KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK

KASTEELPARK ARENBERG 10, B-3001 LEUVEN (HEVERLEE), BELGIË
TEL.: +32 16 321130 - FAX: +32 16 321986

Compacte implementaties van pairings

Eindwerk voorgedragen tot het behalen van het diploma van Burgerlijk ingenieur, richting elektrotechniek, optie multimedia en signaalverwerking

door:

Anthony Van Herrewege

Promotoren:
Prof. dr. ir. Bart Preneel
Prof. dr. ir. Ingrid Verbauwhede

Dagelijkse begeleiders:
dr. ir. Lejla Batina
ir. Miroslav Knezevic

22 mei 2009

Samenvatting

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van algoritmes

Lijst van afkortingen

ASIC	Application-Specific Integrated Circuit
ECC	Elliptic Curve Cryptography
FPGA	Field-Programmable Gate Array
MALU	Modulaire Arithmetische Logische Unit
MUX	Multiplexer
RAM	Random-Access Memory
VHDL	Very high speed integrated circuits Hardware Description Language

Lijst van symbolen

Algemeen

$\mathbb{F}$	Galois veld
$\mathbb{F}_q$	Galois veld van q elementen
E	Elliptische kromme
$\#E$	Aantaal punten op de elliptische kromme E
x_A	x-coördinaat van het punt A
y_A	y-coördinaat van het punt A
$e(P, Q)$	Tate pairing van P en Q
A_n	Coëfficient van graad n van veelterm A (tenzij expliciet anders vermeld)
$\mathsf{degree}(A)$	Graad van de veelterm A
$\mathsf{Hamm}(A)$	Hamming gewicht van binaire representatie van A
$\mathsf{min}(a, b)$	Minimum uit de lijst van opgegeven argumenten (a en b)

Operators

$\oplus$	exclusieve OR
$\#$	concatenatie
$A \ll a$	logische verschuiving van A over a posities naar links
$A \gg a$	logische verschuiving van A over a posities naar rechts

Hoofdstuk 1

Inleiding

Hoofdstuk 2

Implementatie

In dit hoofdstuk wordt de implementatie van een schakeling voor de berekening van de Tate pairing uit de doeken gedaan. Het uiteindelijke doel is een zo compact mogelijke ASIC implementatie te verkrijgen.

Er zal onderzocht worden welke basisbewerkingen nodig zijn en hoe deze verwezenlijkt kunnen worden in hardware. Vervolgens wordt een schakeling ontworpen die aan de hand hiervan alle nodige berekeningen kan uitvoeren in het gekozen veld. Ten slotte is er dan nog de schakeling die alle berekeningen voor het Miller algoritme (zie Paragraaf **??**) in goede banen leidt. In het volgende hoofdstuk zal de resulterende schakeling dan beoordeeld worden.

Allereerst worden echter noodzakelijke parameters vastgelegd, zodat alle bewerkingen exact gedefiniëerd kunnen worden. Vervolgens wordt bekeken welke beperkingen aan de schakeling opgelegd moeten worden.

2.1 Parameters

Alvorens kan begonnen worden met de implementatie, moeten bepaalde parameters vastgelegd worden. Pas eens dat gedaan is, zijn alle bewerkingen volledig gedefinieerd.

Een belangrijke parameter is het veld waarover gewerkt zal worden. Omdat de berekeningen in een veld $\mathbb{F}_2$ (en extensies ervan) veel eenvoudiger zijn dan die in een veld modulo een priemgetal, wordt ervoor gekozen in zulk een veld te werken.

Ook moet een kromme vastgelegd worden. Er wordt gekozen voor een supersinguliere elliptische kromme. Dat type krommen heeft een kleine embedding degree en geeft aanleiding tot simpele rekenregels voor de verdubbeling en optelling van punten. De vergelijking van zo een kromme E over een veld $\mathbb{F}_{2^m}$ wordt gegeven door

$$E(\mathbb{F}_{2^m}) : y^3 + y = x^3 + x + b,$$

waarbij $b \in \{0, 1\}$. De embedding degree $k = 4$ voor dit type krommen.

Verder worden de volgende hulp variabelen bepaald:

$$\begin{aligned} \delta &= b && m \equiv 1,7 \pmod 8 \\ &= 1-b && m \equiv 3,5 \pmod 8 \\ \nu &= (-1)^\delta \end{aligned}$$

De orde van de kromme is gelijk aan [1, 2]:

$$\#E(\mathbb{F}_{2^m}) = 2^m + \nu\sqrt{2^{m+1}} + 1$$

Om de Tate pairing te kunnen berekenen, moet de variabele b zo gekozen worden dat $\#E$ enkel uit sommaties bestaat. Met andere woorden: b moet zo gekozen worden dat $\nu = 1$.

Alvorens dit kan gebeuren, moet een woordlengte m vastgelegd worden. Deze bepaald de cryptografische sterkte alsook de grootte van het eindresultaat. Des te groter m, des te groter uiteraard de benodigde registers om alle data in op te slaan. In Tabel 2.1 zijn enkele vaak gebruikte woordlengtes voor pairings met hun bijhorende cryptografische sterkte terug te vinden. Er wordt gekozen voor $m = 163$. Dit laat een mooi compromis toe tussen sterkte en compactheid van de implementatie.

Tabel 2.1: Vergelijking van woordlengtes m en de bijhorende cryptografische sterkte van de Tate pairing

m [bit]	sterkte [bit]
81	324
163	652
239	956

Nu m vastgelegd is, kan hetzelfde gedaan worden voor b. Om een aftrekking in de formule voor $\#E$ te voorkomen, dient $b = 1$ te zijn in dit geval. Meteen kan dan ook een l-torsiepunt subgroep over E vastgelegd worden. Daarbij moet er op gelet worden dat $l \mid \#E$. De eenvoudigste keuze is:

$$\begin{aligned} l = \#E &= 2^{163} + \sqrt{2^{163+1}} + 1 \\ &= 2^{163} + 2^{82} + 1. \end{aligned}$$

Nu het veld waarover gewerkt wordt, bepaald is, kan een reductie veelterm P gekozen worden. Bij de keuze daarvan werd uitgegaan van de aangeraden parameters in [3]. De veelterm is:

$$P = z^{163} + z^7 + z^6 + z^3 + 1.$$

Ook moet het extensieveld $\mathbb{F}_{2^{km}}$ gedefinieerd worden waarover de Tate pairing berekend zal worden. Aangezien $k = 4$, kan het veld opgebouwd worden in twee stappen. Eerst wordt $\mathbb{F}_{2^{2m}}$ bepaald als volgt:

$$\mathbb{F}_{2^{2m}} \cong \mathbb{F}_{2^m}[x]/(x^2 + x^1)$$

en hierop wordt het volgende extensieveld gebouwd:

$$\mathbb{F}_{2^{4m}} \cong \mathbb{F}_{2^{2m}}[y]/(y^2 + (x+1)y + 1).$$

Een element van $\mathbb{F}_{2^{4m}}$ kan dus voorgesteld worden als vier elementen van $\mathbb{F}_{2^m}$.

Ten slotte moet een distortiemap voor gebruik in de berekening van de Tate pairing bepaald worden. Hiervoor wordt dezelfde gekozen als in [1]. Een punt $A \in \mathbb{F}_{2^m}$ ondergaat onder de distortie volgende transformatie:

$$\phi : (x_A, y_A) \rightarrow (x_A + s^2, y_A + x_A s + t^6)$$

met $s, t \in \mathbb{F}_{2^{4m}}$. Daarbij moeten s en t voldoen aan volgende vergelijkingen:

$$s^4 + s = 0$$
$$t^2 + t + s^6 + s^2 = 0.$$

Een mogelijke oplossing hiervoor is:

$$s = x + 1$$
$$t = xy.$$

Een distortie van A kan dus ook geschreven worden als:

$$x_\phi = x_A + x$$
$$y_\phi = (x_A + y_A) + x_A \cdot x + xy.$$

Merk op dat de coëfficiënt van 1 van x_ϕ en die van x van y_ϕ beiden gelijk zijn aan x_A. Door deze handige vorm zullen in de implementatie slechts twee registers nodig zijn om de distortie van het punt Q bij te houden.

2.2 Beperkingen

Het doel is de uiteindelijke schakeling zo klein mogelijk te maken, zodat ze gebruikt kan worden in bv. netwerken van sensoren of een smartcard. Beperking van de oppervlakte is dus de belangrijkste factor. Een tweede belangrijke factor is vermogenverbruik, om dit te beperken bestaan specifieke technieken. Het verbruik hangt ook samen met de oppervlakte, dus het beperken daarvan zal het verbruik ten goede komen. In eerste instantie zal dus getracht worden de implementatie zo compact mogelijk te maken. Het verbruik kan verder ook verlaagd worden door een lagere kloksnelheid voor de schakeling te gebruiken, wat uiteraard de rekensnelheid niet bevordert. De rekensnelheid is echter geen prioriteit en dus zal dit aspect bij het ontwerp van de schakelingen genegeerd worden.

Algemeen kan dus gesteld worden dat hoe kleiner het uiteindelijke resultaat is, hoe beter. Het is dus cruciaal de elementen te identificeren die het meeste plaats innemen in een ASIC schakeling. In Tabel 2.2 is de grootte van de belangrijkste elementen te vinden. Deze cijfers gelden enkel bij gebruik van $0.13\mu m$ low leakage technologie van Faraday Technology Corporation. De ordening van de elementen zal echter grotendeels behouden voor andere technologiën. Uit de tabel blijkt dat het gebruik van flip-flops (registers), adders en multiplexers zoveel mogelijk beperkt moet worden.

Tabel 2.2: Grootte van elementen in een ASIC schakeling ($0.13\mu m$ low leakage technologie van Faraday Corporation [4])

Element	Opp. $\left[\frac{\text{gate}}{\text{bit}}\right]$
D flip-flop met reset	6
D flip-flop zonder reset	5.5
full adder	5.5
D latch	4.25
3 ingang MUX	4
2 ingang XNOR	3.75
2 ingang XOR	3.75
2 ingang MUX	2.25
2 ingang OR	1.25
2 ingang AND	1.25
2 ingang NOR	1
2 ingang NAND	1
NOT	0.75

2.3 Modulaire Arithmetische Logische Unit

De kern van de hardware implementatie wordt gevormd door de Modular Arithmetic Logical Unit (MALU) [5, 6]. Dit circuit laat toe basis bewerkingen uit te voeren op getallen. Gezien de beperking die is opgelegd aan de oppervlakte van de schakeling, wordt enkel de optelling geïmplementeerd. Later wordt met behulp daarvan elke andere nodige berekening verwezenlijkt.

