هفتمین کنفرانس بین المللی انجمن رمز ایران

۲۴ و ۲۵ شهریورماه ۱۳۸۹، دانشگاه صنعتی خواجه نصیرالدین طوسی

انجمن رمزایران

دانشگاه صنعتی خواجه نصیرالدین طوسی

!

طراحی و پیاده سازی یک پردازنده کارآمد ضرب اسکالر خم بیضوی

در میدان $GF(2^{163})$

حسین مهدیزاده[۱]، مسعود معصومی[۲]، محمود احمدیان[۲]

تهران، دانشگاه شهید ستاری، مرکز تحصیلات تکمیلی [۱]

h.mahdizade@yahoo.com

تهران، دانشگاه صنعتی خواجه نصیرالدین طوسی، دانشکده مهندسی برق و کامپیوتر [۲]

m_masoumi@eetd.kntu.ir, m_ahmadian@kntu.ac.ir

چکیده

در این مقاله یک پردازنده ضرب اسکالر خم بیضوی کارآمد در میدان باینری $GF(2^{163})$ طراحی و با استفاده از کدهای قابل سنتز VHDL پیاده‌سازی شده است. طراحی واحد های محاسبات میدانی موثر در پردازنده و به خصوص واحد محاسباتی ضرب میدانی با بکارگیری ایده درخت باینری برای XOR کردن، منجر به کاهش طول مسیر بحرانی گردید. همچنین استفاده از اجرای موازی عملیات ضرب میدانی و نیز حداکثر به اشتراک گذاری منابع در پیاده سازی باعث بهبود عملکرد پرازنده در مقایسه با بهترین پیاده‌سازی‌های گزارش شده تاکنون در ادبیات مربوطه شد. زمان اجرای ضرب اسکالر با طول رقم ۵۵ در پردازنده جدید 18.362 µs و مساحت اشغالی آن بر روی تراشه Xilinx - XC4VLX200 برابر 13198 slices می‌شود که آن را برای پیاده سازی بر روی تراشه FPGA کاملا مناسب می‌سازد.

واژه های کلیدی

رمزنگاری خم بیضوی، محاسبات میدانی، ضرب اسکالر، پیاده‌سازی سخت افزاری، FPGA.

۱- مقدمه !

همزمان با پیشرفت و فراگیری روزافزون مخابرات دیجیتال و افزایش حجم مبادله اطلاعات از طریق شبکه های متنوع مخابره داده ها نیاز به استانداردها و الگوریتم های مخابراتی با کارآیی بالا از جمله ضروریات دنیای امروز محسوب می شود. یکی از وظایف مهم صاحب نظران امروز دنیای ارتباطات طراحی الگوریتم هایی با قابلیت و انعطاف پذیری بالاست که بتواند پاسخگوی تقاضا برای شبکه های مخابراتی پر سرعت، پر ظرفیت، کم حجم، کم هزینه و در عین حال امن باشد. رمزنگاری کلید عمومی مبتنی بر خم بیضوی[۱] در سال ۱۹۸۵ میلادی توسط نیل کوبلیتز و ویل میلر[۲] به طور جداگانه مطرح گردید. امنیت این مکانیزم رمزنگاری بر دشواری مسئله لگاریتم گسسته بر روی خم‌های بیضوی یا ECDLP [۳] استوار می‌باشد 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

[1] Elliptic curve cryptography

[2] Neal Koblitz and Victor Miller

[3] Elliptic curve discrete logarithm problem

روش RSA برسد. به طور مثال کلید ۱۶۳ بیتی رمزنگاری خم بیضوی سطح امنیتی یکسانی با کلید ۱۰۲۴ بیتی RSA دارد. بدیهی است با کاهش طول کلید، طراحان سیستم های رمزنگاری کلید عمومی قادرند سیستم هایی پر سرعت تر، کم حجم تر و دارای توان پردازشی بالاتر با صرفه جویی در منابع سخت افزاری طراحی نمایند.

قبل از این که به پیاده سازی یک سیستم رمزنگاری مبتنی بر خم بیضوی بپردازیم نیاز است که متناسب با اهدا ف مورد نظر برای سیستم انتخاب هایی صورت بگیرد. پارامترهایی که مربوط به حوزه خم بیضوی است از جمله میدان متناهی که ضرایب و اعضای میدان تحت آن قرار می گیرند، نمایش اعضای میدان، انتخاب خم و الگوریتم های مربوط به محاسبات میدانی شامل ضرب، مربع و معکوس و محاسبات مربوط به خم بیضوی از قبیل جمع دو نقطه، دو برابر کردن نقطه و الگوریتم های ضرب اسکالر نمونه هایی از این انتخاب ها می باشند. در این بین رعایت ملاحظات امنیتی، بستر پیاده سازی (سخت افزار و نرم افزار) مزایا و محدودیت موجود در هر بستر و محدودیت های محیط ارتباطی(از قبیل پهنای باند) در انتخاب های مذکور موثر می‌باشند[۲].

در میان بسترهای پیاده سازی، FPGA ها به خاطر دارا بودن مزایایی از قبیل زمان و هزینه طراحی و پیاده سازی کم، انعطاف پذیری، قابلیت پیکربندی مجدد و کارایی بالا از جمله بسترهای پرکاربرد هستند که در طراحی و پیاده سازی الگوریتم های رمزنگاری مورد توجه می باشند [۱].

