کارگاه‌های آموزشی مرکز اطلاعات علمی

آموزش نرم‌افزار Word برای پژوهشگران

روش تحقیق کمی

آموزش مهارت‌های کاربردی در تدوین و چاپ مقالات ISI

طراحی و پیاده سازی یک کمک پردازنده برای رمزنگاری مبتنی بر منحنی های بیضوی

محمد جمشیدی
jamshidi56ir@yahoo.com

بابک صادقیان
BaSadegh@aut.ac.ir

آزمایشگاه امنیت داده ها
دانشگاه صنعتی امیرکبیر- دانشکده مهندسی کامپیوتر

چکیده

در این مقاله طراحی یک کمک پردازنده برای رمزنگاری مبتنی بر منحنی های بیضوی شرح داده شده است. در مقایسه با پیاده سازی های قبلی، ما از الگوریتم مونتگمری برای عملیات ضرب نقطه و به همراه نمایش Optimal Normal Basis برای عناصر میدان استفاده کرده ایم. عملیات کمک پردازنده در میدان $GF(2^{155})$ انجام می پذیرد، ارتباطات کمک پردازنده را طوری در نظر گرفته ایم که بتوان آنرا با هر پردازنده ۱۶ بیتی بکار برد، هدفمان طراحی کمک پردازنده ای بوده که محدودیت زمانی سیستم IFF[1] (سیستم شناسایی دوست یا دشمن) را برآورده سازد و در عین حال کمترین میزان سخت افزار را نیز مصرف کند. طرح کمک پردازنده را برای FPGAهای Xilinx سنتز نموده ایم که برای تراشه XCV300 ، CLB ۲۱۳۹مصرف نموده است و عملیات ضرب نقطه در آن $3.38ms$ طول می کشد که حدوداً دو برابر سریعتر از طرح مشابه [3] است.

کلمات کلیدی: رمزنگاری ، منحنی های بیضوی، کمک پردازنده، FPGA

۱- مقدمه

کوبلیتز و میلر در سال ۱۹۸۵ رمزنگاری منحنی های بیضوی (ECC[2]) را ارائه کردند. ECC در مقایسه با سایر سیستم های رمز کلید عمومی مثل RSA و لگاریتم گسسته مزایای طول کلید کمتر، امنیت بیشتر و سخت افزار مورد نیاز کمتر را دارد. ECC برای کاربردهایی که در آنها محدودیت تعداد بیت وجود دارد مناسب است و مزایای آن عبارتند از [3]:

ECC بالاترین درجه محرمانگی به ازای هر بیت را در بین همه سیستمهای رمز کلید عمومی دارد، بنابراین حافظه کمتری نیاز دارد.

[1] -Identification Friend or Foe
[2] -Elliptic Curve Cryptogrphy

ECC احتمالاً خیلی امن تر از RSA می باشد، زیرا تاکنون بزرگترین پیمانه های حل شده برای RSA ۵۱۲ بیتی و ECC ۹۷ بیتی بوده ، با این تفاوت که توان محاسباتی که برای Crack کردن ECC ۹۷ بیتی بکار برده شده تقریباً ۲ برابر توان محاسباتی است که برای RSA ۵۱۲ بیتی بکار برده شده است. مسأله لگاریتم گسسته اساس سیستم رمز منحنی های بیضوی می باشد.

پیاده سازی های قبلی از رمزنگاری مبتنی بر منحنی های بیضوی شامل پیاده سازی روی تراشه های VLSI و FPGA می باشد. در [11] طراحی و پیاده سازی VLSI یک پردازنده برای سیستم رمز منحنی های بیضوی ارائه شده و در آن از نمایش Normal Basis برای نمایش عناصر میدان استفاده شده است. در این طرح برای $GF(2^{155})$ از 11000 گیت استفاده شده و فرکانس سیکل ساعت آن 40MHz می باشد. در [12] طراحی VLSI یک پردازنده سیستم رمزمنحنی های بیضوی با استفاده از Asynchronous Wave Pipline (AWPs) انجام شده است. در این طرح از نمایش Normal Basis برای نمایش عناصر میدان استفاده شده است. نتایج شبیه سازی Layout این طرح سرعت 1.5GHz را درتکنولوژی $0.35\mu m$ $CMOS$ نشان می دهد. با این سرعت زمان اجرای عملیات ضرب نقطه در $GF(2^{270})$ برابر $434.7\mu s$ می باشد.