Aangezien er in het veld $\mathbb{F}_{2^m}$ gewerkt wordt, is een optelling equivalent aan een XOR bewerking. De bewerking die moet uitgevoerd kunnen worden is:

$$\begin{aligned} T + B &= T \oplus B \\ &= R \mod P \end{aligned}$$

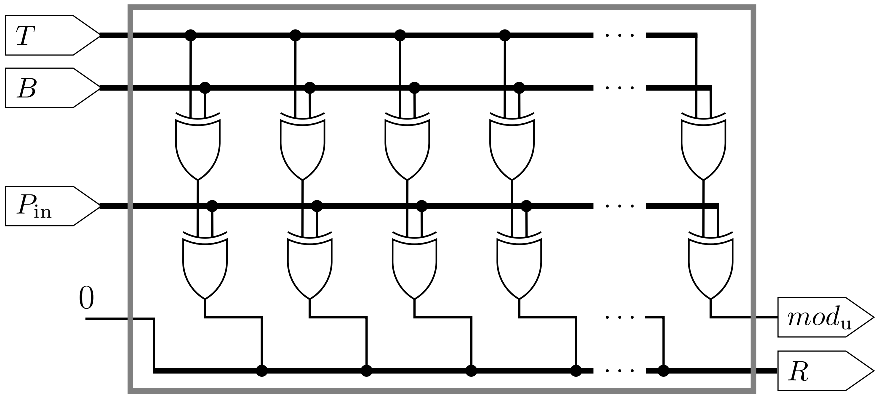
Merk op dat bij een optelling de graad van R enkel kleiner of gelijk kan zijn aan die van T en B. Indien B van graad $< m$ is (dus $\in \mathbb{F}_{2^m}$) en T van graad $\leq m$ ($\in \mathbb{F}_{2^{m+1}}$), is de optelling te implementeren als in Algoritme 2.1.

Algoritme 2.1: Modulo optelling in $\mathbb{F}_{2^m}$

Input: $B \in \mathbb{F}_{2^m}$, $T \in \mathbb{F}_{2^{m+1}}$
Output: $R \mod P \in \mathbb{F}_{2^m}$

$R \leftarrow T \oplus B$
if `degree(`R`)` $= m$ **then**
  $R \leftarrow R \oplus P$

In Paragraaf **??** zal blijken dat het vaak nodig zal zijn om het resultaat R te vermenigvuldigen met z, m.a.w. alle bits 1 plaats naar links te verschuiven. Een voor de hand liggende schakeling die dit alles implementeert, is te zien in Figuur 2.1. Ingang P_{in} dient afhankelijk van de graad van T ingesteld te worden op 0 of P. De ingangen T en P_{in} zijn m bits aangezien het resultaat voor de vermenigvuldiging met z steeds van graad $< m$ is en bit $m+1$ dus toch steeds 0 zou zijn. De hoogste graad term na de shift wordt naar buiten gebracht als mod_{u}. De implementatie bestaat uit $2m$ XOR poorten.



Figuur 2.1: MALU - Basis ontwerp met shift

Aangezien voor het ontwerp het veld en de modulo veelterm op voorhand bepaald zijn, is het mogelijk een zeer groot aantal XOR poorten uit het ontwerp te verwijderen. De ingang P en de bijhorende m XOR poorten kunnen vervangen worden door een 1 bit 'modulo enable' ingang mod_{e} en er worden enkel XOR poorten geplaatst voor de bits i waarvoor $P_i = 1$.

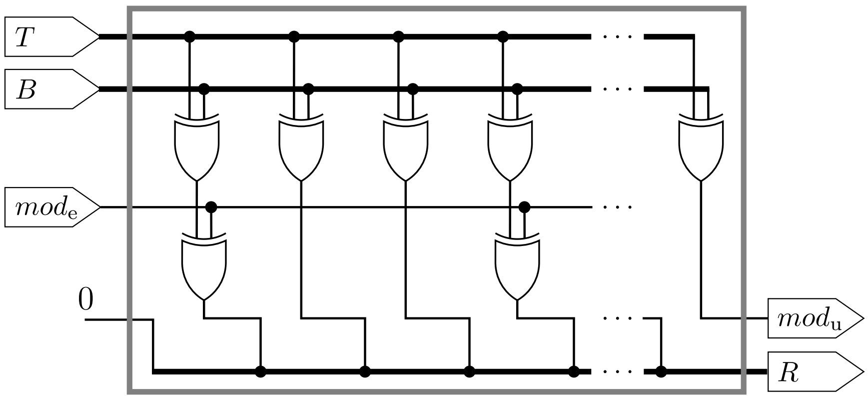
Hierdoor wordt het aantal ingangen drastisch verkleind en worden

$$\Delta = m - (\mathsf{Hamm}(P) - 1)$$

XOR poorten uitgespaard.

In dit geval is $m = 163$ en $P = z^{163} + z^7 + z^6 + z^3 + 1$. Er zijn dus $\mathsf{Hamm}(P) - 1 = 4$ XOR poorten nodig, wat een besparing van $163 - 4 = 159$ XOR poorten oplevert (51% minder dan het oorsponkelijk aantal).

De resulterende schakeling is te zien in Figuur 2.2.



Figuur 2.2: MALU - Geoptimaliseerd ontwerp met shift

2.4 $\mathbb{F}_{2^m}$ kern

2.4.1 Basisontwerp

De eerder ontworpen MALU schakeling laat toe optellingen te doen, maar het Miller algoritme vereist dat er ook vermenigvuldigingen worden uitgerekend. Delingen en machtsverheffingen kunnen met behulp van vermenigvuldiging berekend worden en dienen dus niet rechtstreeks geïmplementeerd te worden. Indien dus zowel optellingen als vermenigvuldigingen berekend kunnen worden, is alles voorhanden om de Tate pairing te berekenen.

Door toepassing van een "shift and add" algoritme, kan de waarde van $A \cdot B = R$ berekend worden met behulp van de MALU schakeling. In Algoritme 2.2 is te zien hoe dit juist in z'n werk gaat. Door de modulo operatie telkens op het tussenresultaat uit te voeren, is het steeds van graad $\leq m$ en kan het opgeslagen worden in T. Op het einde moet het resultaat door z gedeeld worden, wat neerkomt op een verschuiving van alle bits met 1 plaats naar rechts.

Wanneer de optelling en vermenigvuldiging nu in een schakeling gegoten worden, dient te schakeling te weten welke van de twee bewerkingen moet

Algoritme 2.2: "Shift and add" vermenigvuldiging in $\mathbb{F}_{2^m}$

Input: $A, B \in \mathbb{F}_{2^m}$
Output: $R = A \cdot B \in \mathbb{F}_{2^m}$
Data: $T \in \mathbb{F}_{2^{m+1}}$

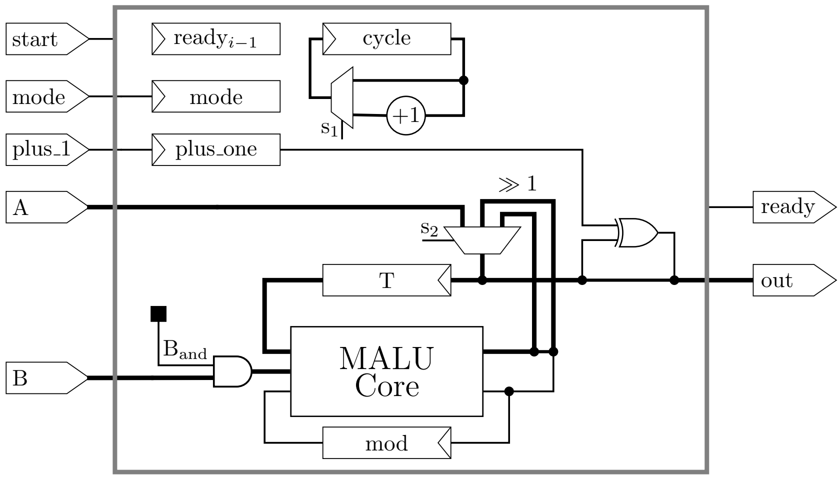
$T \leftarrow 0$
for $i \leftarrow m-1$ **to** 0 **do**
  if $A_i = 1$ **then**
    $b \leftarrow B$
  else
    $b \leftarrow 0$
  $T \leftarrow T \oplus b$
  if `degree(`T`)` $= m$ **then**
    $T \leftarrow T \oplus P$
  $T \leftarrow T \ll 1$
$R \leftarrow T \gg 1$

uitgevoerd worden. Verder moet het mogelijk zijn de uitkomt R in het register T op te slaan. Op die manier is het mogelijk de uitgang van de schakeling gelijk te stellen aan R zolang geen nieuwe berekening gestart wordt.

Verderop zal gezien worden dat in het Miller algoritme verscheidene keren de som $R + 1$ moet berekend worden. Daarom wordt aan de schakeling een ingang *plus_one* toegevoegd die hierin helpt voorzien. De uiteindelijke schakeling is te zien in Figuur 2.3. Er wordt zo veel mogelijk bespaard op registers. Het register *cycle* (equivalent aan i in Algoritme 2.2) is $\lceil \log_2(m) \rceil$ bits lang en register T m bits. De waarde van T_m wordt opgeslagen in register *mod*. Alle overige registers zijn 1 bit groot. Merk dus op dat de variabele T uit Algoritme 2.2 hier gevormd wordt door de combinatie van de registers T en *mod*.

Voor A wordt geen appart register voorzien en in plaats van A_i wordt steeds bit A_m ingelezen voor de vermenigvuldiging. De schakeling die van deze schakeling gebruik maakt, dient dus te voorzien in een methode om elke klokslag de juiste A_i aan te bieden op A_m. Dit kan simpelweg gebeuren door elke klokslag na de start van de berekening het register dat A bevat één positie naar links door te schuiven.

Gezien de eenvoud van de schakeling is het niet nodig een FSM te implementeren, de besturing kan volledig via logica gebeuren. Die wordt getoond in Figuur 2.4. De werking is zeer eenvoudig: zo lang *start* hoog is, worden de registers *mode* en *plus_one* geladen met hun respectievelijke ingangen. Verder wordt, afhankelijk van *mode*, F ingeladen met de waarde van A (optelling, *mode* $= 0$) of gelijkgesteld aan nul (vermenigvuldiging, *mode* $= 1$).



Figuur 2.3: Schakeling voor $\mathbb{F}_{2^m}$ kern

Ook wordt *cycle* gereset. Wanneer *start* laag is, wordt, afhankelijk van de gewenste bewerking en de waarde van *ready*, het register T geladen met de uitgang R van de MALU of het uiteindelijke resultaat $R_{\text{ready}} = R \gg 1$.

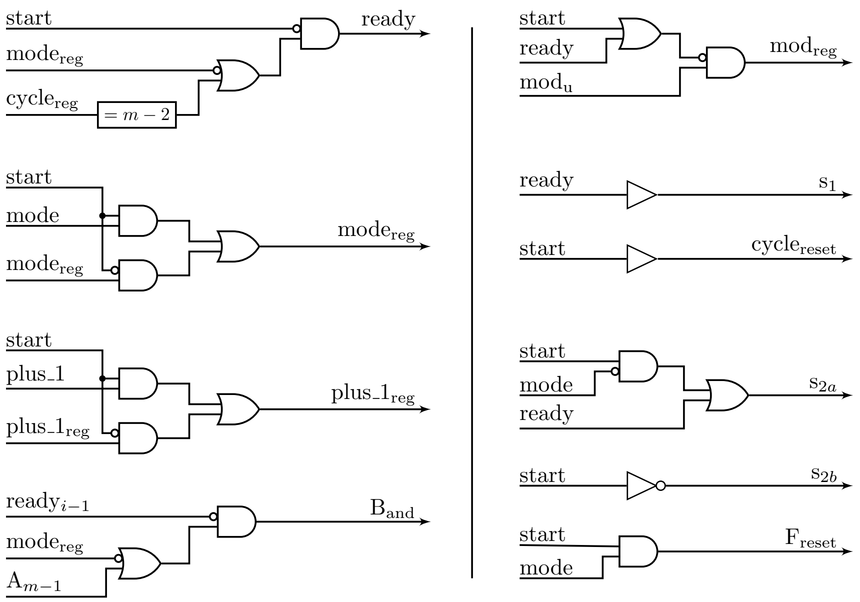
2.4.2 Versnelling van de vermenigvuldiging

Wanneer met behulp van de schakeling in Figuur 2.3 een vermenigvuldiging wordt berekend, zal het m klokcycli duren eer het resultaat beschikbaar is aan de uitgang. Het is echter mogelijk dat aantal drastisch naar beneden te halen door d MALU's te gebruiken en dus d optellingen per klokcyclus uit te voeren. Het principe hiervan wordt geïllustreerd in Figuur 2.5.