در این مقاله چون هدف پیاده سازی افزایش سرعت و کاهش مساحت اشغالی بر روی بستر FPGA می باشد، از نمایش چند جمله ای در میدان باینری به علت امکان پیاده سازی موثر استفاده می شود. همچنین به علت استفاده از الگوریتم ضرب اسکالر مونتگمری، خم بیضوی نا ابر منفرد[۴] در میدان باینری طبق پیشنهاد NIST [۵] انتخاب شده است که کوچکترین میدان باینری پیشنهادی $GF(2^{163})$ است [۳].

در این کار سعی نموده ایم با حداکثر موازی کردن انجام عملیات میدانی با توجه به تعداد واحد های محاسبات میدانی موجود به خصوص در انجام عمل ضرب میدانی و همچنین اشتراک منابع در استفاده از این واحدها، به یک پردازنده سریع و دارای سطح پیاده سازی نسبتا کوچک برسیم به طوری که با تغییر طول ارقام در ضرب کننده می توان به زمان اجرا و سطح پیاده سازی گوناگونی رسید.

در ادامه مقاله، در قسمت ۲ یک پیش زمینه ریاضی از محاسبات مربوط به خم های بیضوی و میدان های متناهی را بیان نموده و سپس در بخش ۳ معماری پیشنهادی برای طراحی پردازنده ضرب اسکالر خم بر روی FPGA را تشریح می کنیم. در انتها نتایج سنتز به دست آمده را ارائه نموده و با سایر کارهای مشابه قبلی مقایسه می نماییم.

۲- پیش زمینه ریاضی

۲-۱- خم های بیضوی و محاسبات بر روی آنها

سیستم های رمزنگاری مبتنی بر خم بیضوی را می توان بر روی میدان های $GF(p)$ و $GF(p^m)$ (p عدد اول است) پیاده سازی نمود. از آن جایی که طراحی پردازنده ها توسط میدان های باینری سریع تر و کوچکتر می باشد، بنابراین میدان متناهی $GF(2^m)$ را با نمایش چند جمله ای برای پیاده سازی به کار می بریم.

همان طور که در قسمت قبل ذکر کردیم از بین انواع خم ها، خم بیضوی نا ابر منفرد باینری را بر اساس پیشنهاد NIST انتخاب می کنیم. خم های نا ابر منفرد نسبت به خم های ابر منفرد[۶] ترجیح داده می شوند زیرا محاسبات بر روی این خم ها سریع تر و امن تر می باشند[۴،۵].

برای اطمینان از امنیت رمزنگاری مبتنی بر خم بیضوی در میدان های $GF(2^m)$ حداقل مرتبه میدان ۱۶۰ بیت پیشنهاد شده است [۶،۷].

فرض کنید که $F_q = GF(2^m)$ یک میدان متناهی با مشخصه دو باشد. یک خم بیضوی نا ابر منفرد $E(F_q)$ با معادله زیر تعریف می شود،

$$y^2 + xy \equiv x^3 + ax^2 + b \pmod{f(x)}, \qquad (۱)$$

به طوری که $a, b \in F_q$ ، $b \neq 0$ و $f(x)$ چند جمله ای کاهشی تحویل ناپذیر است.

مجموعه نقاط $(x, y) \in GF(2^m) \times GF(2^m)$ که در معادله بالا صدق می کنند به اضافه نقطه در بینهایت (عنصر خنثی) که به صورت $\mathcal{O}$ نشان می دهند، یک گروه آبلی جمعی را درست می کنند.

فرض کنید که $P = (x,y)$ نقطه ای باشد که در معادله بالا صدق می کند، قرینه نقطه P در میدان های باینری به صورت $-P = (x,x+y)$ قابل حصول می باشد، همچنین نقطه در بینهایت به عنوان عضو خنثی گروه جمعی نقاط خم به صورت زیر عمل می کند.

$$P + \mathcal{O} = P, \quad P + (-P) = \mathcal{O} \qquad (۲)$$

[4] Non-supersingular

[5] National Institute of Standards and Technology

[6] Supersingular

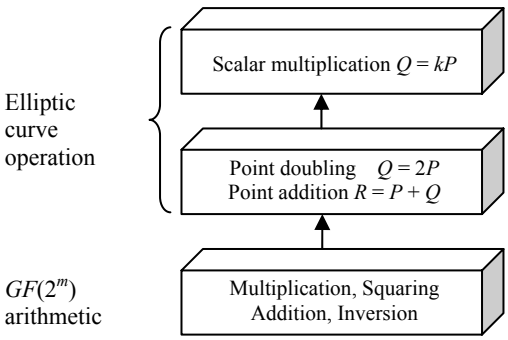
جمع P + Q از نقاط $P = (x_1,y_1)$ و $Q = (x_2,y_2)$ (با فرض این که $P, Q \neq \mathcal{O}$) نقطه $R = (x_3,y_3)$ است به طوری که :

$$\begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 = \lambda(x_1 + x_3) + y_1 + x_3 \\ \lambda = (y_2 + y_1)/(x_2 + x_1) \end{cases} \qquad (۳)$$

دو برابر نقطه $P = (x_1,y_1)$ یعنی نقطه $R = 2P = (x_3,y_3)$ به صورت زیر قابل محاسبه است :

$$\begin{cases} x_3 = \lambda^2 + \lambda + a \\ y_3 = \lambda(x_1 + x_3) + y_1 + x_3 \\ \lambda = x_1 + y_1/x_1 \end{cases} \qquad (۴)$$

اکنون پس از بیان نحوه عملیات جمع و دو برابر کردن نقاط خم، ضرب اسکالر در نقاط خم را بیان می نماییم. برای این که یک عملیات ضرب اسکالر انجام شود نیاز است که مدل سه لایه ای شکل ۱ طراحی شود. پایین ترین لایه یعنی محاسبات مربوط به میدان متناهی را در قسمت بعد مورد بررسی قرار می دهیم. حال به انتخاب یک الگوریتم ضرب اسکالر متناسب با اهداف مورد نظر در پیاده سازیمان می پردازیم.