در [9] طراحی و پیاده سازی روی FPGA یک پردازنده سیستم رمزنگاری مبتنی بر منحنی های بیضوی ارائه شده و در آن از نمایش Polynomial Basis برای نمایش عناصر میدان استفاده شده است. در این طرح تأکید بر روی میدان های ترکیبی $(GF((2^n)^m))$ بوده است و نتایج شبیه سازی و پیاده سازی روی FPGA ، مدت زمان اجرای عملیات ضرب نقطه را در $GF((2^8)^{21})$ برابر $4.5ms$ نشان می دهد.

در [10] یک کمک پردازنده برای سیستم رمز منحنی های بیضوی پیشنهاد شده است و در آن از نمایش Polynomial Basis برای نمایش عناصر میدان استفاده شده است، این طرح در حالت $GF(2^{163})$ روی FPGA پیاده سازی شده است و فرکانس سیکل ساعت آن 3MHz و مدت زمان اجرای عملیات ضرب نقطه در آن $80ms$ می باشد.

در [3] یک پردازندهٔ Microcode برای سیستم رمز منحنی های بیضوی ارائه شده و در آن از نمایش ONB[1] استفاده شده است. این طرح در حالت $GF(2^{163})$ روی FPGA پیاده سازی شده است و فرکانس سیکل ساعت آن 36MHz و مدت زمان اجرای عملیات ضرب نقطه در آن $6.75ms$ می باشد.

در [2] معماری های پردازندهٔ منحنی بیضوی برای محاسبهٔ عملیات ضرب نقطه در $GF(P),GF(2^m)$ بررسی شده اند، در $GF(2^m)$ از نمایش Polynomial Basis استفاده شده است. این معماری ها برای پیاده سازی روی FPGA مناسب می باشند.

در این مقاله پیاده سازی یک کمک پردازنده برای رمزنگاری منحنی های بیضوی روی FPGAهای Xilinx شرح داده شده است. در اکثر پیاده سازی های قبلی نیاز به یک اینترفیس با پهنای زیاد برای ارتباط با ورودی می باشد، همچنین اکثر آنها از الگوریتم Double and Add برای واحد کنترلشان استفاده کرده اند. ما در اینجا اینترفیس با هر پردازنده ۱۶ بیتی را در نظر گرفته ایم. همچنین از ترکیب نمایش ONB برای عناصر میدان به همراه استفاده از الگوریتم مونتگمری برای واحد کنترل استفاده کرده ایم. طرح کمک پردازنده را برای FPGAهای Xilinx سنتز نموده ایم که برای تراشه XCV300 ، CLB ۲۱۳۹مصرف نموده است و عملیات ضرب نقطه در آن $3.38ms$ طول می کشد.

ادامه مطالب این مقاله به صورت مقابل می باشد: در قسمت ۲ تعریف منحنی های بیضوی آورده شده و در قسمت ۳ عملیات ضرب نقطه و الگوریتم مونتگمری شرح داده شده است، در قسمت ۴ عملیات محاسباتی میدان $GF(2^n)$ بررسی شده اند ، در

[1] -Optimal Normal Basis

قسمت ۵، واحدهای مختلف کمک پردازنده شرح داده شده است و در قسمت ۶ نتایج حاصل از شبیه سازی و سنتز طرح کمک پردازنده آورده شده و نهایتاً در قسمت ۷ نتیجه گیری آورده شده است.

۲- منحنی های بیضوی روی میدان $GF(2^n)$

یک منحنی بیضوی non-supersingular در میدان $GF(2^n)$، مجموعه همه نقاطی است که در معادله زیر صدق می کنند:

$$y^2 + xy = x^3 + ax^2 + b \qquad a, b \in GF(2^n),\ b \neq 0$$

تاکنون هیچ روشی برای تحلیل این منحنی ها یافت نشده که پیچیدگی زمانی آن کمتر از نمایی باشد.

۳- عملیات ضرب نقطه

معمولاً در کاربردهای رمزنگاری از عملیات ضرب نقطه استفاده می شود، عملیات ضرب نقطه به صورت زیر می باشد:

$$Q = k.P = \underbrace{P + P + ... + P}_{k}$$

که P یک نقطه روی منحنی و $k \in GF(2^n)$ یک عدد n بیتی می باشد.