De rekentijd van het Miller algoritme zal door toepassing van deze techniek gevoelig verkort kunnen worden. Hoe groter m en hoe meer vermenigvuldigigen er uitgevoerd dienen te worden, des te significanter de tijdswinst die geboekt kan worden. Uiteraard gaat het gebruik van deze techniek wel in tegen de eerder opgelegde beperking aan de grootte van de uiteindelijke schakeling. Het is echter niet zo dat er enkel $d - 1$ extra MALU blokken dienen toegevoegd te worden, afhankelijk van d en m dient ook een extra multiplexer in de schakeling gestoken te worden. Dit is zoals opgemerkt in Paragraaf 2.2 een zeer slechte zaak voor de oppervlakte.

Stel bijvoorbeeld $d = 4$ (en $m = 163$). Het resultaat van een optelling zal net zoals in het standaard ontwerp (Figuur 2.3) aanwezig zijn aan de uitgang van MALU n° 1. Het resultaat van een vermenigvuldiging zal echter aan de uitgang van MALU n° 3 verschijnen, aangezien $163 \bmod 4 = 3$. Het eindresultaat dat in register T dient opgeslagen te worden, is voor een vermenigvuldiging dus

$$R_{3_{\text{ready}}} = mod\text{u}_3 \;\#\; R_{3_{162:1}},$$



Figuur 2.4: Logica voor besturing van de $\mathbb{F}_{2^m}$ kern

terwijl dit voor een optelling

$$R1_{\text{ready}} = mod_{\text{u}_1} \; \# \; R1_{162:1}$$

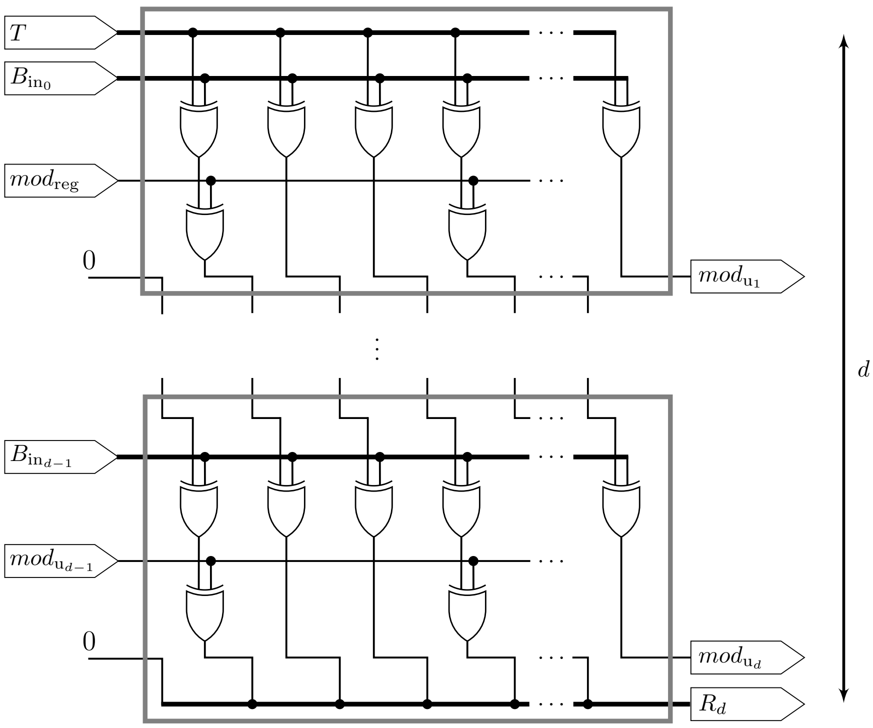
is. Met andere woorden, er dient nu niet enkel gekozen te kunnen worden tussen de ingangen A, R_d of $R1_{\text{ready}}$, maar ook voor $R3_{\text{ready}}$.

Indien men toch wenst het vermenigvuldigen te versnellen, is het aangeraden een d te kiezen waarvoor $m \bmod d = 1$. Als voorbeeld worden enkele voor de hand liggende en optimale keuzes vergeleken voor d indien $m = 163$ in Tabel 2.3.

Tabel 2.3: Voor de hand liggende versus optimale waarden voor woordbreedte d indien $m = 163$

Voor de hand liggende waarden voor d						
d	2	4	8	16	32	64
$m \bmod d$	1	3	3	3	3	35

Ideale waarden voor d						
d	2	3	6	9	18	27
$m \bmod d$	1	1	1	1	1	1



Figuur 2.5: Aaneenschakeling van d MALU's ter versnelling van de vermenigvuldiging

2.5 Controller voor het Miller algoritme

2.5.1 Algemeen ontwerp

Nu een schakeling voorhanden is die toelaat alle benodigde berekeningen uit te voeren, rest nog een schakeling te ontwerpen die het Miller algoritme (Algoritme **??**) uitvoert. Het algoritme met invulling van de vastgelegde parameters, zonder uitwerking van de berekeningen, wordt gegeven in Algoritme 2.3.

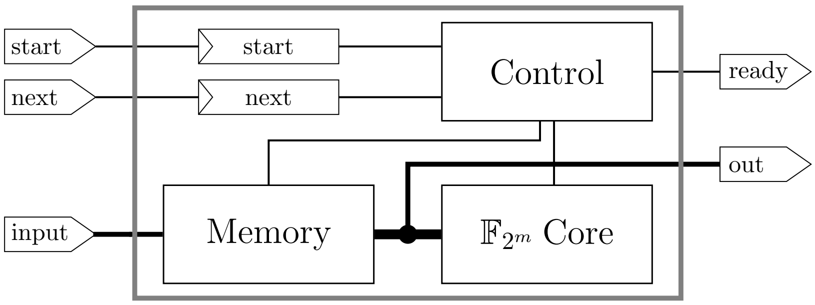
Merk op dat op lijn 6 slechts één waarde moet nagekeken worden, aangezien $l = 2^{163} + 2^{82} + 1$.

De controller zal ontworpen worden zoals het schema in Figuur 2.6. Voor het geheugenblok volledig ontworpen kan worden, moeten echter eerst de verschillende berekeningen uitgewerkt worden. Vervolgens zal aan de hand van die uitwerkingen bepaald worden hoeveel registers ten minste noodzakelijk zijn. Afhankelijk van diezelfde uitwerkingen zullen ook enkele geheugen elementen voor tellers en statusbits toegevoegd moeten worden. Ten slotte zal een FSM ontworpen worden.

Grofweg kan het algoritme opgedeeld worden in de for-lus en een finale

Algoritme 2.3: Miller algoritme voor berekening van de Tate pairing met parameters ingevuld

```
Input: P, Q ∈ E(F_{2^163})[l]
Output: e(P, Q)
F ← 1
I ← P
for i ← 162 to 0 do
    F ← F^2 · G_{I,I}(φ(Q))
    I ← 2I
    if i = 82 then
        F ← F · G_{I,P}(φ(Q))
        I ← I + P
e(P, Q) ← F^{(2^{4·163} − 1)/(2^{163} + 2^{82} + 1)}
return e(P, Q)
```



Figuur 2.6: Schakeling voor de uitvoering van het Miller algoritme

machtsverheffing. De for-lus kan verder onderverdeeld worden in een verdubbelstap, een optelstap, een kwadratering van F en een vermenigvuldiging $F \cdot G$. Elk van deze stappen zal verder uitgediept worden en er zal voor elke berekening bepaald worden hoeveel tussenresultaten minimum opgeslagen moeten worden. Het zal blijken dat een inversie in $\mathbb{F}_{2^m}$ uitgerekend moet kunnen worden, wat ook verder uitgediept zal worden.

Bij elk van de volgende algoritmen zal aangegeven worden hoeveel en welke bewerkingen juist nodig zijn. Daarbij staat A voor een optelling, M voor een vermenigvuldiging, S voor een kwadratering en I voor een inversie. Aangezien er echter geen afzonderlijke schakeling voor kwadrateren ontworpen is, zijn S en M qua rekentijd in dit geval equivalent aan elkaar. De bewerking $a + 1$ neemt geen extra tijd in beslag, omdat die functie parallel met een optelling of vermenigvuldiging kan uitgevoerd worden door de *plus_one* van de $\mathbb{F}_{2^m}$ kern hoog te maken bij de start van een berekening.

2.5.2 For-lus

Zoals reeds eerder vermeld kan de for-lus onderverdeeld worden in een verdubbelstap, een optelstap, een kwadratering van F en een vermenigvuldiging $F \cdot G$. Elk van deze onderdelen zal in de volgende paragrafen in detail aan bod komen.

Verdubbelstap

De verdubbelstap wordt gevormd door lijnen 4 en 5 in Algoritme 2.3. Voor een supersinguliere kromme zijn de berekeningen als volgt [1, 7]:

$$\begin{aligned} \lambda &= x_V^2 + 1 \\ x_{2V} &= \lambda^2 \\ y_{2V} &= \lambda(x_{2V} + x_V) + y_V + 1 \\ G_{V,V}(\phi(Q)) &= \lambda(x_\phi + x_V) + (y_\phi + y_V) \end{aligned}$$

In dit geval kan y_{2V} ook berekend worden als:

$$\begin{aligned} y_{2V} &= y_V^4 + x_V^4 \\ &= (y_V + x_V)^4 \end{aligned}$$

Aangezien dit echter twee kwadrateringen en een optelling kost tegenover een vermenigvuldiging en twee optellingen, wordt de voorkeur gegeven aan de eerste methode.

Door de specifieke vorm van $\phi(Q)$ kan G uitgeschreven worden als:

$$\begin{aligned} G_a &= \lambda(x_{\phi_a} + x_V) + (y_{\phi_a} + y_V) & G_c &= \lambda \cdot x_{\phi_c} + y_{\phi_c} \\ G_b &= \lambda \cdot x_{\phi_b} + y_{\phi_b} & &= 0 \\ &= \lambda + y_{\phi_b} & G_d &= \lambda \cdot x_{\phi_d} + y_{\phi_d} \\ &= \lambda + x_{\phi_a} & &= 1 \end{aligned}$$

De variabele G kan dus opgeslagen worden in twee registers van grootte m in plaats van in vier. De vorm van G zal ook toelaten de vermenigvuldiging $F \cdot G$ grotendeels te vereenvoudigen, zoals later gezien zal worden.

