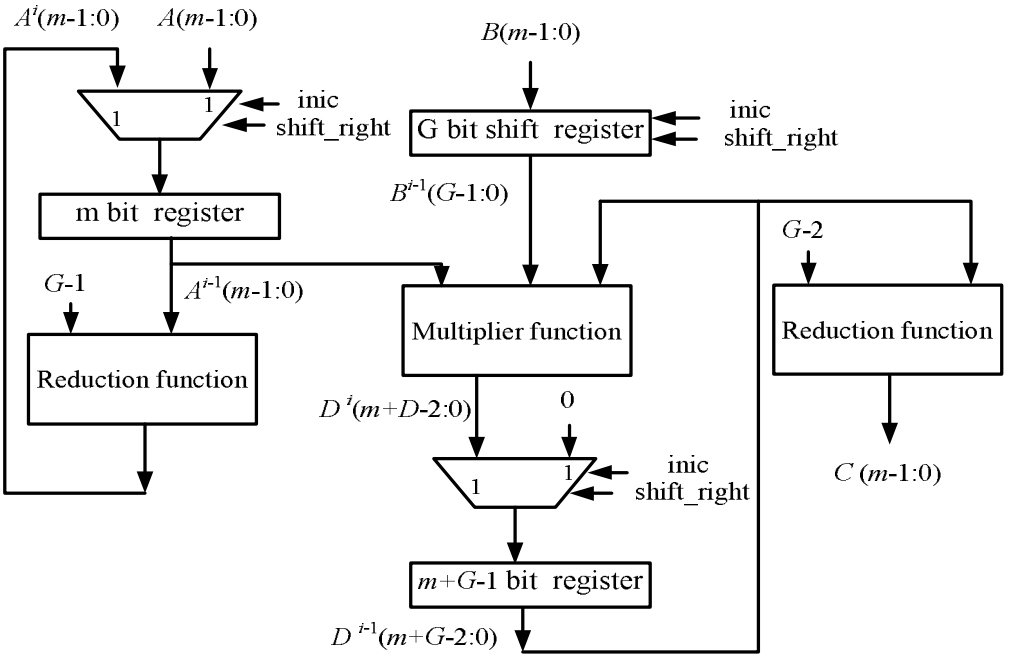


شکل ۶. XOR کردن پنج بیت با همدیگر با استفاده از یک گراف درخت با روش مستقیم

اکنون معماری پیاده‌سازی الگوریتم ضرب میدانی LSD را مطرح می‌کنیم. همان‌گونه که در شکل (۵) مشاهده می‌شود سه مرحله برای پیاده‌سازی این الگوریتم وجود دارد. مراحل ۱ و ۲ الگوریتم ضرب LSD را که در معادله (۱۱) آمده است، می‌توان به‌صورت موازی و هم‌زمان پیاده‌سازی کرد.

$$A^{(i)} = A^{(i-1)}\alpha^G \bmod P(x)$$
$$D^{(i)} = A^{(i-1)} \cdot B_{i-1} + D^{(i-1)} \quad (۱۱)$$

مرحله ۱ الگوریتم ضرب LSD برای کاهش $m+G$ بیت به m بیت استفاده می‌شود و مرحله دوم، ضرب کلمه در چند جمله میدان را نشان می‌دهد. در مرحله سوم نیز برای رسیدن به پاسخ نهایی، $m+G-1$ بیت به m بیت کاهش داده می‌شود. معماری مراحل مختلف پیاده‌سازی الگوریتم ضرب LSD در شکل (۷) ترسیم شده است. مرحله ۱ توسط تابع کاهش سمت چپ شکل (۷)، مرحله دوم به وسیله تابع ضرب و مرحله ۳ نیز به‌وسیله تابع کاهش سمت راست شکل (۷) اجرا می‌شود. اکنون پس از بررسی مسیر بحرانی ضرب‌کننده LSD، تعداد گیت‌های مورد نیاز برای پیاده‌سازی آن را محاسبه می‌کنیم.



شکل ۷. معماری پیاده‌سازی ضرب‌کننده LSD در این کار

تعداد گیت‌های AND در مرحله دوم الگوریتم LSD با استفاده از ماتریس معادله ۹ برابر است با:

$$\#AND = G \times (G - 1) + (m - G + 1) \times G = m \times G$$

برای محاسبه تعداد گیت‌های XOR بایستی تعداد این گیت‌ها را در مراحل ۱، ۲ و ۳ الگوریتم LSD (شکل ۵) محاسبه کنیم. تعداد گیت XOR مورد نیاز مراحل ۱ و ۳ که به‌عنوان کاهش میدانی استفاده می‌شوند با توجه به الگوریتم شکل (۴) و حداکثر درجه‌ای که باید کاهش دهند، برابر خواهد بود با:

$$\#XOR = 4 \times G + 4 \times (G - 1)$$

تعداد گیت XOR مورد نیاز در مرحله دوم الگوریتم LSD برای ضرب کلمه‌ای $A^{(i-1)} \cdot B_{i-1}$ و همچنین XOR کردن حاصل به‌دست آمده از ضرب کلمه‌ای با مقدار $D^{(i-1)}$ برابر است با:

$$\#XOR = (m - 1)(G - 1) + m + G - 1 = m \times G$$

در مرحله اول الگوریتم ضرب LSD، برای به‌دست آوردن $A^{(i)}$ نیاز است که $A^{(i-1)}$ از مرحله قبل در یک رجیستر m بیتی ذخیره شود. بنابراین تعداد فیلیپ فلاپ مورد نیاز در این مرحله برابر خواهد بود با:

$$\#FF = m$$

در مرحله دوم الگوریتم ضرب LSD، برای به‌دست آوردن $D^{(i)}$ نیاز است که $D^{(i-1)}$ از مرحله قبل در یک رجیستر $m+G$-1 بیتی ذخیره شود، علاوه بر این برای ذخیره حاصل عملیات *XOR* هنگام ضرب‌های جزئی[1] به یک رجیستر G بیتی نیاز می‌شود بنابراین:

$$\#FF = (m + G - 1) + G = m \times 2G - 1$$

در جدول (۱) منابع سخت‌افزاری مورد نیاز برای ضرب‌کننده LSD استفاده شده در این مقاله با کارهای ارائه شده در مراجع [۲۷] و [۲۸] مقایسه شده است. همان‌گونه که از این مقایسه پیداست استفاده از روش پیشنهادی در مقایسه با روش SAM[2] باعث صرفه جویی جزئی در منابع سخت‌افزاری می‌شود، اما در مقایسه با روش DAM[3] تعداد گیت‌های AND مورد نیاز کاهش قابل ملاحظه‌ای می‌یابد.