عملیات جمع دو نقطه در حالت مختصات دوبعدی (affine coordinates) و مختصات سه بعدی (projective coordinates) قابل انجام است، در حالت مختصات دوبعدی از عمل معکوس سازی استفاده می شود. عمل معکوس سازی، عمل زمان بری است چون به دنباله ای از عملیات ضرب و توان دوم رسانی تبدیل می شود. اگر از منحنی بیضوی در حالت مختصات سه بعدی استفاده شود عمل معکوس سازی حذف می گردد. برای جزئیات بیشتر در مورد مختصات projective و تبدیل از مختصات affine به projective می توانید به [1] مراجعه کنید.

عملیات جمع دو نقطه در حالت مختصات سه بعدی به صورت زیر می باشد:

با فرض $P = (x_1, y_1, z_1),\ Q = (x_2, y_2, z_2),\ R = P + Q = (X_3, Y_3, Z_3)$، که $x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3 \in GF(2^n)$:

اگر $P \neq Q$ (عملیات Add):

$$X_3 = AD$$
$$Y_3 = CD + A^2(Bx_1 + Ay_1)$$
$$Z_3 = A^3 z_1$$

$$D = A^2(A + az_1) + z_1 BC, C = A + B.$$

اگر $P = Q$ (عملیات Double):

$$X_3 = AB$$
$$Y_3 = x_1^4 A + B(x_1^4 + y_1 z_1 + A)$$
$$Z_3 = A^3$$

$$B = bz_1^4 + x_1^4, A = x_1 z_1$$

الگوریتم های مختلفی برای عملیات ضرب نقطه وجود دارد [2]، که عبارتند از ۶ الگوریتم:

Double-and-Add (binary), ω-ary, Addition-Subtraction, Signed ω-ary, Width-ω Addition-Subtraction , Montgomery

در جدول ۱ پیچیدگی محاسباتی و حافظه مورد نیاز الگوریتم های ضرب نقطه مطابق [2] نمایش داده شده است. پیچیدگی محاسباتی بر مبنای تعداد عملیات Add (یا تفریق) و Double معین شده است ($m = [\log_2^k]$). حافظه مورد نیاز هم به صورت تعداد مکان حافظه n بیتی برای نگهداری عناصر میدان در نظر گرفته شده است.

جدول ۱ : پیچیدگی الگوریتم های ضرب نقطه

الگوریتم	#Point Add	#Point Double	حافظه مورد نیاز
Double-and-Add	m/2	m	4
ω-ary	$2^{\omega-1}+m/\omega$	$2^{\omega-1}+m$	$2^{\omega+1}$
Addition-Subtraction	m/3	m	4
Signed ω-ary	$2^{\omega-2}+m/\omega$	$2^{\omega-2}+m$	2^{ω}
Width-ω Addition-Subtraction	$2^{\omega-2}+m/\omega$	m	$2^{\omega-1}$
Montgomery	m	m	8

با توجه به اینکه عملیات Add و Double در الگوریتم مونتگمری با سایر الگوریتم ها متفاوت است، لذا معیار مقایسه را تعداد عملیات محاسباتی میدان متناهی قرار می دهیم. با توجه به عملیات Add و Double که در فوق شرح دادیم تعداد عملیات محاسباتی میدان متناهی برای Add و Double در مختصات projective به صورت جدول ۲ بدست آوردیم:

جدول ۲: تعداد عملیات محاسباتی میدان متناهی برای عملیات Add و Double

	#Add	#Square	#Mult
Add	۷	۱	۱۳
Double	۵	۴	۷

پیچیدگی محاسباتی الگوریتم های ضرب نقطه بر اساس تعداد عملیات محاسباتی میدان متناهی را در جدول ۳ محاسبه کرده و نشان داده ایم. در این جدول مقدار $\omega = 5$ برای الگوریتم های زیر انتخاب کرده ایم:

ω-ary, Signed ω-ary, Width-ω Addition-Subtraction

با توجه به مقایسه ای که بین الگوریتم های بالا انجام داده ایم الگوریتم مونتگمری دارای کمترین پیچیدگی محاسباتی می باشد و از لحاظ میزان حافظه مورد نیاز نیز در رتبه دوم قرار دارد. با توجه به تعامل بین میزان حافظه و پیچیدگی محاسباتی روش مونتگمری به عنوان بهترین روش انتخاب می شود.