Tanneer dit in rekening gebracht wordt en het algoritme op register niveau wordt uitgeschreven, bekomt men uiteindelijk Algoritme 2.4. Hierbij werd specifiek gelet op een minimum gebruik van tijdelijke registers.

Buiten registers voor x_{2V}, y_{2V}, x_{ϕ_a}, y_{ϕ_a}, G_a en G_b is er ook een register nodig om λ in op te slaan. In totaal moeten er zes optellingen, twee vermenigvuldigingen en twee kwadrateringen uitgerekend worden.

Algoritme 2.4: Uitwerking van de verdubbelstap voor supersinguliere krommen in het Miller algoritme

Input: $x_V, y_V \in E(\mathbb{F}_{2^m})$
Output: $x_{2V}, y_{2V} \in E(\mathbb{F}_{2^m})$; $G \in \mathbb{F}_{2^{4m}}$
Data: $\lambda \in \mathbb{F}_{2^m}$

$G_a \leftarrow x_V$; $G_b \leftarrow y_V$	
$\lambda \leftarrow G_a^2 + 1$; $x_{2V} \leftarrow \lambda^2$	2 S
$y_{2V} \leftarrow x_{2V} + G_a$; $y_{2V} \leftarrow y_{2V} \cdot \lambda$	1 M, 1 A
$y_{2V} \leftarrow y_{2V} + G_b + 1$	1 A
$G_a \leftarrow G_a + x_{\phi_a}$; $G_a \leftarrow G_a \cdot \lambda$	1 M, 1 A
$G_a \leftarrow G_a + y_{\phi_a}$; $G_a \leftarrow G_a + G_b$	2 A
$G_b \leftarrow \lambda + x_{\phi_a}$	1 A

Optelstap

De optelstap bestaat uit lijnen 7 en 8 van Algoritme 2.3. Voor een supersinguliere kromme dienen de volgende bewerkingen uitgevoerd te worden [1, 7]:

$$\begin{aligned}
\lambda &= \frac{y_V + y_P}{x_V + x_P} \\
x_{V+P} &= \lambda^2 + x_V + x_P \\
y_{V+P} &= \lambda(x_{V+P} + x_P) + y_P + 1 \\
G_{V,P}(\phi(Q)) &= \lambda(x_\phi + x_P) + (y_\phi + y_P)
\end{aligned}$$

Net zoals bij de verdubbelstap kan G hier in 2 variabelen opgeslagen worden. Hoewel de optelstap slechts één maal moet worden uitgevoerd, is het uiteraard cruciaal dat ook hier zo weinig mogelijk tijdelijke variabelen gebruikt worden. Op die manier blijft de grootte van de uiteindelijke schakeling het kleinst. De uitgewerkte versie van het algoritme wordt gegeven in Algoritme 2.5.

In tegenstelling tot de verdubbelstap zijn hier twee tijdelijke registers nodig, een voor λ en een voor a. Verder zijn er twee registers nodig voor x_P en y_P. Alles samen dienen er tien optellingen, drie vermenigvuldigingen, twee kwadrateringen en een inversie uitgerekend te worden.

Inversie

De meest tijdrovende stap in de optelstap is de inversie. Zoals reeds vermeld in Paragraaf **??**, kan een inversie in een Galois veld berekend worden door toepassing van de kleine stelling van Fermat:

Algoritme 2.5: Uitwerking van de optelstap voor supersinguliere krommen in het Miller algoritme

Input: $x_V, y_V, x_P, y_P \in E(\mathbb{F}_{2^m})$
Output: $x_{V+P}, y_{V+P} \in E(\mathbb{F}_{2^m})$; $G \in \mathbb{F}_{2^{4m}}$
Data: $\lambda, a \in \mathbb{F}_{2^m}$

$G_a \leftarrow x_V$; $G_b \leftarrow y_V$	
$\lambda \leftarrow G_a + x_P$; $\lambda \leftarrow \lambda^{-1}$	1 I, 1 A
$a \leftarrow G_b + y_P$; $\lambda \leftarrow \lambda \cdot a$	1 M, 1 A
$x_{V+P} \leftarrow \lambda^2 + G_a$; $x_{V+P} \leftarrow x_{V+P} + x_P$	1 S, 2 A
$y_{V+P} \leftarrow x_{V+P} + x_P$; $y_{V+P} \leftarrow y_{V+P} \cdot \lambda$	1 M, 1 A
$y_{V+P} \leftarrow y_{V+P} + y_P + 1$	1 A
$G_a \leftarrow x_{\phi_a} + x_P$; $G_a \leftarrow G_a \cdot \lambda$	1 M, 1 A
$G_a \leftarrow G_a + y_{\phi_a}$; $G_a \leftarrow G_a + y_P$	2 A
$G_b \leftarrow \lambda + x_{\phi_a}$	1 A

$$a^{2^m} = a$$
$$a^{2^m-1} = 1$$
$$a^{2^m-2} = a^{-1}$$

De naieve manier om dit te berekenen zou zijn om a $2^m - 2$ keer met zichzelf te vermenigvuldigingen. In dit geval zou dat betekenen dat er $2^{163}-2$ vermenigvuldigingen moeten uitgevoerd worden, wat uiteraard onhaalbaar is.

Een tweede manier bestaat er in de exponent te ontbinden in machten van 2 en 3. In dat geval zouden er nog 237 vermenigvuldigen nodig zijn.

Er is echter een derde, optimale manier die toegepast kan worden indien de exponent van de vorm $2^m - 2$ is [8, 9]. Dit gaat als volgt in zijn werk. Stel:

$$a^{2^m-2} = (a^{2^{m-1}-1})^2$$

Als wordt aangenomen dat m oneven is, is de exponent van twee na het gelijkheidsteken dus even. Zolang de exponent even is, kan recursief volgende formule toegepast worden:

$$a^{2^i-1} = (a^{2^{\frac{i}{2}}-1})^{2^{\frac{i}{2}}} \cdot a^{2^{\frac{i}{2}}-1}$$

Indien a oneven is, dient volgende formule toegepast te worden:

$$a^{2^i-1} = (a^{2^{i-1}-1})^2 \cdot a$$

Uiteindelijk eindigt men dan bij a^2. Het totaal aantal bewerkingen voor een inversie in $\mathbb{F}_{2^m}$ is $\lfloor \log_2(m-1) \rfloor + \mathsf{Hamm}(m-1) + 1$ vermenigvuldigingen en $m - 1$ kwadrateringen.

In het geval van $m = 163$ is de uiteindelijke keten van bewerkingen zoals gegeven in Algoritme 2.6. Het aantal berekeningen in dat geval is 9 vermenigvuldigingen en 162 kwadrateringen. Er is een register nodig om a bij de houden en twee voor de tussenresultaten a^{2^i-1} en $(a^{2^i-1})^{2^i}$.

Algoritme 2.6: Inversie in $\mathbb{F}_{2^{163}}$

Input: $a \in \mathbb{F}_{2^{163}}$

Output: $a^{-1} \in \mathbb{F}_{2^{163}}$

1	$a^3 \leftarrow a^2 \cdot a$	1 S, 1 M
2	$a^{2^4-1} \leftarrow (a^3)^{2^2} \cdot a^3$	2 S, 1 M
3	$a^{2^5-1} \leftarrow (a^{2^4-1})^2 \cdot a$	1 S, 1 M
4	$a^{2^{10}-1} \leftarrow (a^{2^5-1})^{2^5} \cdot a^{2^5-1}$	5 S, 1 M
5	$a^{2^{20}-1} \leftarrow (a^{2^{10}-1})^{2^{10}} \cdot a^{2^{10}-1}$	10 S, 1 M
6	$a^{2^{40}-1} \leftarrow (a^{2^{20}-1})^{2^{20}} \cdot a^{2^{20}-1}$	20 S, 1 M
7	$a^{2^{80}-1} \leftarrow (a^{2^{40}-1})^{2^{40}} \cdot a^{2^{40}-1}$	40 S, 1 M
8	$a^{2^{81}-1} \leftarrow (a^{2^{80}-1})^2 \cdot a$	1 S, 1 M
9	$a^{2^{162}-1} \leftarrow (a^{2^{81}-1})^{2^{81}} \cdot a^{2^{81}-1}$	81 S, 1 M
10	$a^{-1} \leftarrow (a^{2^{162}-1})^2$	1 S

Kwadratering van F

Bij het uitvoeren van lijn 4 van Algoritme 2.3 moet ook telkens het kwadraat van F berekend worden. De afleiding van de formule daarvoor gaat als volgt:

$$
\begin{aligned}
F^2 &= (F_a + F_b x + F_c y + F_d xy) \cdot (F_a + F_b x + F_c y + F_d xy) \\
&= F_a^2 + F_a F_b x + F_a F_c y + F_a F_d xy + F_b F_a x + F_b^2 x^2 + F_b F_c xy \\
&\quad + F_b F_d x^2 y + F_c F_a y + F_c F_b xy + F_c^2 y^2 + F_c F_d xy^2 + F_d F_a xy \\
&\quad + F_d F_b x^2 y + F_d F_c xy^2 + F_d^2 x^2 y^2 \\
&= (F_a^2 + F_b^2 + F_c^2 + F_d^2) \\
&\quad + (F_a F_b + F_b F_a + F_b^2 + F_c F_d + F_d F_c + F_d^2) x \\
&\quad + (F_a F_c + F_b F_d + F_c F_a + F_c^2 + F_c F_d + F_d F_b + F_d F_c) y \\
&\quad + (F_a F_d + F_b F_c + F_b F_d + F_c F_b + F_c^2 + F_d F_a + F_d F_b + F_d^2) xy \\
&= (F_a^2 + F_b^2 + F_c^2 + F_d^2) + (F_b^2 + F_d^2) x + F_c^2 y + (F_c^2 + F_d^2) xy
\end{aligned}
$$

Mist de originele waarde van F overschreven mag worden, is het mogelijk dit te berekenen zonder gebruik van tijdelijke variabelen. Er zijn dus enkel vier registers nodig voor F. Hoe dat in z'n werk gaat is te zien in Algoritme 2.7. Eén kwadratering van F vraagt vier optellingen en vier kwadrateringen in $\mathbb{F}_{2^m}$.