۴-۴. معماری پیشنهادی برای معکوس‌کننده میدان متناهی[4]

طراحی معکوس‌کننده طبق قضیه کوچک فرما با استفاده از ضرب و مربع میدانی قابل پیاده‌سازی است. بنابراین برای محاسبه معکوس برخلاف روش توسعه یافته اقلیدسی نیاز به داشتن یک واحد مجزای معکوس از بین می‌رود. بنابراین در پیاده‌سازی سخت‌افزاری با هدف صرفه‌جویی در منابع مصرفی استفاده از روش فرما در طراحی معکوس‌کننده مناسب بوده و منجر به کاهش پیچیدگی می‌شود، اگر چه در این روش ارجاع به واحدهای محاسباتی ضرب و مربع زیاد می‌شود، اما همان‌طور که در قسمت ۲-۲ ذکر شد می‌توان در رمزنگاری خم بیضوی با انتخاب مختصات تصویری مناسب استفاده از عملیات معکوس را به حداقل ممکن رساند. برای معکوس میدان در این مقاله طبق قضیه فرما از زنجیره جمع [۲۹] استفاده شده است که به‌روش معکوس اتو تسوجی معروف است. با این روش تعداد ضرب‌ها که در روش فرما *m-2* است به $\lfloor log_2(m-1) \rfloor + HW(m-1) - 1$ ضرب کاهش می‌یابد. $HW(.)$ وزن همینگ یا تعداد یک‌ها در نمایش باینری آرگومان آن است. در طراحی معکوس‌کننده طبق الگوریتم اتو تسوجی به‌دلیل این که انجام محاسبات به‌صورت زنجیروار صورت می‌گیرد، برای انجام ضرب در هر مرحله نیاز است که حاصل ضرب در مرحله قبل را داشته

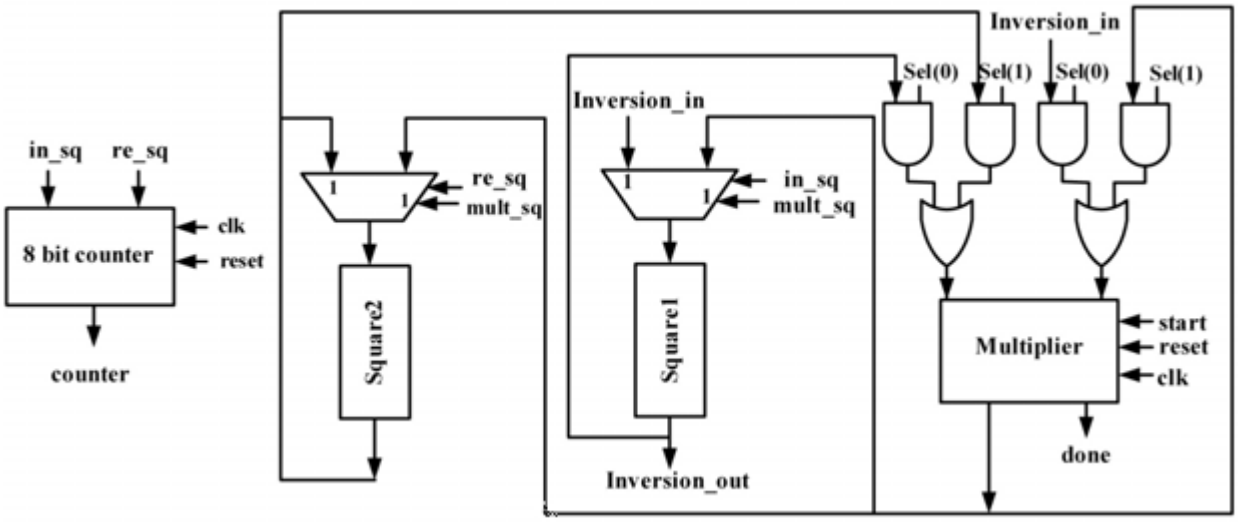
[1] Partial Products

[2] Single Accumulator Multiplier
[3] Double Accumulator Multiplier
[4] Finite Field Inversion

باشیم و چون امکان انجام موازی محاسبات وجود ندارد افزایش تعداد واحدهای محاسباتی ضرب در این مرحله مؤثر نخواهد بود [۲۹]. حال برای طراحی کارآمد یک واحد محاسباتی معکوس‌کننده براساس الگوریتم اتو تسوجی نیاز است به‌دنبال معماری باشیم که بتوانیم مسیر بحرانی را به حداقل ممکن برسانیم. به‌عبارتی دیگر مسیر بحرانی معکوس‌کننده منطبق بر مسیر بحرانی ضرب‌کننده باشد. اگر در طراحی معماری معکوس‌کننده تنها از یک واحد مربع‌کننده استفاده کنیم، این واحد باید عمل مربع را برای ورودی به معکوس‌کننده، خروجی ضرب‌کننده و خروجی همان واحد مربع‌کننده (تکرار عمل مربع) انجام دهد. بنابراین نیاز است که یک مالتی پلکسر سه به یک بر سر راه ورودی به مربع‌کننده استفاده کنیم. خروجی این واحد مربع نیز در ترکیب با تعداد قابل توجهی گیت‌های AND، OR و NOT به ورودی ضرب‌کننده ارجاع داده می‌شود که نتیجه این معماری قرار گرفتن مسیر بحرانی بر روی واحد مربع‌کننده و نه ضرب‌کننده خواهد بود. حال اگر به معماری معکوس‌کننده یک واحد مربع‌کننده دیگر اضافه کنیم، می‌توان از یک مالتی پلکسر دو به یک به جای مالتی پلکسر سه به یک استفاده کرد. در این معماری جدید، عمل مربع در مراحل ۱، ۳ و ۸ جدول (۱) و همچنین مربع نهایی در الگوریتم اتو- تسوجی توسط واحد مربع‌کننده اول انجام شده و سایر عملیات مربع در جدول (۱) با واحد مربع‌کننده دوم اجرا می‌شود. بدین ترتیب بر سر راه ورودی‌های ضرب‌کننده گیت‌ها با پیچیدگی کمتری قرار می‌گیرد، که این مطلب را به‌خوبی می‌توان در معماری ترسیم شده در شکل (۸) مشاهده کرد.