جدول ۳: پیچیدگی محاسباتی الگوریتم های ضرب نقطه براساس تعداد عملیات محاسباتی میدان متناهی

الگوریتم	#Inversion	#Mult	#Square	#Add	حافظه
Double-and-Add	1	13.5m	4.5m	8.5m	4
ω-ary	1	9.6m+320	4.2m+80	6.4m+192	64
Addition-Subtraction	1	11.33m	4.33m	7.33m	4
Signed ω-ary	1	9.6m+160	4.2m+40	6.4m+96	32
Width-ω Addition-Subtraction	1	9.16m+104	4.16m+8	6.16m+56	16
Montgomery	1	7m+10	4m+3	3m+7	8

در شکل ۱ الگوریتم ضرب نقطه مونتگمری برای مختصات projective نمایش داده شده است. عملیاتی که در توابع Mxy و ... انجام می شوند ، عملیات محاسباتی در $GF(2^n)$ می باشند. الگوریتم مونتگمری در حالت مختصات affine هم وجود دارد و برای جزئیات بیشتر در مورد الگوریتم مونتگمری در مختصات های affine و projective می توانید به [6] مراجعه کنید.

۴- عملیات محاسباتی $GF(2^n)$

عناصر میدان اصطلاحاً با Basis نمایش داده می شوند. یکی از طبقه‌بندی های معماری‌های میدان متناهی بر اساس نحوه نمایش عناصر میدان می‌باشد که مهمترین آنها ، نمایش Polynomial Basis (Standard یا Canonical) و Normal Basis می‌باشد. در Normal Basis یک عنصر A به صورت زیر نمایش داده می‌شود:

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i} \qquad a_i \in GF(2), \beta \in GF(2^n)$$

در این نمایش عناصر میدان به صورت بردار باینری n بیتی نمایش داده می‌شوند.

نوع خاصی از Normal Basis را ONB گویند. ONBها منجر به کمترین پیچیدگی برای ضرب‌کننده میدان متناهی می‌شوند. یک ONB در $GF(2^n)$ برای ۲۳٪ از مقادیر $n < 1000$ وجود دارد [7]. دو نوع ONB وجود دارد که به آنها نوع اول و نوع دوم می‌گویند. در $GF(2^{155})$، ONB نوع دوم وجود دارد.

۴-۱- جمع و توان دوم رساننده در $GF(2^n)$

عمل جمع در $GF(2^n)$ بسادگی با XOR کردن بیت‌های متناظر انجام می‌شود. چون از نمایش Normal Basis استفاده می‌کنیم، عمل توان دوم رسانی بوسیله یک عمل انتقال دورانی به چپ انجام می‌شود:

$$\alpha = \sum_{i=0}^{n-1} a_i \beta^{2^i} = (a_{n-1}, a_{n-2}, ..., a_1, a_0) \Rightarrow \alpha^2 = \sum_{i=0}^{n-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{n-1} a_{i-1} \beta^{2^i} = (a_0, a_{n-1}, a_{n-2}, ..., a_1)$$

۴-۲- ضرب در $GF(2^n)$

معمولاً ضرب دشوارترین عمل در $GF(2^n)$ می‌باشد که سرعت و کارآیی سیستم رمز را معین می کند. معماری های ضرب در $GF(2^n)$ به دو دسته bit-serial (یک بیت خروجی در هر سیکل ساعت) و bit-parallel (همه بیتهای خروجی در یک سیکل ساعت محاسبه می‌شوند) تقسیم می‌شوند. معماریهایی به صورت ترکیبی(hybrid) وجود دارند که قسمتی از آن بصورت سریال و قسمتی از آن به صورت موازی اجرا می‌شود. در طرح کمک پردازنده ما از ضرب کننده bit-serial استفاده کرده ایم چون این کمک پردازنده را برای سیستم IFF در نظر گرفته ایم و در آنجا ، محدودیت زمانی زیادی نداریم . عملیات ضرب در $GF(2^n)$ در نمایش Normal Basis به صورت زیر می‌باشد. فرض کنید که $A = \sum_{i=0}^{n-1} a_i \beta^{2^i} = (a_{n-1}, a_{n-2}, ..., a_1, a_0)$ و $B = \sum_{i=0}^{n-1} b_i \beta^{2^i} = (b_{n-1}, b_{n-2}, ..., b_1, b_0)$ عناصر دلخواهی از $GF(2^n)$ و $C = A.B$ باشد، پس داریم:

$$C = \sum_{k=0}^{n-1} c_k \beta^{2^k} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \beta^{2^i} \beta^{2^j} \qquad (۱)$$

$$\beta^{2^i} \beta^{2^j} = \sum_{k=0}^{n-1} \lambda_{ij}^{(k)} \beta^{2^k}, \qquad \lambda_{ij}^{(k)} \in \{0,1\} \qquad (۲)$$