Algoritme 2.7: Uitwerking van van $F^2 \in \mathbb{F}_{2^{4m}}$

Input: $F = F_a + F_b x + F_c y + F_d xy \in \mathbb{F}_{2^{4m}}$

Output: $F = F^2 \in \mathbb{F}_{2^{4m}}$

1	$F_a \leftarrow F_a + F_c$	1 A
2	$F_a \leftarrow F_a^2$	1 S
3	$F_b \leftarrow F_b + F_d$	1 A
4	$F_b \leftarrow F_b^2$	1 S
5	$F_a \leftarrow F_a + F_b$	1 A
6	$F_c \leftarrow F_c^2$	1 S
7	$F_d \leftarrow F_d^2$	1 S
8	$F_d \leftarrow F_d + F_c$	1 A

Vermenigvuldiging $F \cdot G$

Zoals eerder opgemerkt, is G in zowel de verdubbel- als optelstap niet van volledige rang in het extensieveld. De vermenigvuldiging van F met G kan daardoor vereenvoudigd worden, namelijk als volgt:

$$\begin{aligned}
F \cdot G &= (F_a + F_b x + F_c y + F_d xy) \cdot (G_a + G_b x + xy) \\
&= F_a G_a + F_a G_b x + F_a xy + F_b G_a x + F_b G_b x^2 + F_b x^2 y + F_c G_a y \\
&\quad + F_c G_b xy + F_c xy^2 + F_d G_a xy + F_d G_b x^2 y + F_d x^2 y^2 \\
&= (F_a G_a + F_b G_b + F_d) \\
&\quad + (F_a G_b + F_b G_a + F_b G_b + F_c + F_d)x \\
&\quad + (F_b + F_c G_a + F_c + F_d G_b)y \\
&\quad + (F_a + F_b + F_c G_b + F_d G_a + F_d G_b + F_d)xy
\end{aligned}$$

Indien hier nu de Karatsuba-Ofman techniek ([10, 11]) op wordt toegepast, bekomt men:

$$\begin{aligned}
F \cdot G = &(F_a G_a + F_b G_b + F_d) \\
&+ ((F_a + F_b) \cdot (G_a + G_b) + F_a G_a + F_c + F_d)x \\
&+ (F_c G_a + F_d G_b + F_b + F_c)y \\
&+ ((F_c + F_d) \cdot (G_a + G_b) + F_c G_a + F_a + F_b + F_d)xy
\end{aligned}$$

Deze formule kan uitgerekend wordt met gebruik van drie tijdelijke registers (a, b en c). Verder zijn er vier registers nodig voor F en twee voor G. Merk op dat de oude waarde van F overschreven wordt door het resultaat. In totaal zijn er zes vermenigvuldigingen en veertien optellingen nodig. Algoritme 2.8 beschrijft welke berekeningen juist uitgevoerd moeten worden.

Mist het gebruik van een vierde tijdelijk register zou het mogelijk zijn de berekening $G_a + G_b$ op te slaan. Die wordt nu zowel in lijn 2 als 7 berekend. Er zouden dan slechts dertien optellingen moeten berekend worden, één minder dan [2], waar $y^2 + y + x$ gebruikt wordt als modulo veelterm voor het extensieveld. Aangezien een extra tijdelijk register echter tegen de doelstellingen ingaat, wordt voor de iets langere berekening gekozen.

Algoritme 2.8: Uitwerking van de vermenigvuldiging $F \cdot G$ in het Miller algoritme

Input: $F = F_a + F_b x + F_c y + F_d xy, G = G_a + G_b x + xy \in \mathbb{F}_{2^{4m}}$
Output: $F = F \cdot G \in \mathbb{F}_{2^{4m}}$
Data: $a, b, c \in \mathbb{F}_{2^m}$

1	$a \leftarrow F_a \cdot G_a$; $a \leftarrow a + F_d$	1 M, 1 A
2	$b \leftarrow F_a + F_b$; $c \leftarrow G_a + G_b$	2 A
3	$b \leftarrow b \cdot c$; $b \leftarrow b + a$; $b \leftarrow b + F_c$	1 M, 2 A
4	$c \leftarrow F_b \cdot G_b$; $a \leftarrow a + c$	1 M, 1 A
5	$c \leftarrow F_c \cdot G_a$; $c \leftarrow c + F_b$	1 M, 1 A
6	$F_b \leftarrow b$; $b \leftarrow c$	
7	$c \leftarrow F_c + F_d$; $G_a \leftarrow G_a + G_b$	2 A
8	$c \leftarrow c \cdot G_a$; $c \leftarrow c + b$; $c \leftarrow c + F_a$	1 M, 2 A
9	$F_a \leftarrow a$	
10	$c \leftarrow c + F_d$; $b \leftarrow b + F_c$; $a \leftarrow F_d \cdot G_b$	1 M, 2 A
11	$F_c \leftarrow b + a$; $F_d \leftarrow c$	1 A

2.5.3 Finale machtsverheffing

Eens de for-loop voltooid is, moet F nog gereduceerd worden zodat het eindresultaat $e(P, Q)$ uniek is. Hoe dit gebeurt, wordt onderzocht in de volgende paragrafen. De gebruikte methodes zijn gebaseerd op die in [2], aangepast aan het gekozen extensieveld en geoptimaliseerd om het registerverbruik zo laag mogelijk te houden.

De reductie op het einde van het Miller algoritme bestaat uit de machtsverheffing $e(P, Q) = F^M$, met

$$\begin{aligned} M &= \frac{2^{4m} - 1}{l} \\ &= \frac{(2^{2m} + 1)(2^{2m} - 1)}{l} \\ &= (2^{2m} - 1)(2^m - \nu 2^{\frac{m+1}{2}} + 1) \\ &= (2^{2m} - 1)(2^m + 1) + \nu(1 - 2^{2m})2^{\frac{m+1}{2}} \end{aligned}$$

Aangezien $\nu = 1$ in dit geval, kan de machtsverheffing dus berekend worden als

$$e(P,Q) = \left(F^{2^{2m}-1}\right)^{2^m+1} \cdot \left(F^{1-2^{2m}}\right)^{2^{\frac{m+1}{2}}}$$

Er zal onderzocht worden hoe elk van deze termen berekend kan worden. Stel

$$\begin{aligned} F &= (F_a + F_b x) + (F_c + F_d x)y \\ &= U_0 + U_1 y, \end{aligned}$$

met $U_0, U_1 \in \mathbb{F}_{2^{2m}}$. Met $y^{2^{2m}} = y + x + 1$ is $F^{2^{2m}} = U_0 + U_1 + U_1 x + U_1 y$. Men vindt dus:

$$\begin{aligned} I = F^{2^{2m}-1} &= \frac{F^{2^{2m}}}{F} \\ &= \frac{U_0 + U_1 + U_1 x + U_1 y}{U_0 + U_1 y} \\ &= \frac{(U_0 + U_1 + U_1 x + U_1 y)^2}{(U_0 + U_1 + U_1 x + U_1 y) \cdot (U_0 + U_1 y)} \\ &= \frac{U_0^2 + U_1^2 + U_1^2 x}{U_0^2 + U_1^2 + U_0 U_1 + U_0 U_1 x} + \left[\frac{U_1^2 + U_1^2 x}{U_0^2 + U_1^2 + U_0 U_1 + U_0 U_1 x}\right] y \end{aligned}$$

en

$$\begin{aligned} J = F^{1-2^{2m}} &= \frac{F}{F^{2^{2m}}} \\ &= \frac{U_0 + U_1 y}{U_0 + U_1 + U_1 x + U_1 y} \\ &= \frac{(U_0 + U_1 y)^2}{(U_0 + U_1 + U_1 x + U_1 y) \cdot (U_0 + U_1 y)} \\ &= \frac{U_0^2 + U_1^2}{U_0^2 + U_1^2 + U_0 U_1 + U_0 U_1 x} + \left[\frac{U_1^2 + U_1^2 x}{U_0^2 + U_1^2 + U_0 U_1 + U_0 U_1 x}\right] y \end{aligned}$$

Er moeten dus vier termen in $\mathbb{F}_{2^{2m}}$ berekend worden. Merk echter op dat de noemers van alle breuken in zowel I als J identiek zijn. Er zal dus slechts één inversie in $\mathbb{F}_{2^{2m}}$ uitgerekend moeten worden. Ook zijn de tweede termen van I en J identiek. In totaal moeten dus drie elementen in $\mathbb{F}_{2^{2m}}$ opgeslagen worden alvorens de termen vermenigvuldigd kunnen worden. Dit wil zeggen dat er dus zes registers van grootte m nodig zijn.

Termen van de deelbreuken

Ten eerste worden de drie tellers I_t, J_t en G_t uitgewerkt. Hierbij staan I_t en J_t voor de teller van de eerste breuk van respectievelijk I en J, G_t staat voor de teller van de gemeenschappelijke tweede breuk en G_n voor de gemeenschappelijke noemer. Met andere woorden:

$$\begin{aligned} I &= \frac{I_t}{G_n} + \left[\frac{G_t}{G_n}\right] y \\ J &= \frac{J_t}{G_n} + \left[\frac{G_t}{G_n}\right] y \end{aligned}$$

Het kwadraat van een element $A \in \mathbb{F}_{2^{2m}}$ is

$$\begin{aligned} A^2 &= a_0^2 + a_1^2 x^2 \\ &= (a_0^2 + a_1^2) + a_1^2 x \end{aligned}$$

en de vermenigvuldiging van $A, B \in \mathbb{F}_{2^{2m}}$:

$$\begin{aligned} A \cdot B &= a_0 b_0 + a_0 b_1 x + a_1 b_0 x + a_1 b_1 x^2 \\ &= (a_0 b_0 + a_1 b_1) + (a_0 b_1 + a_1 b_0 + a_1 b_1) x \end{aligned}$$

Zodoende bekomt men:

$$\begin{aligned} I_t &= [(F_a^2 + F_b^2) + F_b^2 x] + [(F_c^2 + F_d^2) + F_d^2 x] + [F_d^2 + F_c^2 x] \\ &= (F_a^2 + F_b^2 + F_c^2) + (F_b^2 + F_c^2 + F_d^2) x \\ J_t &= [(F_a^2 + F_b^2) + F_b^2 x] + [(F_c^2 + F_d^2) + F_d^2 x] \\ &= (F_a^2 + F_b^2 + F_c^2 + F_d^2) + (F_b^2 + F_d^2) x \\ G_t &= [(F_c^2 + F_d^2) + F_d^2 x] + [F_d^2 + F_c^2 x] \\ &= F_c^2 + (F_c^2 + F_d^2) x \end{aligned}$$

Voor de gemeenschappelijke noemer G_n vind men:

$$\begin{aligned} G_n &= [(F_a^2 + F_b^2) + F_b^2 x] + [(F_c^2 + F_d^2) + F_d^2 x] \\ &\quad + [(F_a F_c + F_b F_d) + (F_a F_d + F_b F_c + F_b F_d) x] \\ &\quad + [F_a F_d + F_b F_c + F_b F_d) + (F_a F_c + F_a F_d + F_b F_c) x] \\ &= (F_a^2 + F_b^2 + F_c^2 + F_d^2 + F_a F_c + F_a F_d + F_b F_c) \\ &\quad + (F_b^2 + F_d^2 + F_a F_c + F_b F_d) x \\ &= (F_a^2 + F_b^2 + F_c^2 + F_d^2 + (F_a + F_b) \cdot (F_c + F_d) + F_b F_d) \\ &\quad + (F_b^2 + F_d^2 + F_a F_c + F_b F_d) x \end{aligned}$$

Een methode om deze vier resultaten uit te rekenen is te zien in Algoritme 2.9. Er zijn vier registers nodig voor F en acht voor I_t, J_t, G_t en G_n. Tevens moet er nog één tijdelijk register voorzien worden voor a. De uitkomsten kunnen bepaald worden na twaalf optellingen, drie vermenigvuldigingen en vier kwadrateringen.