شکل ۱- مدل سه لایه ای ضرب اسکالر نقطه خم بیضوی.

برای ارزیابی الگوریتم های ضرب اسکالر از نظر سرعت اجرا و حافظه اشغالی، تعداد عملیات ضرب میدانی و تعداد نقاط ذخیره شده برای پیش محاسبات به طور معمول مد نظر قرار می گیرند. برای بررسی جزیی تر هزینه مورد نیاز برای الگوریتم های ضرب اسکالر به [۲] مراجعه شود.

در این مقاله الگوریتم ضرب اسکالر LD [۷] در مختصات تصویری[۸] [۸] که دارای عملکرد بالایی می باشد استفاده شده است، این روش یک پیاده سازی موثر از الگوریتم ضرب اسکالر مونتگمری [۹] می باشد. علت استفاده از مختصات تصویری عدم نیاز به معکوس در محاسبات اصلی ضرب اسکالر می باشد، چرا که اجرای عملیات معکوس زمان بر می باشد. بنابراین استفاده از مختصات تصویری از نظر زمان اجرای الگوریتم مقرون به صرفه است. برای آشنایی بیشتر با انواع مختصات تصویری و نحوه نمایش نقاط به [۲] مراجعه شود.

در شکل ۲ الگوریتم ضرب اسکالر LD مشاهده می شود. این الگوریتم را به سه مرحله می توان تقسیم نمود؛ محاسبات ابتدای الگوریتم که تبدیل مختصات نقطه دریافتی به عنوان ورودی از آفینی به تصویری است، محاسبات حلقه اصلی که بر اساس بیت های کلید عملیات جمع و دو برابر کردن نقطه انجام می شود و محاسبات انتهای الگوریتم که شامل تبدیل مختصات تصویری به آفینی برای تعیین حاصل kP در مختصات آفینی می باشد. استفاده از الگوریتم LD دارای فوایدی می باشد از جمله این که چون نیاز به ذخیره نقاط به عنوان پیش محاسبه ندارد به رجیسترهای کمتری در مقایسه با سایر الگوریتم ها احتیاج دارد. یکی از مهمترین خواص این الگوریتم امکان اجرای موازی عملیات میدانی در حلقه اصلی هنگام اجرای عملیات جمع و دو برابر کردن نقاط می باشد به طوری که با حداکثر اجرای موازی می توان عملیات جمع و دو برابر کردن نقطه را به ازای هر بیت از کلید به صورت موازی پیاده نمود که این خصوصیت در تشریح معماری الگوریتم در قسمت های بعدی بیشتر مورد بررسی قرار می گیرد. یکی دیگر از فواید این الگوریتم با توجه به انجام عملیات مشابه به ازای هر بیت از کلید، مقاوم بودن آن در برابر حملات کانال جانبی زمانی و توان ساده می باشد.

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$ with $k_{t-1} = 1$, $P = (x, y) \in E(F_2^m)$.
OUTPUT: kP.
1. $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$. {Compute $(P,2P)$}
2. For i from $t-2$ downto 0 do
  2.1 If $k_i = 1$ then
    $T \leftarrow Z_1$, $Z_1 \leftarrow (X_1Z_2 + X_2Z_1)^2$, $X_1 \leftarrow xZ_1 + X_1X_2TZ_2$.
    $T \leftarrow X_2$, $X_2 \leftarrow X_2^4 + bZ_2^4$, $Z_2 \leftarrow T^2Z_2^2$.
  2.2 Else
    $T \leftarrow Z_2$, $Z_2 \leftarrow (X_1Z_2 + X_2Z_1)^2$, $X_2 \leftarrow xZ_2 + X_1X_2Z_1T$.
    $T \leftarrow X_1$, $X_1 \leftarrow X_1^4 + bZ_1^4$, $Z1 \leftarrow T^2Z_1^2$.
3. $x_3 \leftarrow X_1/Z_1$.
4. $y_3 \leftarrow (x+X_1/Z_1)[(X_1+xZ_1)(X_2+xZ_2)+(x^2+y)(Z_1Z_2)](xZ_1Z_2)^{-1} + y$.
5. Return(x_3, y_3)

شکل ۲- الگوریتم ضرب اسکالر Lopez و Dahab [۸].

تعداد محاسبات میدانی الگوریتم ضرب اسکالر LD را می توان به صورت جدول ۱ ارائه نمود.

[7] Lopez and Dahab

[8] Projective coordinates

جدول ۱- هزینه اجرای الگوریتم ضرب اسکالر LD

Complexity	Affine to Proj	Proj. Scalar Mul	Proj. to Affine
Square	2	$5(m-1)$	1
Addition	1	$3(m-1)$	5
Multiplication	0	$6(m-1)$	3
Division/ Inversion	0	0	2

۲-۲- محاسبات در میدان متناهی $GF(2^m)$

جمع و تفاضل دو عنصر متعلق به میدان باینری ساده ترین عملیات ریاضی میدان می باشد زیرا بیت های هر عنصر با بیت های مشابه (از نظر ارزش مکانی) عنصر دیگر XOR می شوند و نیازی به کاهش در میدان نمی باشد.