جدول ۱. مقایسه منابع مورد نیاز برای پیاده‌سازی SAM، DAM و روش پیشنهادی

روش	تعداد گیت‌های AND	تعداد گیت‌های XOR	تعداد فیلیپ فلاپ
SAM [۲۵]	(m + 4)G + 4(G - 1)	(m + 3)G + 4(G - 1)	2m + G – 1
DAM [۲۵]	(m + 4)G + 4(G - 1)	(m + 3)G + 4(G - 1) +(m - 1)	3m + G – 2
کار پیشنهادی	mG	mG+4(2G - 1)	2(m + G) - 1



شکل ۸. معماری معکوس‌کننده اتو تسوجی طراحی شده در این کار

۵. معماری پیشنهادی پردازنده ضرب عددی خم بیضوی

در این قسمت یک معماری جدید برای ضرب عددی نقطه خم ارائه شده است که در مقایسه با نتیجه‌های به‌دست آمده در سایر کارهای دیگر قابل توجه است. معماری مربوط به ضرب عددی در سطح بالای هرم طراحی رمزنگاری خم بیضوی به‌کار رفته است. پردازنده به‌گونه‌ای طراحی شده که با استفاده از تکنیک‌های موازی کردن و کاهش بیکاری واحدهای محاسبات، منجر به مسیر بحرانی کوتاه‌تر و تأخیر نهایی کمتر شود. برای طراحی پردازنده ضرب عددی خم بیضوی آن را به دو قسمت انجام محاسبات در مختصات تصویری و انجام محاسبات برای تبدیل مختصات تصویری به آفینی تقسیم می‌کنیم. برای انجام محاسبات در مختصات تصویری از قسمت ۱ و ۲ الگوریتم LD که در شکل (۲) نشان داده شد، استفاده می‌کنیم. در طراحی این قسمت از پردازنده، تعداد واحدهای محاسبات به‌گونه‌ای انتخاب شده که امکان اجرای موازی محاسبات میدانی در الگوریتم ضرب عددی LD وجود داشته باشد. به‌همین دلیل از سه واحد ضرب میدانی در پیاده‌سازی حلقه اصلی الگوریتم LD -که جمع و دو برابر کردن نقطه در آن انجام می‌شود- استفاده می‌کنیم. بنابراین با توجه به قسمت ۲.۱ الگوریتم شکل (۲) در مرحله اول سه ضرب X_1Z_2، X_2Z_1 و $T Z_2$ $(T \leftarrow X_2)$ به‌صورت موازی توسط سه ضرب‌کننده نشان داده در شکل (۹) انجام می‌شوند، سپس در مرحله دوم سه ضرب x_P Z_1، $X_1X_2T Z_2$ $(T \leftarrow Z_1)$ و bZ_2^4 به‌صورت موازی انجام می‌شوند. با

این کار هر بار تکرار حلقه در قسمت ۲ الگوریتم LD به‌جای اجرا با تأخیر شش ضرب میدانی با تأخیر دو ضرب میدانی انجام می‌شود. در این قسمت از پردازنده (انجام محاسبات در مختصات تصویری) از پنج واحد مربع‌کننده و دو واحد جمع‌کننده استفاده می‌شود که معماری آن در شکل (۹) مشاهده می‌شود. چهار واحد مربع‌کننده برای محاسبه X_2^4، X_1^4 یا Z_2^4، Z_1^4 استفاده می‌شود و یک واحد مربع‌کننده دیگر برای محاسبه $(X_1Z_2 + X_2Z_1)^2$ به‌کار می‌رود. همچنین پس از این که اولین ضرب میدانی انجام شد، نیاز است که حاصل $(X_1Z_2 + X_2Z_1)^2$ و $X_2^4 + bZ_2^4$ در رجیسترهای t_1 و t_2 ذخیره شوند تا برای انجام ضرب در مراحل بعد استفاده شوند. در طراحی این پردازنده از واحدهای محاسباتی ضرب، معکوس و مربع‌کننده استفاده می‌شود. چنانچه بخواهیم یک پردازنده با فرکانس کارکرد مناسب طراحی کنیم، می‌بایست مسیر بحرانی پردازنده بر روی بزرگ‌ترین مسیر در بین واحدهای محاسباتی قرار گیرد. از آنجا که واحد محاسباتی معکوس‌کننده در قبل به‌گونه‌ای طراحی شد که مسیر بحرانی آن بر مسیر بحرانی ضرب‌کننده منطبق شد و همچنین مسیر ضرب‌کننده از مربع‌کننده بزرگ‌تر است، در طراحی یک پردازنده کارآمد می‌بایست مسیر بحرانی آن بر روی مسیر بحرانی ضرب‌کننده قرار گیرد. بنابراین برای جلوگیری از افزایش طول مسیر بحرانی نیاز است که طراحی معماری در صورت امکان به‌صورت ترکیبی و هم‌زمان با استفاده از گیت‌های منطقی انجام شود، هر چند به‌کارگیری این گونه معماری‌ها ممکن است منجر به افزایش سطح پیاده‌سازی شود، اما دارای این مزیت است که فرکانس کارکرد پردازنده افزایش می‌یابد.

همچنین در طراحی معماری مربوط به محاسبات تصویری برای استفاده از مقادیر اولیه قسمت ۱ الگوریتم شکل (۲) محاسبات مجزایی انجام نشده است، زیرا اگر برای محاسبات قسمت ۱ الگوریتم، واحدهای محاسباتی اضافه‌ای علاوه بر آنچه که طراحی شده درنظر گرفته می‌شود، علاوه بر پیچیدگی مسیر بحرانی سطح پیاده‌سازی نیز افزایش خواهد یافت. بنابراین اجرای چنین محاسباتی که در طول مسیر محاسبات اصلی الگوریتم (قسمت ۲ الگوریتم شکل (۲)) قرار می‌گیرد، مناسب نیست و بایستی دنبال راه حلی باشیم. در همین راستا چنانچه بتوانیم از همان محاسبات انجام شده در قسمت ۲ الگوریتم برای به‌دست آوردن نتیجه‌های قسمت ۱ استفاده کنیم، می‌توان از انجام محاسبات اضافی جلوگیری کرد و به نتیجه‌های خوبی رسید. طرحی که در این جا ارائه شده ، بدین گونه است: زمانی که اولین بیت پر ارزش کلید ۱ شد، نیاز است که محاسبات قسمت ۱ انجام شود، بنابراین برای حالتی که k_i=۱ است اگر مقادیر زیر در انجام محاسبات قسمت ۲.۱ به‌کار گرفته شوند، آنگاه مقادیر اولیه مورد نیاز در الگوریتم LD مطابق با آنچه که در قسمت ۱ شکل (۲) آمده، به‌دست می‌آیند.

$$X_1 \leftarrow 1,\ Z_1 \leftarrow 0,\ X_2 \leftarrow x_P,\ Z_2 \leftarrow 1 \qquad (۱۲)$$

حاصل انجام محاسبات در قسمت ۲.۱ الگوریتم LD با استفاده از مقادیر رابطه (۱۲) به‌صورت زیر به‌دست می‌آید:

$$X_1 \leftarrow x_P,\ Z_1 \leftarrow 1,\ X_2 \leftarrow {x_P}^4 + b,\ Z_2 \leftarrow {x_P}^2 \qquad (۱۳)$$

معماری پیاده‌سازی محاسبه مقادیر اولیه در شکل (۹) مشاهده می‌شود. بدین ترتیب که مقادیر ۰، ۱ و x_P با استفاده از مالتی پلکسر به خروجی‌ها ارجاع داده می‌شوند تا در مرحله بعد با توجه به این که بیت کلید ۱ است، به ورودی متناسب در شکل (۹) وارد شوند و درنهایت مقادیر رابطه (۱۳) به‌دست می‌آیند.