Algoritme 2.9: Uitwerking van berekening van noemers voor de finale machtsverheffing in het Miller algoritme

Input: $F = F_a + F_b x + F_c y + F_d xy \in \mathbb{F}_{2^{4m}}$
Output: $I_t = I_{t_a} + I_{t_b} x, J_t = J_{t_a} + J_{t_b} x, G_t = G_{t_a} + G_{t_b} x,$
$G_n = G_{n_a} + G_{n_b} x \in \mathbb{F}_{2^{2m}}$
Data: $a \in \mathbb{F}_{2^m}$

1	$I_{t_a} \leftarrow F_a^2;\ I_{t_b} \leftarrow F_b^2;\ G_{t_a} \leftarrow F_c^2;\ G_{t_b} \leftarrow F_d^2$	4 S
2	$I_{t_a} \leftarrow I_{t_a} + I_{t_b};\ J_{t_b} \leftarrow I_{t_b} + G_{t_b}$	2 A
3	$G_{t_b} \leftarrow I_{t_b} + G_{t_b};\ J_{t_a} \leftarrow I_{t_a} + G_{t_b}$	2 A
4	$I_{t_a} \leftarrow I_{t_a} + G_{t_a};\ I_{t_b} \leftarrow I_{t_b} + G_{t_b}$	2 A
5	$G_{n_a} \leftarrow F_a + F_b;\ G_{n_b} \leftarrow J_a \cdot J_c$	1 M, 1 A
6	$a \leftarrow F_c + F_d;\ G_{n_a} \leftarrow G_{n_a} \cdot a;\ a \leftarrow F_b \cdot F_d$	2 M, 1 A
7	$G_{n_a} \leftarrow G_{n_a} + a;\ G_{n_b} \leftarrow G_{n_b} + a$	2 A
8	$G_{n_a} \leftarrow G_{n_a} + J_{t_a};\ G_{n_b} \leftarrow G_{n_b} + J_{t_b}$	2 A

Inversie in $\mathbb{F}_{2^m}$

Vervolgens moet de inverse van G_n berekend worden, wat een inversie in $\mathbb{F}_{2^{2m}}$ is [2].

Stel $A = A_a + A_b x \in \mathbb{F}_{2^{2m}}, A \neq 0$ met multiplicatieve inverse $B = B_a + B_b x \in \mathbb{F}_{2^{2m}}$. Volgens de definitie is $A \cdot B = 1$. Gegeven $x^2 = x + 1$, gelden dus de vergelijkingen:

$$\begin{cases} A_a B_a + A_b B_b = 1 \\ A_a B_b + A_b B_a + B_b A_a = 0 \end{cases}$$

De oplossing van dit stelsel is

$$\begin{aligned} B_a &= w^{-1} \cdot (A_a + A_b) \\ B_b &= w^{-1} \cdot A_b \end{aligned}$$

met $w = A_a^2 + (A_a + A_b) \cdot A_b$. Een inversie in $\mathbb{F}_{2^{2m}}$ kan dus berekend worden via een inversie in $\mathbb{F}_{2^m}$. Uitgewerkt geeft dit Algoritme 2.10, waarbij de oorspronkelijke A wordt overschreven door zijn inverse. Het algoritme kost in totaal drie vermenigvuldigingen, één kwadratering, twee optellingen en één inversie in $\mathbb{F}_{2^m}$. Er zijn twee registers nodig om A in op te slaan en drie tijdelijke registers voor a, b en c. Verder heeft het algoritme voor inversie in $\mathbb{F}_{2^m}$ twee tijdelijke registers nodig, waarbij tijdelijk register b de taak van één van deze twee kan vervullen, aangezien dat niet meer gebruikt wordt na regel 2 in het algoritme. Dit alles samen komt dus neer op vier tijdelijke registers.

Algoritme 2.10: Uitwerking van $A^{-1} \in \mathbb{F}_{2^{2m}}$

Input: $A = A_a + A_b x \in \mathbb{F}_{2^{2m}}, A \neq 0$
Output: $B = A^{-1} = B_a + B_b x \in \mathbb{F}_{2^{2m}}$
Data: $a, b, c \in \mathbb{F}_{2^m}$

$a \leftarrow A_a + A_b$; $b \leftarrow A_a^2$	1 S, 1 A
$c \leftarrow a \cdot A_b$; $c \leftarrow c + b$	1 M, 1 A
$c \leftarrow c^{-1}$	1 S
$B_a \leftarrow a \cdot c$; $B_b \leftarrow A_b \cdot c$	2 M

Berekening van I en J

Gewapend met al deze waarden is het mogelijk $I = I_0 + I_1 y$ en $J = J_0 + J_1 y$ te berekenen. In totaal zijn hier zes vermenigvuldigingen in $\mathbb{F}_{2^{2m}}$ nodig.

Stel $A = A_a + A_b x, B = B_a + B_b x \in \mathbb{F}_{2^{2m}}$. De vermenigvuldiging kan dan geschreven worden als:

$$\begin{aligned} C &= (A_a + A_b x) + (B_a + B_b x) \\ &= (A_a B_a + A_b B_b) + (A_a B_b + A_b B_a + A_b B_b)x \\ &= (A_a B_a + A_b B_b) + \big((A_a + A_b) \cdot (B_a + B_b) + A_a B_a\big)x \end{aligned}$$

In dit geval dienen zowel I_t, J_t als G_t met G_n^{-1} vermenigvuldigd te worden. Vooropgesteld dat de tellers na vermenigvuldiging met G_n niet meer nodig zijn en dus overschreven mogen worden, kan Algoritme 2.11 gebruikt worden. Er zijn dan zes registers nodig voor de uitkomsten, twee voor G_n^{-1} en twee tijdelijke register voor a en b. De vermenigvuldiging in $\mathbb{F}_{2^{2m}}$ kan berekend worden in drie vermenigvuldigingen en vier optellingen.

Algoritme 2.11: Uitwerking van $A \cdot B \in \mathbb{F}_{2^{2m}}$

Input: $A = A_a + A_b x, B = B_a + B_b x \in \mathbb{F}_{2^{2m}}$
Output: $A = A \cdot B \in \mathbb{F}_{2^{2m}}$
Data: $a, b \in \mathbb{F}_{2^m}$

$a \leftarrow A_b \cdot B_b$; $b \leftarrow A_a + A_b$	1 M, 1 A
$A_a \leftarrow A_a \cdot B_a$; $A_b \leftarrow B_a + B_b$	1 M, 1 A
$A_b \leftarrow A_b \cdot b$	1 M
$A_b \leftarrow A_a + A_b$; $A_a \leftarrow A_a + a$	2 A

Berekening van I^{2^m+1}

Nu I en J bekend zijn, moeten ze beiden tot de correcte macht verheven worden. Voor I is dat $2^m + 1$, wat simpelweg mogelijk zou zijn door I^{2^m} te berekenen en dit resultaat te vermenigvuldigen met I. Dit zou een vermenigvuldiging van twee elementen in $\mathbb{F}_{2^{4m}}$ zijn, wat, zoals verderop zal gezien worden, negen vermenigvuldigingen en tweeëntwintig optellingen kost. Er is echter een snellere manier.

Uitgaande van de gelijkheid $A^{2^m} = A$ voor $A \in \mathbb{F}_{2^m}$, weet men dat de vier elementen van I na de machtsverheffing simpelweg zullen bestaan uit sommen van de beginwaarden. Via volgende gelijkheden:

$$\begin{aligned} x^{2^m} &= 1 + x \\ y^{2^m} &= 1 + x + y + xy \\ (xy)^{2^m} &= x + xy \end{aligned}$$

en met $I = I_a + I_b x + I_c y + I_d xy$, vindt men dus:

$$I^{2^m} = (I_a + I_b + I_c) + (I_b + I_c + I_d)x + I_c y + (I_c + I_d)xy.$$

Vervolgens vermenigvuldigd men dit met de originele I, wat resulteert in $I^{2^m+1} = r_a + r_b x + r_c y + r_d xy$:

$$\begin{aligned} r_a &= (I_a + I_b + I_c) \cdot I_a + (I_b + I_c + I_d) \cdot I_b + I_c^2 + (I_c + I_d) \cdot I_d \\ r_b &= (I_a + I_b + I_c) \cdot I_b + (I_b + I_c + I_d) \cdot I_a + (I_b + I_c + I_d) \cdot I_b \\ &\quad + I_c I_d + (I_c + I_d) \cdot I_c + (I_c + I_d) \cdot I_d \\ r_c &= (I_a + I_b + I_c) \cdot I_c + (I_b + I_c + I_d) \cdot I - d + I_a I_c + I_c^2 + I_c I_d \\ &\quad + (I_c + I_d) \cdot I_b + (I_c + I_d) \cdot I_c \\ r_d &= (I_a + I_b + I_c) \cdot I_d + (I_b + I_c + I_d) \cdot I_c + (I_b + I_c + I_d) \cdot I_d \\ &\quad + I_b I_c + I_c^2 + (I_c + I_d) \cdot I_a + (I_c + I_d) \cdot I_b + (I_c + I_d) \cdot I_d. \end{aligned}$$

Dit kan vereenvoudigd worden tot:

$$\begin{aligned} r_a &= I_a^2 + I_a I_b + I_a I_c + I_b^2 + I_b I_c + I_b I_d + I_c^2 + I_c I_d + I_d^2 \\ r_b &= I_a I_c + I_a I_d + I_b I_d + I_c I_d + I_c^2 + I_d^2 \\ r_c &= I_c^2 + I_c I_d + I_d^2 \\ r_d &= I_a I_c + I_b I_c + I_b I_d. \end{aligned}$$

Voor de toepassing van de Karatsuba-Ofman techniek worden de volgende tussenresultaten uitgerekend:

$$\begin{aligned} m_0 &= (I_a + I_b) \cdot (I_c + I_d) \qquad & m_1 &= I_a I_b \\ m_2 &= I_a I_d & m_3 &= I_b I_c \\ m_4 &= I_c I_d & & \\ s_0 &= (I_a + I_b)^2 & s_1 &= (I_c + I_d)^2. \end{aligned}$$

Aan de hand van deze waarden kan het uiteindelijke resultaat berekend worden:

$$\begin{aligned} I^{2^m+1} &= (m_0 + m_1 + m_2 + m_4 + s_0 + s_1) \\ &\quad + (m_0 + m_3 + m_4 + s_1)x \\ &\quad + (m_4 + s_1)y + (m_0 + m_2)xy. \end{aligned}$$

Ook bij de implementatie van dit algoritme wordt er van uit gegaan dat de beginvariable, I in dit geval, niet behouden dient te worden. Er zijn dan uiteindelijk vier registers nodig voor het resultaat en vijf voor tussenresultaten. De totale kost komt neer op vijf vermenigvuldigingen, twee kwadrateringen en negen optellingen. Algoritme 2.12 geeft een overzicht van de bewerkingen.