از روابط (۱)و(۲) می توان نتیجه گرفت که:

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \lambda_{ij}^{(k)}, \qquad 0 \le k \le n-1 \qquad (۳)$$

اگر طرفین رابطهٔ (۲) به توان 2^{-1} برسد، عبارت زیر حاصل خواهد شد:

$$\beta^{2^{i-1}} \beta^{2^{j-1}} = \sum_{k=0}^{n-1} \lambda_{i-1,j-1}^{(k)} \beta^{2^k} = \sum_{k=0}^{n-1} \lambda_{ij}^{(k)} \beta^{2^{k-1}} \qquad (۴)$$

با محاسبه ضرایب β^{2^0} داریم:

$$\lambda_{i,j}^{(l)} = \lambda_{i-l,j-l}^{(0)}, \qquad 0 \le i, j, l \le n-1 \tag{۵}$$

رابطه (۳) را می‌توان به صورت زیر نوشت:

$$c_k = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j \lambda_{i-k,j-k}^{(0)} = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{i+k} b_{j+k} \lambda_{ij}^{(0)} \tag{۶}$$

محاسبه ماتریس λ بستگی به مقدار n دارد، پیچیدگی‌سخت‌افزاری ضرب‌کننده بوسیله C_N (تعداد عناصر غیرصفر در ماتریس λ) معین می‌گردد. در حالت ONB مقدار $C_N = 2n-1$ است در این حالت ضرب‌کننده دارای کمترین پیچیدگی سخت‌افزاری می‌باشد.

۴-۲-۱- ضرب‌کننده Massey-Omura

ازمعادلهٔ (٦) می‌توان برای پیاده‌سازی ضرب‌کننده Massey-Omura استفاده کرد. Massey و Omura از یک مدار منطقی با ورودی‌های A و B برای محاسبه c_0 استفاده کرده‌اند[4]. با استفاده از همان مدار منطقی و به وسیلهٔ چرخش دورانی A و B به تعداد k بار می‌توان c_k را محاسبه کرد.

۴-۲-۲- ضرب‌کنندهٔ Agnew

در[8]، Agnew یک معماری برای ضرب‌کننده پیشنهاد داده است خاصیت مهم این روش معماری منظم آن می‌باشد. برای توضیحات بیشتر در مورد این روش می توانید به [4] مراجعه کنید.

۴-۲-۳- مقایسه ضرب‌کننده‌های Massey-Omura و Agnew

ضرب‌کننده‌هایی که در قسمت‌های قبلی بررسی شدند عملیات ضرب را در n سیکل ساعت انجام می‌دهند، پس پیچیدگی زمانی اینها با هم برابر می‌باشد. در [4] مقایسه ای روی تعداد گیت های ضرب کننده های Agnew و Massey-Omura انجام شده که در جدول۵ نتیجه این مقایسه نشان داده شده است.

جدول ۴ : مقایسه تعداد گیت های ضرب کننده های Agnew و Massey-Omura [4]

Massey-Omura	Agnew	
3n	3n	Flip Flop
2n-2	2n	XOR
2n-1	n	AND

با توجه به مقادیر جدول ۵ روش Agnew از لحاظ میزان سخت‌افزار بهینه می‌باشد لذا در طرح ضرب‌کننده کمک پردازنده از روش Agnew استفاده کرده‌ایم.