پس از آنکه طراحی و پیاده‌سازی محاسبات مربوط به جمع و دو برابر کردن نقطه در مختصات تصویری انجام شد، بایستی درگاه‌های ورودی و خروجی معماری شکل (۹) براساس بیت‌های کلید در قالب یک معماری جدید به‌گونه‌ای به یکدیگر متصل شوند که از ورود مقادیر رابطه (۱۲) در محاسبات و همچنین تکرار الگوریتم LD در قسمت ۲ شکل (۲) آگاه شویم. بنابراین برای اینکه در این معماری جدید با توجه به اهداف ذکر شده، پیچیدگی پردازنده افزایش نیابد، لازم است طرحی را برای پیاده‌سازی این معماری جدید ارائه دهیم. در طرح ارائه شده در این قسمت همان‌گونه که در شکل (۱۰) مشخص است، مسیر ورودی و خروجی عملیات جمع و دو برابر کرده نقطه از یکدیگر مجزا شده‌اند تا هنگام جابه‌جا کردن مقادیر خروجی از پیچیده شدن معماری جلوگیری شود. علت به‌کارگیری این ایده آن است که برای انجام عمل جمع، خروجی‌های عملیات جمع و دو برابر کردن بدون وابستگی به مقدار بیت کلید به ورودی‌های عملیات جمع متصل می‌شود.

اگر در زیر به عملیات جمع که به ازای $k_i = 1$ است دقت کنیم، ورودی‌های این عملیات X_1، Z_1، X_2 و Z_2 هستند که خروجی این عملیات در X_1 و Z_1 ذخیره می‌شود.

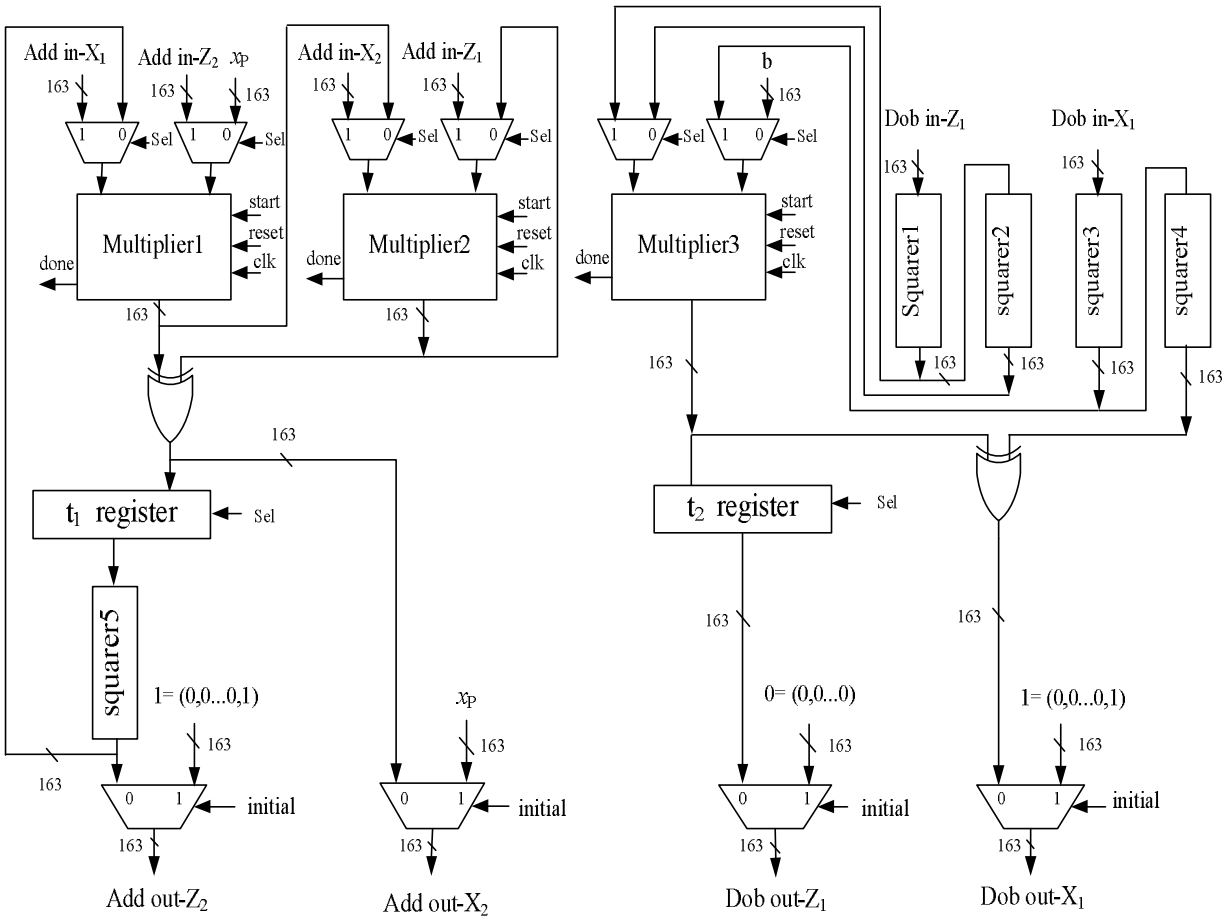
$$T \leftarrow Z_1,\ Z_1 \leftarrow (X_1Z_2 + X_2Z_1)^2,\ X_1 \leftarrow x_P\, Z_1 + X_1X_2T\, Z_2 \qquad (۱۴)$$

تغییر بیت کلید از ۱ به $k_i = 0$ با توجه به طراحی پردازنده منجر به تغییر در عبارت‌های $X_1Z_2 + X_2Z_1$ و $X_1X_2T\, Z_2$ در عبارت جمع می‌شود. از آنجایی که با تغییر در بیت‌های کلید X_1 با X_2 و Z_1 با Z_2 جابه‌جا می‌شود، در عبارت‌های $X_1Z_2 + X_2Z_1$ و $X_1X_2T\, Z_2$ تغییری ایجاد نمی‌شود، بنابراین می‌توان عملیات جمع را در حلقه تکرار الگوریتم بدون دخالت بیت‌های کلید انجام داد و تنها پس از پایان حلقه براساس مقدار بیت کلید جابه‌جایی را انجام داد.