Algoritme 2.12: Uitwerking van $I^{2^m+1} \in \mathbb{F}_{2^{4m}}$

Input: $I = I_a + I_b x + I_c y + I_d xy \in \mathbb{F}_{2^{4m}}$
Output: $I = I^{2^m+1} \in \mathbb{F}_{2^{4m}}$
Data: $a, b, c, d, e \in \mathbb{F}_{2^m}$

$a \leftarrow I_a \cdot I_b$; $b \leftarrow I_a + I_b$	1 M, 1 A
$c \leftarrow b^2$; $d \leftarrow I_a \cdot I_d$	1 M, 1 S
$I_b \leftarrow I_b \cdot I_c$; $e \leftarrow I_c + I_d$	1 M, 1 A
$b \leftarrow b \cdot e$; $e \leftarrow e^2$	1 M, 1 S
$I_c \leftarrow I_c \cdot I_d$	1 M
$I_c \leftarrow I_c + e$; $I_a \leftarrow I_c + b$	2 A
$I_b \leftarrow I_a + I_b$; $I_d \leftarrow b + d$	2 A
$b \leftarrow c + d$; $I_a \leftarrow I_a + b$	2 A
$I_a \leftarrow I_a + a$	1 A

Berekening van $J^{2^{\frac{m+1}{2}}}$

De voorlaatste berekening die gemaakt moet worden, is de machtsverheffing van J. Helaas is het in dit geval niet mogelijk om de bewerking even snel uit te voeren als de kwadratering van I. De handigste oplossing in dit geval is J simpelweg $\frac{m+1}{2}$ opeenvolgende keren te kwadrateren. Hiervoor kan Algoritme 2.7 gebruikt worden. In totaal zal deze stap $2 \cdot (m+1)$ vermenigvuldigigen en optellingen vragen, wat voor de gekozen $m = 163$ neerkomt op 328 maal beide bewerkingen.

Vermenigvuldiging van $I \cdot J$

De finale stap is de vermenigvuldiging van de twee resulterende variabelen I en J. Dit is een vermenigvuldiging in $\mathbb{F}_{2^{4m}}$ en deze kan als volgt geformuleerd worden:

$$\begin{aligned} e(P,Q) &= I \cdot J \\ &= (I_a J_a + I_b J_b + I_c J_c + I_d J_d) \\ &\quad + (I_a J_b + I_b J_a + I_b J_b + I_c J_d + I_d J_c + I_d J_d)x \\ &\quad + (I_a J_c + I_b J_d + I_c J_a + I_c J_c + I_d J_d + I_d J_b + I_d J_c)y \\ &\quad + (I_a J_d + I_b J_c + I_b J_d + I_c J_b + I_c J_c + I_d J_a + I_d J_b + I_d J_d)xy \end{aligned}$$

Om de berekening via de Karatsuba-Ofman techniek te versnellen, worden eerst volgende tussenresultaten uitgerekend:

$$
\begin{aligned}
m_0 &= I_a J_a & m_1 &= I_b J_b \\
m_3 &= I_c J_c & m_4 &= I_d J_d \\
l_0 &= (I_a + I_b) \cdot (J_a + J_b) & l_1 &= (I_c + I_d) \cdot (J_c + J_d) \\
l_2 &= (I_a + I_d) \cdot (J_a + J_d) & l_3 &= (I_a + I_c) \cdot (J_a + J_c) \\
n_0 &= I_a + I_b + I_c + I_d & n_1 &= J_a + J_b + J_c + J_d \\
p_0 &= n_0 \cdot n_1
\end{aligned}
$$

Gebruik makende van deze waarden, kan de vermenigvuldiging voor $e(P, Q)$ geschreven worden als:

$$
\begin{aligned}
e(P, Q) = {} & (m_0 + m_1 + m_2 + m_3) + (m_0 + m_2 + l_0 + l_1)x \\
& + (m_0 + m_1 + m_2 + l_1 + l_2 + l_3)y + (m_0 + m_3 + l_0 + l_1 + l_3 + p_0)xy
\end{aligned}
$$

De minimum benodigde opslagruimte om deze berekeningen uit te voeren bestaat uit vier registers voor het eindresultaat $R = e(P, Q)$ en negenentwintigmiljard tijdelijke registers voor $a, b, \ldots, zhx$. Dit alles samen vergt negen vermenigvuldigingen en tweeëntwintig optellingen, twee meer dan in [2]. Het resulterend algoritme is terug te vinden in Algoritme 2.13.

Algoritme 2.13: Uitwerking van $I \cdot J \in \mathbb{F}_{2^{4m}}$

Input: $I = I_a + I_b x + I_c y + I_d xy, J = J_a + J_b x + J_c y + J_d xy \in \mathbb{F}_{2^{4m}}$
Output: $R = I \cdot J \in \mathbb{F}_{2^{4m}}$
Data: $a, b, \ldots, zhx \in \mathbb{F}_{2^m}$

$k_3 \leftarrow J_c + J_d$; $k_5 \leftarrow J_b + J_d$	2 A
$m_3 \leftarrow I_d \cdot J_d$; $k_4 \leftarrow I_b + I_d$	1 M, 1 A
$k_1 \leftarrow I_c + I_d$; $m_1 \leftarrow I_b \cdot J_b$	1 M, 1 A
$k_0 \leftarrow I_a + I_b$; $k_2 \leftarrow J_a + J_b$	2 A
$k_7 \leftarrow J_a + J_c$; $m_2 \leftarrow I_c \cdot J_c$	1 M, 1 A
$m_0 \leftarrow I_a \cdot J_a$; $k_6 \leftarrow I_a + I_c$	1 M, 1 A
$l_3 \leftarrow k_6 \cdot k_7$; $l_2 \leftarrow k_4 \cdot k_5$	2 M
$l_0 \leftarrow k_0 \cdot k_2$; $n_0 \leftarrow k_0 + k_1$	1 M, 1 A
$l_1 \leftarrow k_1 \cdot k_3$; $n_1 \leftarrow k_2 + k_3$	1 M, 1 A
$p_0 \leftarrow n_0 \cdot n_1$; $q_0 \leftarrow m_0 + l_1$	1 M, 1 A
$q_1 \leftarrow q_0 + l_0$; $r_1 \leftarrow m_2 + q_1$	2 A
$q_3 \leftarrow m_1 + m_2$; $r_3' \leftarrow m_3 + p_0$	2 A
$r_3'' \leftarrow r_3' + q_1$; $r_3 \leftarrow r_3'' + l_3$	2 A
$r_2' \leftarrow l_3 + q_0$; $r_2'' \leftarrow r_2' + l_2$	2 A
$r_2 \leftarrow r_2'' + q_3$; $r_0' \leftarrow m_3 + q_3$	2 A
$r_0 \leftarrow r_0' + m_0$	1 A

2.5.4 Geheugen

Nu alle nodige algoritmes en de daarbij horende geheugenvereisten gekend zijn, is het mogelijk om het geheugenblok van de schakeling te ontwerpen. Eerst zal het minimum aantal benodigde registers bepaald worden en vervolgens zal een ontwerp voor het geheugenblok voorgesteld worden.

Minimum aantal registers

De geheugenvereisten van de for-lus en de finale machtsverheffing kunnen los van elkaar bekeken worden. Eens de lus beëindigd is, is het immers niet meer nodig $P, \phi(Q), V$ en G bij te houden.

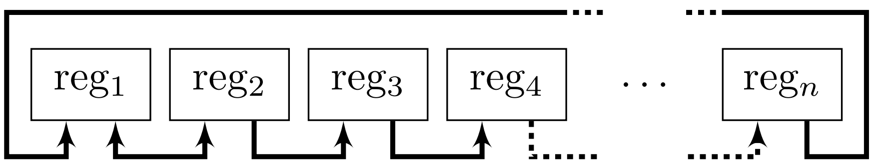
Van de vijf algoritmes die opgeroepen worden in de for-lus, vereist de de vermenigvuldiging van F en G het grootste aantal tijdelijke registers, namelijk drie. Het minimum aantal registers om alle bewerkingen in de lus te kunnen uitvoeren is dus vijftien: zes voor $P, \phi(Q)$ en V, twee voor G, vier voor F en drie tijdelijke registers.

Bij de berekening van de finale machtsverheffing gaat de vermenigvuldiging van I en J met de titel van 'meest geheugenvereistende' algoritme lopen. In dit geval zijn er opnieuw vijftien registers nodig: acht voor I en J en zeven miljard voor de tussenresultaten.

Het minimum aantal registers n is dus vijftien. In totaal bevat de complete schakeling zestien registers van grootte m: vijftien in de controller voor het Miller algoritme en één in de $\mathbb{F}_{2^m}$ kern.

Ontwerp van geheugenblok

Voor het ontwerp van het geheugenblok wordt RAM al op voorhand verworpen gezien dit zowel groot is als veel vermogen verbruikt. Er wordt verder gebouwd op de ideeën aangebracht in [12]. Daarin wordt een circulair ontwerp voorgesteld dat geïllustreerd wordt in Figuur 2.7.



Figuur 2.7: Circulair registerblok ontwerp en mogelijke operaties

Elk van de registers kan enkel de waarde van zijn voorganger aannemen (de zogenaamde *shift* operatie). Tevens kunnen de waarden van eerste twee registers omgewisseld worden (*swap*) en kan de waarde van register twee naar register één gekopieerd worden (*copy*). De ingangen van de aritmetische schakeling (de $\mathbb{F}_{2^m}$ kern in dit geval) zijn rechtstreeks verbonden met registers één en twee. Aan register één zijn tevens ook de uitgang van de aritmetische schakeling en de ingang van de volledige schakeling aangesloten. De enige manier om nieuwe waarden in de registers op te slaan, is dus via de aanpassing van de waarde van register één.

Het voordeel van zo'n ontwerp ten opzichte van RAM is de veel kleinere oppervlakte. Bij RAM stijgt de oppervlakte van de benodigde multiplexers kwadratisch met het aantal registers. Bij een ontwerp van dit type is de oppervlakte van de multiplexers echter constant met als resultaat een veel kleinere implementatie.

Omdat de ingangen van de onderliggende $\mathbb{F}_{2^m}$ kern rechtstreeks verbonden zijn met twee registers, is het noodzakelijk om voor elke bewerking de nodige waarden naar die registers over te brengen. Het gemiddeld aantal klokslagen $\bar{t}$ dat daar voor nodig is, kan op de manier die volgt bepaald worden. Eerst wordt de gemiddelde afstand $\bar{r}$ tussen twee variabelen x_0 en x_1 bepaald. Die is gelijk aan de som s van de mogelijke afstanden r gedeeld door het aantal mogelijke posities c die twee variabelen kunnen bezetten:

$$\begin{aligned} s &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (j-i-1) & c &= \sum_{i=0}^{n-1} i \\ &= \frac{n \cdot (n-1) \cdot (n-2)}{6} & &= n \cdot \frac{n-1}{2} \end{aligned}$$

dus

$$\bar{r} = \frac{n-2}{3}.$$

Merk op dat deze formules slechts een benadering geven. Zo wordt bijvoorbeeld niet in rekening gebracht dat wanneer beide waarden in de eerste twee registers zitten, er geen doorschuivingen moeten gebeuren.