۴-۳- معکوس کردن در $GF(2^n)$

محاسبه معکوس در $GF(2^n)$ با نمایش Normal Basis از دنباله‌ای ضرب وتوان دوم‌رسانی تشکیل شده‌است تأخیر محاسبه معکوس به تعداد عملیات ضرب آن بستگی دارد بنابراین همه الگوریتم‌های معکوس کردن بر روی کاهش تعداد عملیات ضرب تمرکز دارند. در $GF(2^n)$ و با نمایش Normal Basis معمولاً بهتر است که در عملیات معکوس‌سازی از تئوری Fermat استفاده شود چون عمل توان دوم‌رسانی به سادگی با یک انتقال دورانی به چپ انجام می‌شود. با توجه به تئوری Fermat که در آن $\alpha^{p^n} = \alpha$ معکوس α به صورت زیر محاسبه می‌شود:

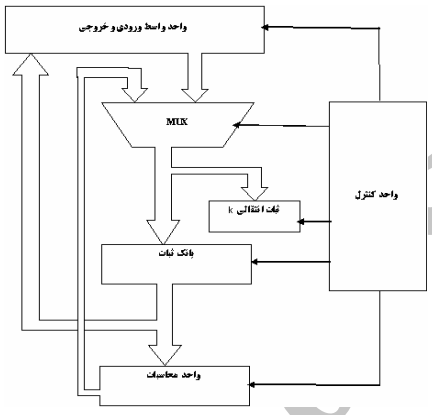
$$\alpha^{2^n} = \alpha \Rightarrow \alpha^{-1} = \alpha^{2^n-2}$$

روش‌هایی که برای تجزیه $2^n - 2$ بکار می‌روند منجر به پیچیدگی‌های مختلفی از لحاظ تعداد عملیات ضرب میدان متناهی و میزان حافظهٔ مورد نیاز برای عملیات معکوس‌سازی می‌شوند.

ما از الگوریتمی که در [5] پیشنهاد شده و مناسب ترین الگوریتم از لحاظ تعداد کمتر عملیات ضرب می باشد استفاده کرده ایم. در این روش محاسبه معکوس $NM = \lfloor \log_2(n-1) \rfloor + H_\omega(n-1) - 2$ عمل ضرب و $NS = (n-1) + H_\omega(n-1) - 2$ عمل توان دوم رسانی نیاز دارد، که $H_\omega(n-1)$ تعداد بیت‌های ۱ نمایش باینری عدد $n-1$ می‌باشد.

۵- واحدهای مختلف کمک پردازنده

کمک پردازنده طراحی شده عملیات رمز نگاری مبتنی بر منحنی های بیضوی را در میدان $GF(2^{155})$ انجام می دهد. ارتباطات این کمک پردازنده با پردازنده های ۱۶ بیتی در نظرگرفته شده است. بلوک دیاگرام کلی طرح کمک پردازنده در شکل ۵ نمایش داده شده است.



شکل ۱ : بلوک دیاگرام کلی طرح کمک پردازنده

قسمت‌های اصلی کمک پردازنده عبارتند از:

۱-واحد واسط ورودی و خروجی: چون مقادیر مورد استفاده ۱۵۵ بیتی و گذرگاه داده ۱۶ بیتی می‌باشد، لذا این واحد برای دریافت و ارسال اطلاعات بصورت ۱۶بیتی استفاده می‌شود.

۲-واحد بانک ثبات: برای نگهداری ضرائب منحنی بیضوی و مقادیر k و مختصات $Q=(x_Q,y_Q), P=(x_P,y_P)$ و همچنین نتایج میانی محاسبات بکار می‌رود.

۳-واحد محاسبات: عملیات محاسباتی $GF(2^{155})$ که شامل جمع و ضرب و توان دوم رساننده می‌باشد و عمل تست صفر شدن در این واحد انجام می‌شود.

۴-واحد کنترل: وظیفهٔ کنترل واحدهای مختلف و همچنین عملیات ضرب نقطه را بر عهده دارد.

۵-واحد ثبات انتقالی k: به علت اینکه واحد کنترل نیاز به مقدار بیت‌های k برای کنترل انجام عملیات ضرب نقطه دارد، لذا k در یک ثبات انتقالی قرارداده می‌شود.

۶- شبیه سازی و سنتز طرح کمک پردازنده

از VHDL برای توصیف طرح کمک پردازنده استفاده شده است. این طرح را برای محصولات شرکت Xilinx سنتز نمودیم. در جدول ۶ نتایج حاصل از سنتز طرح کمک پردازنده روی FPGAهای خانواده های Spartan2، Spartan2E، Virtex و Virtex-II آورده شده است. در این جدول فرکانس سیکل ساعت و همچنین میزان منابع استفاده شده از FPGA نشان داده شده است. ساختار CLBهای FPGAهای این جدول با هم مشابه است و تفاوت آنها در تعداد این منابع و همچنین منابع لازم برای مسیریابی می باشد .