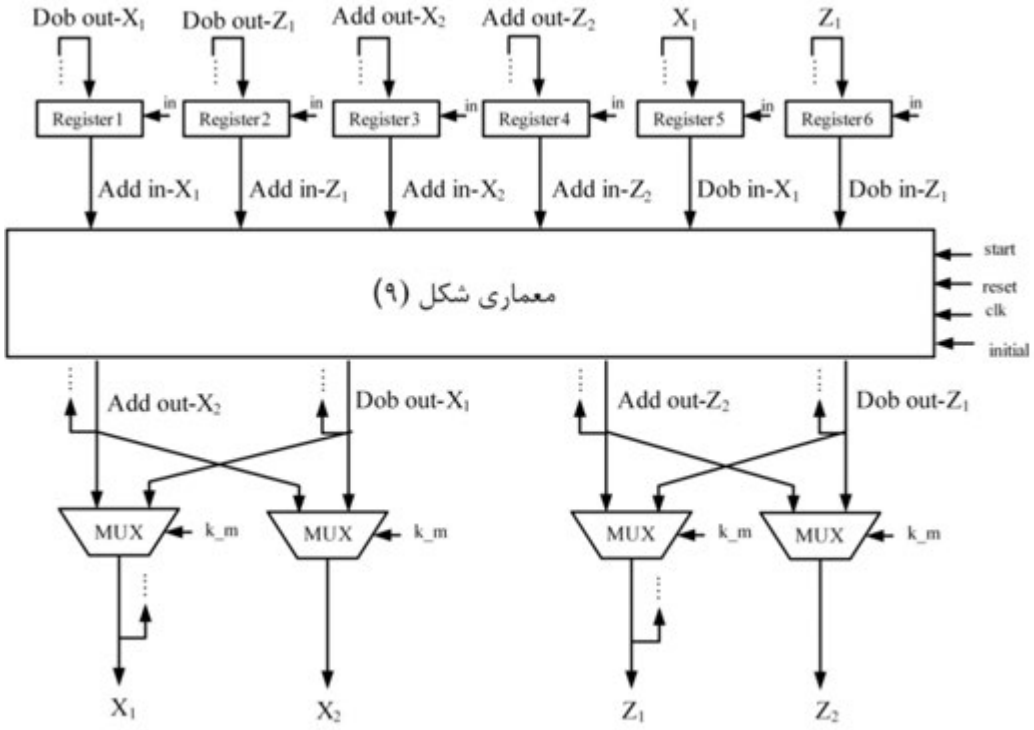
برای انجام عملیات دو برابر کردن به ازای $k_i = 1$ طبق الگوریتم عملیات زیر انجام می‌شود:

$$T \leftarrow X_2,\ X_2 \leftarrow X_2^4 + bZ_2^4,\ Z_2 \leftarrow T^2Z_2^2 \qquad (۱۵)$$

در عملیات دو برابر کردن با تغییر بیت کلید یا به‌عبارتی به ازای $k_i = 0$ جای X_2 و Z_2 با X_1 و Z_1 عوض می‌شود، بنابراین عملیات دو برابر کردن برخلاف جمع دو نقطه به کلید وابسته است، اما محاسبات مربوط به دو برابر کردن در مقایسه با جمع ساده‌تر است.



شکل ۹. معماری انجام محاسبات جمع و دو برابر کردن نقطه در مختصات تصویری الگوریتم LD



شکل ۱۰. معماری تکرار عملیات جمع و دو برابر کردن نقطه در الگوریتم LD براساس بیت‌های کلید

قسمت دوم پردازنده، محاسبات مربوط به تبدیل مختصات تصویری به آفینی را انجام می‌دهد. همان‌گونه که از قسمت سه و چهار الگوریتم شکل (۲) مشخص است، برای پیاده‌سازی آن نیاز به انجام روش محاسبات به‌نسبت زیاد می‌شود که این محاسبات اغلب به‌صورت ترتیبی و پشت سر هم انجام می‌شوند. به‌منظور این که مسیر بحرانی پردازنده همچنان بر روی ضرب‌کننده قرار داشته باشد، نیاز است که معماری این قسمت از پردازنده را نیز تا جایی که ممکن است به‌صورت ترکیبی و هم‌زمان طراحی کنیم. یکی از طرح‌ها که برای پیاده‌سازی تبدیل مختصات تصویری به آفینی در مرجع [۲] پیشنهاد شده، بر این اساس است که تعداد معکوس‌کننده مورد نیاز برای طراحی این قسمت از پردازنده به یک معکوس کاهش یابد، زیرا تنها با محاسبه معکوس $(x_P\ Z_1Z_2)^{-1}$ می‌توان حاصل معکوس دیگر یا به‌عبارتی X_1/Z_1 را با استفاده از ضرب $(x_P\ Z_2X_1)*(x_P\ Z_1Z_2)^{-1}$ به‌دست آورد. به بیانی دیگر در این طرح برای کاهش تعداد معکوس‌ها، تعداد ضرب‌های میدانی افزایش یافته است که با توجه به ترتیبی که در الگوریتم شکل (۱۱) لزوماً رعایت می‌شود (در صورتی که بخواهیم از تعداد واحدهای ضرب محدودی استفاده کنیم)، مراجعه به این واحدها افزایش می‌یابد که این خود منجر به افزایش طول مسیر بحرانی می‌شود. برای پیاده‌سازی این طرح طبق شکل (۱۱) نیاز است که ده ضرب میدانی انجام شود. همچنین برای انجام مراحل دوازدهم به بعد نیاز است که منتظر بمانیم تا حاصل معکوس $(x_P\ Z_1Z_2)^{-1}$ در مرحله دوازدهم به‌دست آید و از آنجایی که انجام مراحل بعد به‌طور کامل به این مرحله وابسته است، تأخیر غیر قابل اجتناب به‌نسبت زیادی در بین روال محاسبات ایجاد می‌شود. برای این طرح استفاده از چهار واحد ضرب‌کننده که به‌صورت موازی به‌کار گرفته شوند، می‌تواند در کاهش طول مسیر بحرانی مؤثر باشد، اما یکی از معایب در این طرح، تأخیر به‌نسبت بالای آن است.