انجام عمل ضرب میدانی مهمترین عملیات در پردازنده ضرب اسکالر می باشد به طوری که اغلب در تعیین فرکانس کاری مدار و میزان سطح اشغالی ابزار مورد استفاده موثر می باشد بنابراین استفاده از یک ضرب کننده کارآمد دارای اهمیت می باشد. ضرب کننده های میدانی مورد استفاده در پیاده سازی الگوریتم های ضرب اسکالر به طور کلی به سه دسته ضرب کننده های سری، موازی و سری- موازی تقسیم می شوند. ضرب کننده های سری دارای سطح پیاده سازی کوچکی هستند ولی به دلیل این که در m سیکل عمل ضرب انجام می شود سرعت آنها پائین است [۱۰،۱۱]. ضرب کننده های موازی دارای سطح اشغالی بزرگی هستند اما به دلیل انجام عمل ضرب در یک سیکل دارای سرعت بالایی می باشند [۱۲،۱۳،۱۴]. ضرب کننده های سری- موازی برای کاربرد های رمزنگاری بسیار مناسب می باشند زیرا بهینه سازی مشترک در زمان اجرا و سطح پیاده سازی توسط این ضرب کننده ها امکان پذیر می باشد [۱۵،۱۶]. نمونه هایی از پیاده سازی ضرب اسکالر با استفاده از ضرب کننده های سری- موازی در [۱۷،۱۸،۱۹] موجود می باشد.

ضرب کننده ای که برای پیاده سازی در این مقاله استفاده شده، ضرب کننده سری- موازی طبق الگوریتم شکل ۳ می باشد و چون ضرب از رقم کم ارزش ضرب شونده شروع شده به ضرب کننده LSD [۹] معروف می باشد.

Input: $a,b \in GF(2^m)$
Output: $c \in GF(2^m)$, $c = ab$ over $GF(2^m)$
Set: $A^{(0)} = a$, $C^{(0)} = 0$, $d = \lceil m/D \rceil$
for i from 1 to d do
$A^{(i)} = A^{(i-1)}\alpha^D \bmod f(x)$, (1)
$C^{(i)} = A^{(i-1)} \cdot B_{i-1} + C^{(i-1)}$ (2)
Where
$A^{(i)} = \sum_{j=0}^{m-1} A_j^{(i)} \alpha^j$
$C^{(i)} = \sum_{j=0}^{m+D-2} c_j^{(i)} \alpha^j$ and
$B_i = \begin{cases} \sum_{j=0}^{D-1} b_{D_{i+j}} \alpha^j & 0 \le i \le d-2 \\ \sum_{j=0}^{m-1-D(d-1)} b_{D_{i+j}} \alpha^j & i = d-1 \end{cases}$
end for
return $C^{(d)} \bmod f(x)$

شکل ۳- الگوریتم ضرب کننده LSD [۱۵].

برای انجام عمل مربع در میدان های باینری قبل از کاهش بین بیت های ورودی صفر قرار می گیرد. حال اگر طبق پیشنهاد [۱۳] مربع و کاهش را ترکیب نماییم می توانیم در یک سیکل عمل مربع را انجام دهیم.

در انتهای الگوریتم شکل ۲ برای تبدیل مختصات تصویری به آفینی به واحد محاسباتی معکوس نیاز می شود. طراحی معکوس کننده طبق قضیه کوچک فرما با استفاده از ضرب و مربع میدانی قابل پیاده سازی است. بنابراین برای محاسبه معکوس برخلاف روش توسعه یافته اقلیدسی نیاز به داشتن یک واحد مجزای معکوس از بین می رود و در پیاده سازی سخت افزاری با هدف صرفه جویی در منابع مصرفی استفاده از روش فرما در طراحی معکوس کننده مناسب بوده و منجر به کاهش پیچیدگی می شود، اگر چه در این روش ارجاع به واحدهای محاسباتی ضرب و مربع زیاد می شود اما می توان در رمزنگاری خم بیضوی با انتخاب مختصات تصویری مناسب استفاده از معکوس را به حداقل ممکن رساند و این مشکل را نیز حل نمود.

برای معکوس در این مقاله طبق قضیه فرما از زنجیره جمع [۲۰] استفاده شده است که به روش معکوس اتو تسوجی معروف می باشد. با این روش تعداد ضرب ها که در روش فرما m-2 است به $\lfloor log_2(m-1) \rfloor + HW(m-1) - 1$ ضرب کاهش می-یابد. $HW(.)$ وزن همینگ، یعنی تعداد یک ها در نمایش باینری آرگومان آن می باشد. الگوریتم اتو تسوجی در شکل ۴ آورده شده است.

[9] Least significant digit

Inputs: Field element a,
Binary representation of $m-1=(m_{l-1},\ldots,m_1,m_0)_2$
Output: $b \equiv a^{(-1)}$
$b \leftarrow a^{m_{l-1}}$,
$e \leftarrow 1$,
for $i = l-2$ downto 0 do
  $b \leftarrow b^{2^e} b$,
  $e \leftarrow 2e$,
  if ($m_i == 1$) then
    $b \leftarrow b^2 a$,
    $e = e+1$,
$b \leftarrow b^2$,

شکل ۴- الگوریتم معکوس اتوتسوجی[۲۰].