در این جدول زمان لازم برای محاسبهٔ عملیات ضرب نقطه نیز محاسبه شده است این محاسبه بر اساس فرکانس سیکل ساعت و تعداد سیکل های لازم برای محاسبهٔ عملیات ضرب نقطه در بدترین حالت، انجام شده است.

جدول ۵: نتایج حاصل از سنتز طرح کمک پردازنده روی FPGAهای Xilinx

Time for K.P (ms)	Clock(MHz)	Dffs or Latches	Function Generators	CLB Slices	تراشهٔ انتخاب شده از خانواده	خانواده
3.84	48.2	4277	4052	2139	XIS2200	Spartan2
4.29	43.1	4277	4052	2139	XIS2E200	Spartan2E
3.3	56	4277	4052	2139	XCV200	Virtex
2.15	85.7	4277	4052	2139	XCVE200	VirtexE
1.58	117	4277	4060	2139	XC2V500	Virtex-II

۶-۱- مقایسه با طرح [3]

در [3] یک پردازندهٔ Microcode برای سیستم رمز منحنی های بیضوی ارائه شده است که مشابه طرح پیشنهادی ما می باشد. در آن از نمایش ONB استفاده شده و برای واحد ضرب کنندهٔ آن، ضرب کنندهٔ Agnew بکار برده اند. تفاوت طرح پیشنهادی و این طرح در واحد کنترل می باشد، ما از الگوریتم مونتگمری استفاده کرده ایم و در این طرح از الگوریتم Double and Add استفاده شده است. همچنین ما واحد کنترل را با ماشین حالت متناهی (FSM) ایجاد کرده ایم ولی در این طرح از Microcode برای واحد کنترل استفاده کرده اند. هدف از طرح ما طراحی یک کمک پردازنده برای سیستم IFF بوده بطوریکه محدودیت های این سیستم را پوشش دهد. هدف از طرح [3] طراحی یک پردازنده بوده که برای هر مقدار ONB از n بتوان آنرا ساخت در ضمن استفاده از Microcode در آن باعث شده که اینترفیس آن با PC بسیار ساده شود.

با توجه به این که طرح فوق برای XCV300 سنتز شده است، ما نیز طرح پیشنهادی را برای این FPGA سنتز کردیم تا بهتر بتوانیم این طرح ها را با هم مقایسه کنیم. نتایج حاصل از سنتز طرح پیشنهادی و این طرح در جدول ۷ نشان داده شده است.

جدول ۶: مقایسه طرح پیشنهادی و طرح [3]

K.P operation per second	Time for K.P (ms)	Clock(MHz)	CLB Slices	طرح
148	6.75	36	1567	طرح[3]
294	3.38	54.6	2139	طرح پیشنهادی

با توجه به مقادیر جدول ۷ طرح پیشنهادی سریعتر از طرح [3] می باشد، در عوض طرح پیشنهادی مقدار CLB بیشتری استفاده می کند. البته باید توجه داشت که طرح پیشنهادی دارای یک واحد ورودی-خروجی می باشد که این واحد 155 CLB Slice مصرف می کند با توجه به این نکته میزان CLB طرح پیشنهادی 1984 می شود که باز هم بیشتر از میزان CLB طرح [3] می باشد. با توجه به نتایج جدول ۷ و تشابه زیاد طرح پیشنهادی ما و طرح [3] و اینکه اختلاف آنها در واحد کنترلشان می باشد (در طرح پیشنهادی ما از الگوریتم مونتگمری و در طرح آنها از الگوریتم Double-and-Add استفاده شده است) می توان چنین نتیجه گیری کرد که سرعت و تعداد CLB بیشتر طرح پیشنهادی ما به علت استفاده از الگوریتم مونتگمری و پیاده سازی FSM واحد کنترل می باشد.