یکی دیگر از روش‌هایی که می‌توان پیاده‌سازی کرد، براساس مرحله ۴ شکل (۲) است که در زیر مشاهده می‌شود.

$$y_3 \leftarrow (x_P + X_1/Z_1)[(X_1 + x_P Z_1)(X_2 + x_P Z_2) + (x_P{}^2 + y)(Z_1 Z_2)](x_P Z_1 Z_2)^{-1} + y_P \quad (۱۶)$$

برای انجام محاسبات در این طرح نیاز به دو واحد معکوس است و چنانچه از سه واحد ضرب استفاده کنیم، به‌جز این که در مرحله اول عملیات می‌توان سه ضرب‌کننده را به‌صورت موازی پیاده‌سازی کرد، در مراحل بعد بیکاری واحدهای ضرب افزایش می‌یابد (به‌علت وابستگی ضرب در مراحل فعلی به خروجی ضرب در مراحل قبل و همچنین خروجی معکوس‌کننده) که این عامل سبب ایجاد تأخیر می‌شود. یکی از پیشنهادهایی که به‌جهت پیاده‌سازی به‌صورت موازی بسیار کارآمد بوده و استفاده از آن منجر به کاهش تأخیر می‌شود، این است که در رابطه (۱۶) مقدار $(Z_1Z_2)^{-1}$ را در داخل کروشه وارد کنیم که حاصل آن به‌صورت عبارت زیر خواهد شد:

$$y_3 \leftarrow (x_P + X_1/Z_1)[(X_1/Z_1 + x_P)(X_2/Z_2 + x_P) + (x_P{}^2 + y)](x_P)^{-1} + y_P \quad (۱۷)$$

در همین راستا ابتدا Z_1^{-1}، Z_2^{-1} و x^{-1} را توسط سه واحد معکوس‌کننده به‌صورت هم‌زمان محاسبه می‌کنیم و سپس پنج ضرب مورد نیاز در رابطه (۱۷) را با دو واحد ضرب که در طی سه مرحله موازی شده، پیاده‌سازی می‌کنیم. برای این قسمت از محاسبات پنج واحد جمع‌کننده نیز نیاز است. مقدار مؤلفه x حاصل نهایی در مختصات آفینی مطابق قسمت ۳ الگوریتم LD در شکل (۲)، $x_3 = X_1/Z_1$ است که پس از محاسبه Z_1^{-1} توسط معکوس‌کننده می‌بایست آن را در X_1 ضرب کنیم. از آنجایی که در ضرب مرحله بعد از حاصل ضرب مرحله قبل یا به‌عبارتی از $X_1*Z_1^{-1}$ در محاسبه ضرب $(X_1/Z_1+x_P)*(X_2/Z_2+x_P)$ استفاده می‌شود، نیاز است که حاصل $X_1*Z_1^{-1}$ را در مرحله اول ذخیره کرده تا در مرحله بعد استفاده کنیم. ذخیره کردن این مقدار در یک رجیستر و استفاده از آن در پالس‌های ساعت بعد طول مسیر بحرانی را افزایش می‌دهد. برای جلوگیری از این رخداد بایستی رجیستر را حذف کنیم. از آنجایی که در قسمت تبدیل مختصات، پیاده‌سازی ضرب‌کننده‌ها به‌صورت موازی انجام شده است، در انجام ضرب مرحله دوم حاصل ضرب مرحله اول از بین می‌رود، اما با توجه به این که در مرحله سوم یکی از ضرب‌کننده‌ها بیکار است (زیرا پنج ضرب طی سه مرحله توسط دو واحد ضرب‌کننده انجام می‌شود) می‌توان ضرب مرحله اول را دوباره تکرار کرد تا پاسخ از بین رفته دوباره احیا شود. بنابراین ضرب $X_1*Z_1^{-1}$ در مرحله سوم تکرار می‌شود تا پاسخ $x_3 = X_1/Z_1$ به‌دست آید تا نیازی به استفاده از رجیستر در این قسمت از پردازنده نباشد و طراحی این قسمت به‌طور کامل ترکیبی و هم‌زمان انجام شود. درنهایت یکی از اقدامات مهمی که در طراحی پردازنده بایستی مدنظر قرار داد، انتخاب طول کلمات (مقدار G) در ضرب‌کننده‌های میدانی است. انتخاب مناسب G نقش به‌سزایی در کاهش تأخیر اجرای عملیات ضرب عددی خم و همچنین کاهش سطح پیاده‌سازی خواهد داشت. یکی از ملاحظاتی که در انتخاب مقدار G برای واحد محاسباتی ضرب باید رعایت شود، این است که از بین G هایی که مقدار $\lceil m/G \rceil$ آنها یکسان است، کوچک‌ترین مقدار را انتخاب می‌کنیم زیرا با وجود سرعت یکسان برای آنها، G کوچک‌تر منابع کمتری مصرف می‌کند. به‌طور مثال برای $m=$ ۱۶۳ از بین G های مجموعه {۵۴-۴۱} مقدار $G=$ ۴۱ را طبق شرایط ذکر شده انتخاب می‌کنیم. برای انجام محاسبات در مختصات تصویری با توجه به تکرار آن (قسمت ۲ الگوریتم شکل (۲)) اجرای سریع محاسبات برای طراحی یک پردازنده کارآمد دارای اهمیت است. بنابراین انتخاب طول کلمه بزرگ (G) برای ضرب‌کننده‌های استفاده شده در محاسبات مختصات تصویری مناسب است. مقادیری که در این قسمت استفاده شده، طول کلمات مختلف است که با G_1 در جدول نتایج اجرای پردازنده نشان می‌دهیم. در انجام محاسبات تبدیل مختصات تصویری به آفینی (قسمت سه و چهار الگوریتم شکل (۲)) به این دلیل که محاسبات تنها یک بار در انتهای الگوریتم مورد استفاده قرار می‌گیرد و همانند قسمت ۲ الگوریتم تکرار نمی‌شود، نیاز مبرمی به بهبود کاهش تأخیر در آن نیست. در این قسمت از پردازنده واحدهای محاسباتی زیادی استفاده شده است وچنانچه از

کلمه با طول کوچک استفاده کنیم، سطح پیاده‌سازی به‌طور قابل ملاحظه‌ای کاهش می‌یابد. کمترین طول کلمه مناسب که مسیر بحرانی پردازنده را تغییر ندهد، برابر G_2 =۱۱ است که در جدول (۴) مشاهده می‌شود. همان‌گونه که توضیح داده شد، با انتخاب G مناسب برای ضرب میدانی مورد استفاده در قسمت اول و دوم پردازنده، می‌توان با افزایش سرعت، کاهش تأخیر پردازنده و کاهش سطح پیاده‌سازی کارآیی پردازنده را افزایش داد.