۳- معماری پردازنده ضرب اسکالر خم بیضوی

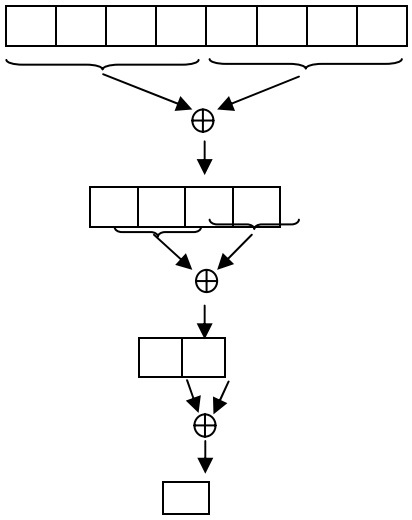
آن چه که به عنوان معماری پردازنده در این قسمت به صورت جزئی بررسی می شود شامل واحد های محاسبات میدانی، مسیر دیتا و رجیسترهای مورد نیاز برای اتصال واحد های محاسبات و واحد کنترل پردازنده می باشد. طراحی برای میدان خاص $GF(2^{163})$ انجام شده است که محاسبات میدانی در پیمانه پنج جمله ای تحویل ناپذیر $f(x) = x^{163}+x^7+x^6+x^3+1$ بنا به پیشنهاد [۲۱] انجام می شود.

۳-۱- واحدهای محاسبات میدان متناهی

واحدهای محاسبات مورد استفاده در پردازنده ضرب، مربع، جمع و معکوس می باشد. همان طور که قبلا ذکر شد ضرب کننده مهمترین واحد محاسباتی است که در تعیین فرکانس کارکرد پردازنده نقش مستقیم دارد. در این مقاله از ضرب کننده LSD استفاده شده و طول کلمات مختلف بررسی شده اند.

چنانچه در معادلات ۱ و ۲ در الگوریتم LSD شکل ۳ نگاه کنیم متوجه این مهم می شویم که می توان دو تابع ضرب و کاهش اصلی را بطور موازی و همزمان پیاده سازی کرد و طول مسیر بحرانی ضرب کننده نیز از ساختار توابع ضرب و کاهش به دست می آید که تابع ضرب در مقایسه با تابع کاهش مسیر بحرانی بلند تری دارد.

چنانچه بخواهیم بیت های یک دنباله را با هم XOR کنیم استفاده از روش درخت باینری[10] برای انجام عمل XOR منجر به کاهش طول مسیر بحرانی می شود. استفاده از روش درخت باینری برای XOR کردن ۸ بیت همانطورکه در شکل ۵ مشاهده می شود باعث کاهش طول مسیر از $7D_{XOR}$ به $3D_{XOR}$ می شود. حال اگر ایده XOR کردن به روش درخت باینری را در مسیر تابع ضرب به کار بریم، مسیر بحرانی واحد محاسباتی ضرب از مقدار $D_{AND}+(D-1)D_{XOR}$ به مقدار $D_{AND}+\lceil log_2(D)\rceil D_{XOR}$ کاهش می یابد که با انجام این کار پریود کارکرد ضرب کننده و در نتیجه پردازنده کاهش می یابد و به این ترتیب با به کار بردن این ایده تاخیر ها را کاهش داده و سرعت اجرای عمل ضرب به عنوان مهمترین عامل در محاسبات ضرب اسکالر را افزایش می دهیم. برای جزئیات بیشتر به [۱] مراجعه نمائید.



شکل ۵ – XOR کردن به روش درخت باینری برای ۸ بیت.

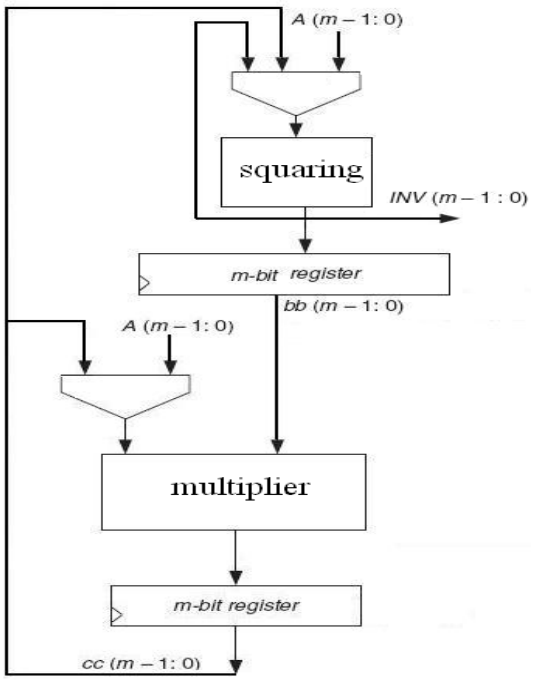
در انتخاب D یا به عبارت دیگر طول ارقام مورد استفاده در ضرب باید ملاحظاتی را رعایت نماییم. از بین D هایی که مقدار $\lceil m/D \rceil$ آن ها یکسان است کوچکترین مقدار را انتخاب می کنیم زیرا با وجود سرعت یکسان برای آن ها D کوچکتر منابع کمتری مصرف می نماید. برای m=163 ، D را می توان کوچکترین مقدار هر فاصله مشخص شده از مجموعه زیر انتخاب نمود.

$$\begin{Bmatrix} 1-14, 15-16, 17-18, 19-20, 21-23, 24-27, \\ 28-32, 33-40, 41-54, 55-81, 82-162, 163 \end{Bmatrix}$$

جهت واحد محاسباتی مربع می توان برای پنج جمله ای تحویل ناپذیر $f(x) = x^{163}+x^7+x^6+x^3+1$ یک واحد محاسباتی ترکیبی پیاده سازی نمود به طوری که در یک پریود ساعت با تاخیر ناچیز و منابع مصرفی کم، عمل مربع را انجام داد.