۷- نتیجه گیری

در این مقاله طراحی یک کمک پردازنده که عملیات رمز نگاری مبتنی بر منحنی های بیضوی را در میدان $GF(2^{155})$ انجام می دهد آورده شده است. بدین منظور مقایسه ای بین الگوریتم های ضرب نقطه انجام داده و بدین نتیجه رسیدیم که الگوریتم

مونتگمری سریعترین الگوریتم ضرب نقطه می باشد. همچنین مقایسه ای بین ضرب کننده های Agnew و Massey-Omura انجام داده ایم که دیدیم ضرب کننده Agnew میزان سخت افزار کمتری مصرف می کند و در طرحمان از ضرب کننده Agnew استفاده کرده ایم. برای اولین بار نیز ترکیب استفاده از نمایش ONB برای عناصر میدان و الگوریتم مونتگمری برای واحد کنترل را بکار برده ایم که این ترکیب منجر به سرعت بهتر در مقایسه با طرح مشابه [3] شد. طرح کمک پردازنده را برای FPGAهای Xilinx سنتز نموده ایم که برای تراشه XCV300، 2139 CLB مصرف نموده است و عملیات ضرب نقطه در آن $3.38ms$ طول می کشد که حدوداً دو برابر سریعتر از طرح مشابه [3] است. این کمک پردازنده برای سیستم IFF طراحی شده است که اگر در سیستم IFF از این کمک پردازنده استفاده شود فرآیند شناسایی $15ms$ طول می کشد. اگر در طی این زمان دو هواپیمای F-14 با ماگزیمم سرعتشان به سمت هم حرکت کنند با طی مسافت ۱۱،۵متر هر هواپیما می تواند دیگری را شناسایی کند.

مراجع

[1] م. جمشیدی، "طراحی و پیاده سازی یک کمک پردازنده خاص منظوره برای سیستم شناسایی دوست یا دشمن در هواپیما"، پایان نامه کارشناسی ارشد، دانشکده مهندسی کامپیوتر و فن آوری اطلاعات، دانشگاه صنعتی امیرکبیر، تهران، ایران، اردیبهشت ۱۳۸۲.

[2] G. Orlando, "Efficient Elliptic Curve Processor Architectures for Field Programmable Logic", PhD's Thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, March, 2002 .

[3] K.H. Leung, K.W. Ma, W.K. Wong, and P.H.W. Leong , "FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor", *In Eight Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '00*, Napa Valley, California, USA, 2000.

[4] S. Sutikno and A. Surya, "An Architecture of $F_{2^{2^N}}$ Multiplier for Elliptic Curves Cryptosystem", *ISCAS'2000, IEEE International Symposium on Circuits and Systems*, pp. I-279_I-282 , May 2000.

[5] L. Gao and G. E. Sobelman , "Improved VLSI Designs for Multiplication and Inversion in $GF(2^m)$ over Normal Bases", *IEEE Trans. On computers* , vol.c-34, no.8, pp. 97-101,Aug. 2000.

[6] J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation" ,*In Cryptographic Hardware and Embedded Systems - CHES'99 (LNCS 1717)*, pages 316-327. Springer-Verlag, 1999.

[7] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone and R. M. Wilson, "Optimal Normal Bases in $GF(p^n)$" , *Discrete Applied Mathematics*, vol. 22, pp. 149-161, 1988/89.

[8] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk and S. A. Vanstone, "An Implementation for a Fast Public Key Cryptosystem", *Journal of Cryptology*, 3(1991), 63-79.

[9] M. Rosner, "Elliptic Curve Cryptosystems on Reconfigurable Hardware", Master's Thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1998.

[10] S. Okada, N. Torii, K. Itoh and M. Takenaka, "Implementation of Elliptic Curve Cryptographic Coprocessor over $GF(2^m)$ on an FPGA", *CHESS 2000, (LNCS 1965)*, pp. 25-40, Springer-Verlag, Berlin Heidelberg, 2000.

[11] G.B. Agnew, R.C. Mullin, and S.A. Vanstone. "An Implementation of Elliptic Curve Cryptosystems over F_2^{155}", *IEEE Journal on Selected areas in Communications*, 11(5):804-813, June 1993.

[12] O. Hauck, A. Katoch and S. A. Huss, "VLSI System Design Using Asynchronous Wave Pipelines: A $0.35\mu m$ CMOS 1.5GHz Elliptic Curve Public Key Cryptosystem Chip", *Proc. IEEE ASYNC 2000*, Eilat, April 2000.

کارگاه‌های آموزشی مرکز اطلاعات علمی

آموزش نرم‌افزار Word برای پژوهشگران

روش تحقیق کمی

آموزش مهارت‌های کاربردی در تدوین و چاپ مقالات ISI