۶. نتایج و بحث

در انجام محاسبات پارامترهای مورد نیاز از قبیل ضریب‌های خم، مرتبه خم و مختصات نقطه مبنا به‌صورت زیر براساس پیشنهاد NIST در مرجع [۴] انتخاب شده است. برای پیاده‌سازی ساختار، پیاده‌سازی ضرب عددی از کدهای قابل سنتز VHDL استفاده شده است. کدها در محیط ISE11.1 سنتز شده و تراشه Virtex-4 FPGA (XC4VLX200) به‌عنوان بستر پیاده‌سازی انتخاب شده است. مساحت کل این تراشه 89088 اسلایس است. همانگونه که در قسمت قبل ذکر شد، برای انجام محاسبات در مختصات تصویری از ضرب‌کننده میدان سریال- موازی با طول کلمه G_1 و برای انجام محاسبات تبدیل مختصات تصویری به آفینی از ضرب‌کننده با طول کلمه G_2 استفاده شده است. نتیجه‌های حاصل از پیاده‌سازی ضرب عددی برای دو طول کلمه مختلف در جدول (۲) آمده و با سایر کارها در جدول (۳) مقایسه شده است. نکته قابل توجه آنکه در جدول (۳) با وجود آنکه کارایی پیاده‌سازی گزارش شده در مرجع [۱۸] بالاتر از این کار است، اما در عوض سرعت پردازنده پیاده‌سازی انجام شده در این مقاله چهار بار بیشتر از پردازنده طراحی شده در مرجع [۱۸] است.

جدول ۲. نتیجه‌های پیاده‌سازی پردازنده ضرب عددی خم بیضوی پیشنهادی در این مقاله

G_1	G_2	Freq. (MHz)	period (ns)	No. of cycles	Time (µs)	slices	Area LUT	FF	Occupied chip area	Efficiency
۴۱	۱۱	۲۵۱/۰۵۴	۳/۹۸	۲۹۹۳	۱۱/۹۲	۱۹۶۰۴	۳۶۷۲۷	۶۹۹۴	%۲۲	۳۷۲/۱
۲۴	۱۱	۲۵۴/۴۴	۳/۹۳	۳۹۷۱	۱۵/۶	۱۵۹۵۱	۲۹۸۶۵	۶۹۷۴	%۱۷	۳۴۹

جدول ۳. نتایج پیاده سازی ضرب عددی خم بیضوی در کارهای قبلی

Ref.	M	FPGA	Freq. (MHz)	Time (µs)	Area (slices)	Area (LUT)	Area (FF)	Efficiency
Orlando and Paar [11]	۱۶۷	XCV400E	۷۶/۷	۲۱۰	–	۳۰۰۲	۱۷۶۹	۲۶۵
N. Gura [12]	۱۶۳	XCV2000E	۶۶/۴	۱۴۴	–	۲۰۰۶۸	۶۳۲۱	۵۶
Jarvinen et. al. [13]	۱۶۳	VinexII V8000	۹۰/۲	۱۰۶	۱۸۰۷۹	-	-	۴۴
Rodriguez et. al. [14]	۱۶۳	XCV2600E	۴۶/۵	۶۳	۲۴ + ۱۸۳۱۴ RAMs	-	-	-
Lutz and Hasan [15]	۱۶۳	Virtex 2000E	۶۶	۲۳۳	-	۱۰۰۱۷	۱۹۳۰	۷۰
Sakiyama [16]	۱۶۳	Virtex II pro 30	۱۰۰	۲۸۰	۸۴۵۰	-	-	-
Chelton and Benaissa [17]	۱۶۳	Virtex-4 VLX200	۱۵۳/۹	۱۹/۵۵	۱۶۲۰۹	۲۶۳۶۴	-	۳۱۶
Ansar and Hasan [18]	۱۶۳	XC2V2000	۱۰۰	۴۶/۵	۳۴۱۶	۷۵۵۹	-	۵۳۲
Yong et.al. [19]	۱۶۳	XC2V6000	۹۳/۳	۳۴/۱۱	۱۳۳۷۶	۲۸۱۲	-	۳۴۰
Kim et. al. [20]	۱۶۳	XC4VLX80	۱۴۳	۱۰	۳۶۳,۲۳	-	-	-

۷. نتیجه‌گیری

در این مقاله یک پردازنده کارآمد رمزنگاری خم بیضوی را طراحی کردیم. یکی از عوامل مهم در کارایی پردازنده، طراحی و پیاده‌سازی واحدهای محاسبات میدانی کارآمد است. برای انجام ضرب میدانی از الگوریتم ضرب LSD استفاده کردیم تا بتوانیم با انتخاب طول کلمات مختلف، بهینه‌سازی مشترکی در سرعت و سطح پیاده‌سازی پردازنده انجام دهیم. همچنین به‌کارگیری گراف درخت در XOR کردن دو رشته بیت منجر به کاهش مسیر بحرانی واحد محاسباتی ضرب‌کننده شد که این طراحی در کاهش طول مسیر بحرانی معکوس‌کننده و پردازنده تأثیر مستقیم داشت. در طراحی معکوس‌کننده میدان اتو- تسوجی با استفاده از حداکثر موازی‌سازی طول مسیر بحرانی را کاهش دادیم. در طراحی معماری پردازنده، در قسمت انجام محاسبات تصویری با استفاده از سه واحد محاسباتی ضرب موازی شده و کاهش سیکل‌های بیکار در هر مرحله، تأخیر پردازنده را که به‌طور عمده مربوط به محاسبات در مختصات تصویری است، کم کردیم. جدا کردن مسیر جمع کردن نقطه از دو برابر کردن و همچنین استفاده از مقادیر اولیه مناسب در راه‌اندازی اولیه پردازنده،

پیچیدگی پردازنده را کاهش داد. در طراحی قسمت دوم پردازنده که تبدیل مختصات تصویری به آفینی است، با طراحی و پیاده‌سازی معماری مناسب اثر ترتیبی بودن انجام محاسبات را کاهش دادیم. همچنین در طراحی قسمت تبدیل مختصات با توجه به این که به‌دفعات از واحد محاسباتی ضرب استفاده می‌شود و این قسمت از پردازنده تأثیر اندکی در تأخیر نهایی دارد، از کمترین طول کلمه ممکن را استفاده کردیم تا در مصرف منابع سخت‌افزاری صرفه‌جویی شود. نتایج تحقق طراحی پیشنهادی در این مقاله، نشان داد که این کار به‌مراتب از نتیجه‌های گزارش شده در ادبیات مربوطه کارآمدتر است. ضمن آنکه با افزودن نوعی انعطاف‌پذیری به ساختار پردازنده، آن را برای کاربردهای سرعت بالا و نیز کاربردهایی که به‌نوعی مصالحه زمان- سطح اشغالی نیاز دارند، مناسب کردیم. گرچه این طراحی برای پیاده‌سازی بر روی FPGA پیشنهاد شد، اما منحصر به آرایه‌های منطقی آرایش‌پذیر نیست و می‌توان آن را بر روی سایر بسترها از جمله ASIC نیز پیاده‌سازی کرد.