واحد محاسباتی معکوس با استفاده از روش اتو تسوجی پیاده سازی شده است. به این ترتیب با استفاده از واحدهای ضرب و مربع بدون اشغال فضای اضافی به طراحی واحد محاسباتی معکوس که معماری آن در شکل ۶ آمده است می پردازیم.

[10] Binary tree

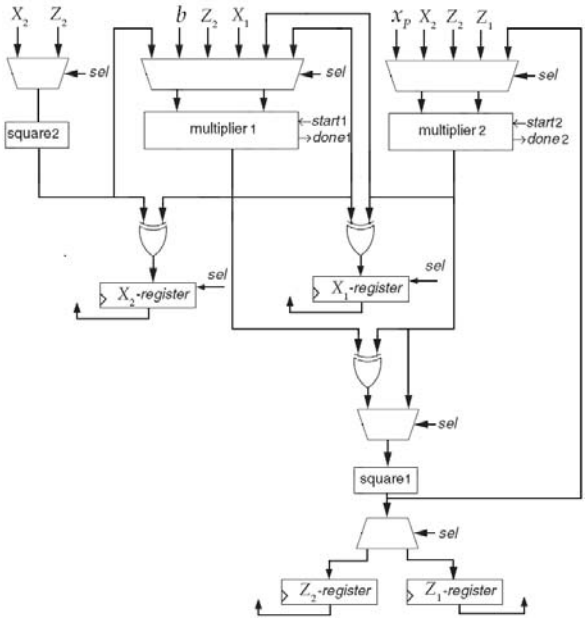


شکل ۶- مسیر داده واحد محاسباتی معکوس کننده.

در طراحی معکوس کننده برای انجام عمل مربع در زنجیره جمع حداکثر نیاز داریم ۸۱ مرتبه عمل مربع تکرار شود. بنابراین ورودی مربع کننده در مسیر دیتای شکل ۶ یک مالتی پلکسر را شامل می شود به گونه ای که تکرار های مختلف را انجام دهد.

۳-۲- ساختار پردازنده

در طراحی این پردازنده تعداد واحدهای محاسبات به گونه ای انتخاب شده است که امکان اجرای موازی محاسبات میدانی در الگوریتم ضرب اسکالر LD وجود داشته باشد به همین دلیل از دو واحد ضرب میدانی در پیاده سازی حلقه اصلی الگوریتم LD که جمع و دو برابر کردن نقطه در آن انجام می شود استفاده می-نماییم. با این کار هر بار تکرار حلقه به جای اجرا با تاخیر ۶ ضرب میدانی با تاخیر ۳ ضرب میدانی انجام می شود. از دو واحد ضرب کننده، یک واحد آن همان ضرب کننده مورد استفاده در معکوس کننده است و با توجه به این که در پیاده سازی جمع و دو برابر کردن نقطه در مختصات تصویری به معکوس نیازی نداریم معماری معکوس کننده را باید به گونه ای طراحی نماییم که امکان استفاده مجزا از واحد ضرب و مربع آن وجود داشته باشد تا به این ترتیب با کارگیری اشتراک منابع سطح اشغالی افزایش پیدا نکند. در شکل ۷ معماری طرح که مربوط به حلقه اصلی الگوریتم LD برای انجام جمع و دو برابر کردن نقطه در هر تکرار است مشاهده می شود.



شکل ۷- معماری جمع و دو برابر کردن در الگوریتم LD.

در طراحی معماری نهایی پردازنده یا به عبارتی تبدیل مختصات تصویری به آفینی(شکل ۲)، عملیات جمع حداقل ۵ بار نیاز می شود و چون روال انجام محاسبات به صورت ترتیبی است، استفاده از یک واحد جمع کننده برای انجام کلیه جمع ها در تبدیل مختصات منجر به کاهش سطح پیاده سازی می شود. در واحد جمع کننده هر عنصر میدان با عنصر دیگر بیت به بیت XOR می شود، که این عمل به صورت ترکیبی و در یک پریود انجام می شود.

در انجام محاسبات پارامترهای مورد نیاز از قبیل ضرایب خم، مرتبه خم و مختصات نقطه مبنا به صورت زیر بر اساس پیشنهاد NIST در [۲۱] انتخاب شده است.

a = 1
b = 0x20A601907B8C953CA1481EB10512F78744A3205FD
n = 0x400000000000000000000292FE77E70C12A4234C33
x = 0x3F0EBA16286A2D57EA0991168D4994637E8343E36
y = 0x0D51FBC6C71A0094FA2CDD545B11C5C0C797324F1

٤- نتایج پیاده سازی و مقایسه

برای پیاده سازی ساختار ضرب اسکالر در الگوریتم LD از کدهای قابل سنتز VHDL استفاده شده است. کدها در محیط ISE11.1 سنتز شده است و تراشه Virtex-4 FPGA (XC4VLX200) به عنوان بستر پیاده سازی انتخاب شده است. مساحت کل این تراشه 89088 slices می باشد.