۸. مراجع

[1] Hankerson, D.; Menezes, A.; Vanstone, S. “Guide to Elliptic Curve Cryptography”; Springer-Verlag, 2004.

[2] Rodriguez-Henriquez, F.; Saqib, N. A.; Díaz Pérez, A.; Koc, C. K. “Cryptographic Algorithms on Reconfigurable Hardware”; Springer, 2006.

[3] Wollinger, T.; Guajardo, J.; Paar, C. “Security on FPGAs: State-of-the-Art and Implementations Attacks”; ACM Trans. on Embedded Computing Sys. 2004, 3, 534- 574.

[4] NIST—National Institute of Standards and Technology, “Recommended Elliptic Curves for Federal Government Use”; http://csrc.nist.gov/encryption, 2000.

[5] Rosing, M. “Implementing Elliptic Curve Cryptography”; Manning Publications Co. 1999.

[6] Blake, I.; Seroussi. G.; Smart, N. “Elliptic Curves in Cryptography”, London Mathematical Society Lecture Note Series 265, Cambridge Univ. Press, 2002.

[7] ANSI, ANSI X9.62 “The Elliptic Curve Digital Signature Algorithm (ECDSA)”; http://www.ansi.org.

[8] Standard Specifications for Public Key Cryptography, IEEE 1363, 2000.

[9] Lopez, J.; Dahab, R. “Fast Multiplication on Elliptic Curves over GF (2^m) without Precomputation”; Workshop on Cryptographic Hardware Embedded Sys. (CHES), 1999.

[10] Montgomery, P. L. “Speeding the Pollard and Elliptic Curve Methods of Factorization”; J. Math. Comp. 1987, 48, 243-264.

[11] Orlando, G.; Paar, C. “A High-Performance Reconfigurable Elliptic Curve Processor for GF (2^m)”; Cryptographic Hardware and Embedded Sys. (CHES), 2000.

[12] Gura, N.; Shantz, S. C.; Eberle, H.; Gupta, S.; Gupta, N. “An End-to-End Systems Approach to Elliptic Curve Cryptography”; Workshop Cryptographic Hardware and Embedded Sys. (CHES), 2002.

[13] Jarvinen, K.; Tommiska, M.; Skytta, J. “A Scalable Architecture for Elliptic Curve Point Multiplication”; ICFPT, 2004.

[14] Rodriguez-Henriquez, G.; Saqib, N. A.; Diaz-Perez, A. “A Fast Parallel Implementation of Elliptic Curve Point Multiplication Over GF (2^m)”; Microprocessors Microsys. 2004, 28, 329–339.

[15] Lutz, J.; Hasan, A. “High Performance FPGA Based Elliptic Curve Cryptographic Co-Processor”; in Proc. Int. Conf. Inf. Technol.: Coding Comput. (ITCC), 2004, 486.

[16] Sakiyama, K.; Batina, L.; Preneel, B.; Verbauwhede, I. “Superscalar Coprocessor for High-Speed Curve-Based Cryptography”; Workshop on Cryptographic Hardware and Embedded Sys. (CHES), 2006, Lecture Notes in Computer Sci. 4249, 415-429.

[17] Chelton, W.; Benaissa, M. “Fast Elliptic Curve Cryptography on FPGA”; IEEE Trans. on Very Large Scale Integration (VLSI) Sys. 2008, 16, 198-205.

[18] Ansari, B.; Hasan, A. “High-Performance Architecture of Elliptic Curve Scalar Multiplication”; IEEE Trans. on Comp. 2008, 57, 1443-1453.

[19] Yong-Ping, D.; Uecheng, O.; Zheng-lin, L.; Yu, H.; Li-hua, Y. “High-Performance Hardware Architecture of Elliptic Curve Cryptography Processor over GF (2^{163})”; J. Zhejiang Univ. Sci. A 2009, 10, 301-310.

[20] Kim, C. H.; Kwon, S.; Hong, C. P. “FPGA Implementation of High Performance Elliptic Curve Cryptographic Processor over GF (2^{163})”; J. Sys. Architecture 2008, 54, 893–900.

[21] Deschamps, J. P. “Hardware Implementation of Finite-Field Arithmetic”; McGraw Hill, 2009.

[22] Scott, P. A.; Tavares, S. E.; Peppard, L. E. “A Fast VLSI Multiplier for GF (2^m)”; IEEE J. Sel. Areas Commun. 1986, 4, 62–66.

[23] Mastrovito, E. “VLSI Architectures for Computations in Galois Fields”; Dept. Elect. Eng., Linkoping Univ., Linkoping, Sweden 1991, 249.

[24] Rodriguez-Henriguez, F.; Koc, C. K. “Parallel Multipliers Based on Special Irreducible Pentanomials”; IEEE Trans. Comput. 2003, 52, 1535–1542.

[25] Wu, H. “Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial Basis”; IEEE Trans. Computer 2002, 51, 750–758.

[26] Reyhani-Masoleh, A.; Hasan, M. A. “Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication Over GF (2^m)”; IEEE Trans. Computer 2004, 53, 945–959.

[27] Kummar, S.; Wollinger, T.; Paar, C. “Optimum Digit Serial GF (2^m) Multipliers for Curve Based Cryptography”; IEEE Trans. Computer 2006, 55, 1306–1311.

[28] Song, L.; Parhi, K. K. “Low-Energy Digit-Serial/Parallel Finite Field Multipliers”; J. VLSI Signal Process. Sys. 1998, 19, 149–166.

[29] Itoh, T.; Tsujii, S. "A Fast Algorithm for Computing Multiplicative Inverses in GF (2^m) using Normal Bases”; Information and Computation 1998, 78, 171-177.