چون ضرب کننده میدان مورد استفاده در این ساختار به صورت سری- موازی بوده، نتایج حاصل از پیاده سازی ضرب اسکالر برای طول کلمات مختلف در جدول ۲ آمده است.

جدول۲- نتایج سنتز پیاده سازی ضرب اسکالر در این مقاله

D	Freq (MHz)	time(µs)	Area			مساحت اشغالی (درصد)
			slices	LUT	FF	
82	239	16.270	16744	31327	5355	19
55	239	18.362	13198	24761	5301	15
41	239	20.452	11848	22267	5274	14
33	239	22.542	10669	20048	5257	12
28	239	24.632	9706	18220	5247	11
24	235	27.150	9299	17499	5239	10
15	226	37.1	8362	15813	5220	9

برای مقایسه کار انجام شده در این مقاله برخی از بهترین کارهای انجام شده در سال های قبل را ذکر می نماییم.

در [۱۷] از ضرب کننده LSD با تاخیر ۴ سیکل و الگوریتم ضرب اسکالر LD استفاده شده و در معکوس کننده برای کاهش تعداد سیکل ها، تکرار مربع را به کار برده است. در [۱۸] ضرب اسکالر خم بیضوی با استفاده از ضرب کننده LSD پیاده سازی شده است. کار انجام شده در [۱۸] روی میدان $GF(2^{167})$ انجام شده است و ضرب کننده میدانی با تاخیر ۱۱ پریود اجرا می شود، همچنین از ۱۰ بلوک RAM هم استفاده می کند. در [۱۹] برای چند طول میدان مختلف ضرب اسکالر برای بسترهای سخت افزاری و نرم افزاری اجرا شده است. در این کار ضرب کننده میدانی با تاخیر ۳ سیکل اجرا می شود. در [۲۲] از دو ضرب کننده موازی به صورت همزمان در الگوریتم ضرب اسکالر LD استفاده شده، به طوری که این ضرب کننده دارای چندین رجیستر در مسیر بحرانی برای افزایش فرکانس کاری مدار می باشد. در [۲۳] که یکی از سریع ترین کارهای گزارش شده می باشد، با استفاده از یک ضرب کننده pipeline شده و به کار بردن دستورالعمل های سفارشی نتایج حاصل شده است. در [۲۴] ضرب اسکالر را در دو خانواده خم های ابر منفرد و نا ابر منفرد مقایسه نموده است.

جدول۳- نتایج پیاده سازی ضرب اسکالر در کارهای قبلی

ref	m	FPGA	Freq (Mhz)	Time (µs)	Area (slices)	Area (LUT)	Area (FF)
[17]	163	Virtex 2000E	66	233	-	10017	1930
[18]	167	XCV 400E	76.7	210	-	3002	1769
[19]	163	XCV 2000E	66.4	144	-	20068	6321
[22]	163	VinexII V8000	90.2	106	18079	-	-
[23]	163	Virtex-4 L200	153.9	19.55	16209	-	-
[24]	163	VirtexII Pro 30	100	280	8450	-	-

حال نتایج کارهای قبلی در جدول ۳ را با نتایج حاصل از کار خودمان در جدول ۲ مقایسه می نماییم. در مقایسه با سریعترین کارها در [۲۲] و [۲۳] به ازای D=55 نتایج سرعت و سطح پردازنده طراحی شده در این مقاله بهتر می باشد. نتایج پیاده سازی [۱۷]، [۱۸]، [۱۹]، و [۲۴] مساحت اشغالی کمتری نسبت به پیاده سازی پیشنهادی دارد اما زمان اجرای الگوریتم در آنها به مراتب بیشتر از روش ارائه شده در این مقاله است. حال اگر ترکیب نتایج زمان و سطح(بر حسب LUT #) را به طور همزمان با معیار کارایی[11] (bits/(time*area)) مورد ارزیابی قرار دهیم میزان کارایی کار ما به ازای D=55 برابر ۳۵۸ است که در مقایسه با بیشترین کارایی در کارهای قبلی که از [۲۳] با مقدار ۲۶۵ به دست می آید، ۳۷٪ بهبود حاصل شده است. در مجموع می توان ادعا نمود که در کاربردهای سرعت بالا پردازنده پیشنهادی برتری قابل ملاحظه ای نسبت به کارهای ارائه شده تاکنون دارد و چنان چه سرعت و مساحت اشغالی هر دو مد نظر باشند باز هم پیاده سازی با روشی که در این کار تشریح شد منجر به نتایج بهتری خواهد شد.

۵- نتیجه گیری

در این مقاله مهمترین عملیات در رمزنگاری خم بیضوی یعنی ضرب اسکالر را در سطح سخت افزار به صورت یک پردازنده پیاده سازی و سنتز نمودیم. از مهمترین عوامل موثر در پیاده سازی پردازنده استفاده از واحد های محاسباتی بهینه می باشد. به همین دلیل در ضرب کننده با XOR کردن به روش درخت باینری، توانستیم طول مسیر بحرانی را کاهش دهیم. معکوس کننده اتو تسوجی هم به دلیل استفاده از واحدهای ضرب و مربع در کاهش سطح پیاده سازی در مختصات تصویری موثر می باشد. در پیاده سازی الگوریتم ضرب اسکالر LD نیز با اجرای موازی دو ضرب کننده، تاخیر اجرای الگوریتم را کاهش دادیم. در انتهای الگوریتم LD نیز برای تبدیل مختصات تصویری به آفینی با استفاده از به اشتراک گذاری منابع، سطح پیاده سازی را کاهش دادیم. نتایج حاصله در مقایسه با آخرین پیاده سازی های سریع و کارآمد قبلی از نظر سرعت و سطح اشغالی بهبود قابل ملاحظه ای را نشان می-دهد.

۶- سپاسگزاری

این مقاله توسط پژوهشکده پردازش هوشمند علایم مورد حمایت مالی و معنوی قرار گرفته است.

مراجع

[۱] مهدیزاده، حسین، "طراحی و پیاده سازی واحد های محاسبات سریع مبتنی بر رمزنگاری خم بیضوی در میدان $GF(2^{163})$ با استفاده

[11] Efficiency

از کدهای قابل سنتز VHDL " ، پایان‌نامه کارشناسی ارشد مخابرات رمز، دانشگاه شهید ستاری، تهران، آذر ۱۳۸۸.

[2] D. Hankerson, A. Menezes, S. Vanstone, *Guide to elliptic curve cryptography*, Springer-Verlag, 2004.
[3] NIST—National Institute of Standards and Technology, "*Recommended elliptic curves for federal government use*," 2000 [Online]. Available: http://csrc.nist.gov/encryption
[4] M. Rosing. *Implementing Elliptic Curve Cryptography*. Manning Publications Co., 1999.
[5] I. Blake, G. Seroussi, and N. *Smart. Elliptic curves in cryptography*, London Mathematical Society Lecture Note Series 265. Cambridge University Press, 2002.
[6] ANSI, ANSI X9.62 *The elliptic curve digital signature algorithm* (*ECDSA*). Available from: http://www.ansi.org
[7] *Standard Specifications for Public Key Cryptography*, IEEE1363,2000.
[8] J. Lopez and R. Dahab, "*Fast multiplication on elliptic curves over GF*(2^m) *without precomputation*," presented at the Workshop on Cryptographic Hardware Embedded Syst. (CHES), Worcester, MA, 1999.
[9] P. L. Montgomery, "*Speeding the Pollard and elliptic curve methods of factorization*," 1987.
[10] P. A. Scott, S. E. Tavares, and L. E. Peppard, "*A fast VLSI multiplier for GF*(2^m)," IEEE J. Sel. Areas Commun., vol. 4, no. 1, pp. 62–66, Jan. 1986.
[11] E. Mastrovito, "*VLSI architectures for computations in galois fields*," Dept. Elect. Eng., Linkoping Univ., Linkoping, Sweden, pp. 249, 1991.
[12] F. Rodriguez-Henriguez and C. K. Koc, "*Parallel multipliers based on special irreducible pentanomials*," IEEE Trans. Comput., vol. 52, no. 12, pp. 1535–1542, Dec. 2003.
[13] H. Wu, "*Bit-parallel finite field multiplier and squarer using polynomial basis*," IEEE Trans. Comput., vol. 51, no. 7, pp. 750–758, Jul. 2002.
[14] A. Reyhani-Masoleh and M. A. Hasan, "*Low complexity bit parallel architectures for polynomial basis multiplication over GF*(2^m)," IEEE Trans. Comput., vol. 53, no. 8, pp. 945–959, Aug. 2004.
[15] L. Song and K. K. Parhi, "*Low-energy digit-serial/parallel finite field multipliers*," J. VLSI Signal Process. Syst., vol. 19, pp. 149–166, 1998.
[16] M. C. Mekhallalati, A. S. Ashur, and M. K. Ibrahim, "Novel *radix finite field multiplier for GF*(2^m)," J. VLSI Signal Process., vol. 15, pp. 233–245, 1997.
[17] J. Lutz and Hasan, A., "*High performance FPGA based elliptic curve cryptographic co-processor*," in Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2004, Las Vegas, Nevada, USA, Apr. 5-7, 2004, vol. 2, pp. 486-492.
[18] G. Orlando and C. Paar. "*A high-performance reconfigurable elliptic curve processor for GF* (2^m)," In Cryptographic Hardware and Embedded Systems (CHES), 2000, Worcester, MA, 2000.
[19] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, "*An end-to-end systems approach to elliptic curve cryptography*," presented at the Workshop Cryptographic Hardware. Embedded Syst. (CHES), Redwood Shores, CA, 2002.
[20] Itoh, T. and Tsujii, S., "*A fast algorithm for computing multiplicative inverses in GF*(2^m) using normal bases," Information and Computation, vol. 78, no. 3, Sep. 1988, pp. 171-177.

[21] United States Dept. of Commerce/National Institute of Standards and Technology. FIPS 186-2. *Digital Signature Standard* (*DSS*). Federal Information and Processing Standards Publication, 2000
[22] K. Jarvinen, M. Tommiska, and J. Skytta, "*A scalable architecture for elliptic curve point multiplication,*" presented at the ICFPT, Brisbane, Australia, 2004.
[23] W.N. Chelton and M. Benaissa, "*Fast elliptic curve cryptography on FPGA,*" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 2, Feb. 2008, pp. 198-205.
[24]K.Sakiyama,L. Batina, B. Preneel, and I.Verbauwhede, "*Superscalar coprocessor for high-speed curve-based cryptography*," in Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006, Yokohama, Japan, Oct. 10-13, 2006, Lecture Notes in Computer Science, vol. 4249, Springer, pp. 415-429.