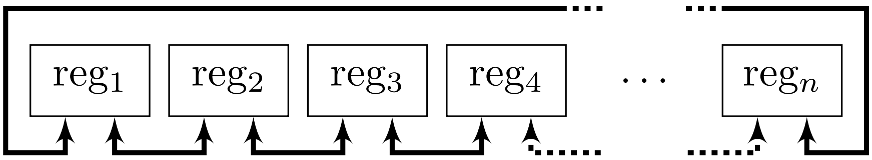
Per positie die x_0 en x_1 van elkaar verwijderd zijn, zal een *swap* operatie uitgevoerd moeten. Daartoe dient x_1 telkens eerst verplaatst te worden tot register twee. Er zijn dus $\bar{r} \cdot n$ doorschuif operaties nodig. Dit is opnieuw een ruwe schatting, die voor kleine n niet zal kloppen. Algemeen kan echter gesteld worden dat

$$\bar{t} \in \mathcal{O}\Big(\frac{n^2}{3}\Big).$$

Verder is het energieverbruik recht evenredig met het aantal schrijfbewerkingen die op de registers worden uitgevoerd. Elke doorschuif operatie kost n zulke bewerkingen, m.a.w. het gemiddeld aantal schrijfbewerkingen $\overline{w}$ dat uitgevoerd moet worden voor gestart kan worden met de berekening van een optelling of vermenigvuldiging, is

$$\overline{w} \in \mathcal{O}\Big(\frac{n^3}{3}\Big).$$

Gezien het feit dat er vijftien registers zijn (tegenover zes in [12]), zou dit zeer veel energie kosten. Verder zou het veel klokslagen in beslag nemen om beide waarden in de correcte registers op te slaan, hoewel zoals eerder bepaald tijdsduur niet van zeer groot belang is. Om dit probleem op te lossen, wordt het mogelijk gemaakt dat elk register ook de waarde van zijn voorganger kan opslaan. Dit is equivalent aan de *swap* (en *copy*) operatie toelaten tussen alle aan elkaar grenzende registers. Verder kunnen *swap* operaties tussen twee paar registers in parallel uitgevoerd worden. Uiteraard wordt hiervoor een prijs betaald, er moet nu namelijk per register een extra multiplexer en een selectie signaal voorzien worden. Het resulterend ontwerp is te zien in Figuur 2.8.



Figuur 2.8: Circulair registerblok geoptimaliseerd voor energieverbruik

In dit geval zal de afstand r van een waarde x tot het eerste register gelijk zijn aan $\mathsf{min}(j-1, n-j+1)$ omdat nu in beide richtingen geschoven kan worden. De gemiddelde afstand $\overline{r}$ is dan

$$\begin{aligned}\overline{r} &= \frac{1}{n} \cdot \sum_{i=1}^{n} \mathsf{min}(j-1, n-j+1) \\ &= \frac{2}{n} \cdot \sum_{i=1}^{\frac{n}{2}} (j-1) \\ &= \frac{n-1}{4}.\end{aligned}$$

Aangezien de omwissel operaties in parallel uitgevoerd kunnen worden, is het gemiddeld aantal klokslagen in dit geval

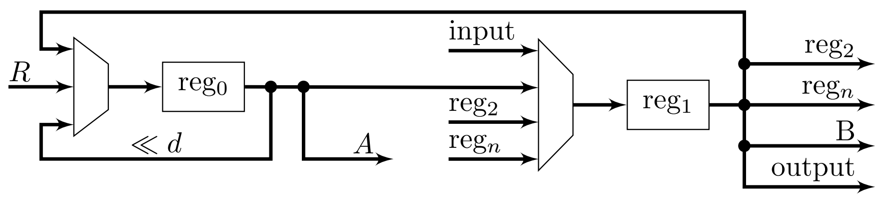
$$\overline{t} \in \mathcal{O}\Big(\frac{n}{4}\Big)$$

en, rekening houdend met het feit dat elke omwissel operatie twee schrijfbewerkingen vraagt en er twee variabelen naar de juiste positie gebracht moeten worden,

$$\overline{w} \in \mathcal{O}(n).$$

Tegenover het originele ontwerp voor het registerblok, zal dit ontwerp veel minder klokslagen verloren laten gaan aan het in de juiste positie brengen van de benodigde variabelen. Ook zal het, ten koste van meer multiplexers, significant minder energie verbruiken. De extra multiplexers zullen echter ook energie vragen, dus in hoeverre dit het totale verbruik naar beneden haalt, moet van implementatie tot implementatie bekeken worden.

Registers één en twee worden respectievelijk met de A en B ingang van de $\mathbb{F}_{2^m}$ kern verbonden. Om vermenigvuldigingen tot een succesvol einde te brengen, moet het register dat met de A ingang verbonden is elke klokslag met d bits naar links doorgeschoven kunnen worden. Hierbij gaat de originele waarde van het register echter wel verloren. Wanneer de algoritmes die het meeste tijdelijke opslag vragen echter nauwkeurig worden bekeken, zal men merken dat dit geen probleem vormt. Vandaar dat er voor gekozen wordt om het derde tijdelijk register uit de doorschuif lus te halen, er moeten dan minder doorschuivingen doorgevoerd worden om elke waarde in de juiste positie te brengen. Verder moet het ook mogelijk zijn waarden die op de ingang van de Miller controller aangelegd worden op te slaan en het resultaat op de uitgang te zetten. Ten slotte moet het resultaat van een berekening in $\mathbb{F}_{2^m}$ opgeslagen kunnen worden. Teneinde dat mogelijk te maken, wordt de uitgang R van de $\mathbb{F}_{2^m}$ kern aan het derde tijdelijk register gekoppeld. Rekend houdend met deze zaken, wordt het ontwerp van de eerste twee registers aangepast zoals te zien in Figuur 2.9.

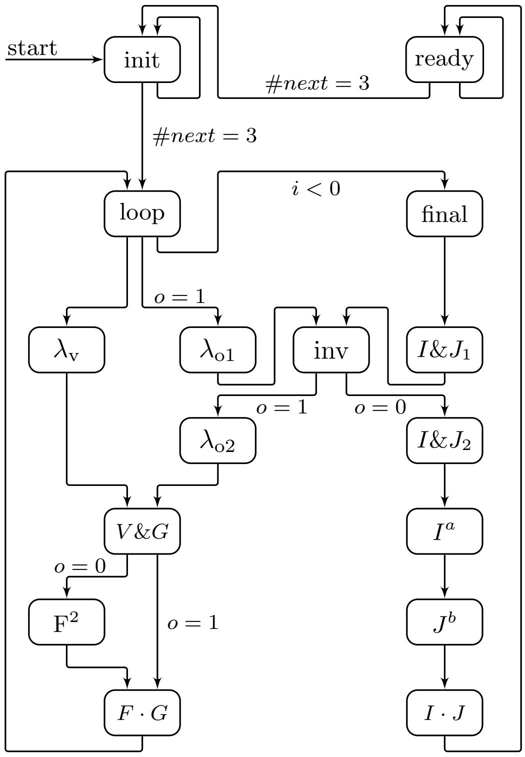


Figuur 2.9: Ontwerp van de schakeling rond de eerste twee registers in het registerblok

2.5.5 FSM

Met alle details over de hardware bekend, kan overgegaan worden tot het ontwerp van een FSM. Daarbij zal een zeer groot deel van de te implementeren states voor niets anders dienen dan het verschuiven van geheugen. Het uiteindelijke resultaat bevat 553 states. Om het geheel overzichtelijk te houden, zal het ontwerp dan ook niet tot op individueel state niveau gebeuren. Ook zal zoveel mogelijk getracht worden reeds geïmplementeerde states opnieuw te gebruiken.

Aangezien een beeld meer zegt dan duizend woorden, wordt het resultaat zonder veel extra uitleg gepresenteerd in Figuur 2.10. Wat wel enige extra aandacht verdient, is de opsplitsing van de verdubbel- en optelstap. Wanneer Algoritme 2.4 en 2.5 opnieuw bekeken worden, valt het op dat buiten de berekening van λ en het nieuwe x-coördinaat van V alle andere berekeningen zeer gelijkaardig zijn. Meer nog, indien de paren (x_V, y_V) en (x_P, y_P) vervangen worden door een algemene (x_A, y_A), dan bestaan beide algoritmes uit dezelfde bewerkingen na het berekenen van λ en x_V. Aangezien na de berekening van λ nog steeds twee registers vrij zijn voor gebruik, kunnen deze aangewend worden om de toepasselijke x en y coördinaten in op te slaan. Op die manier kunnen de verdere berekeningen via dezelfde states uitgewerkt worden, ongeacht of nu de verdubbel- of de optelstap uitgevoerd wordt.



Figuur 2.10: FSM ontwerp van de controller voor het Miller algoritme

2.6 Optimalisaties

Nu de schakeling volledig ontworpen is, kan overgegaan worden tot optimalisering. Daarbij wordt opnieuw op de eerste plaats getracht de oppervlakte kleiner te maken, maar er zal ook aandacht gegeven worden aan het beperken van het energieverbruik van de schakeling. De optimalisaties die in de volgende paragrafen voorgesteld worden, zullen allemaal iets te maken hebben met de registers. Omdat de enkele registers van grootte 1 of $\log_2(m)$ bit niet veel bijdragen aan noch de uiteindelijke oppervlakte noch het verbruik zullen de optimalisaties specifiek gericht zijn op het efficiënter maken van het geheugenblok in de controller voor de Miller loop.

Een register is doorgaans opgebouwd uit een hoeveelheid D-type master-slave flip-flops. Dit type flip-flops is opgebouwd uit twee latches. De eerst latch (master) is verbonden met de ingang D en slaat de waarde daarvan op zolang CLK laag is. De tweede latch (slave) is verbonden met de uitgang Q en neemt de waarde van de eerste latch over eens CLK hoog is. Het effect is dus dat de uitgang Q op een stijgende klokflank de waarde van D overneemt. Er bestaan verschillende gespecialiseerde types: met reset ingang, met enable ingang, met test ingang, ... In de paragrafen die volgen, wordt er van uit gegaan dat alle registers uit dit type flip-flops zijn opgebouwd.

2.6.1 Registers zonder reset

Een makkelijke eerste aanpassing is het verwijderen van de reset ingangen van de registers. Zoals reeds gezien in Tabel 2.2 kost een D flip-flop zonder reset ingang 0.5 gates minder per bit dan één met, een verkleining van 8.5%.

In het geval van $m = 163$ kijkt men dan aan tegen een besparing van $978 - 896.5 = 81.5$ gates per register. Merk echter wel op dat ten minste één register moet overblijven dat wel op 0 (en 1) ingesteld kan worden om F in te stellen aan het begin van het algoritme.

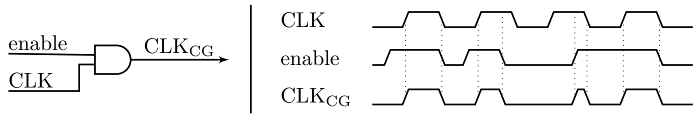
2.6.2 Clock gating

Normaal wordt een register elke klokslag geladen met de waarde aan zijn ingang. Het is dus noodzakelijk het register naar zichzelf terug te koppelen, zodat het mogelijk is reeds opgeslagen waarde in het register te bewaren. Een techniek, genaamd clock gating, laat toe deze terugkoppeling (en de daarbij horende multiplexer) achterwege te laten. Bij deze techniek wordt het kloksignaal enkel gepropageerd naar een register wanneer een daarbij horende enable ingang hoog is. Zolang het enable signaal laag gehouden wordt, blijft de waarde van het register dus constant.

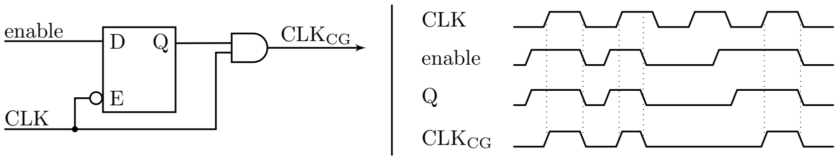
Deze techniek laat ook toe het verbruik van de schakeling drastisch te verlagen. Er zullen immers veel minder interne poorten zijn (bv. in de multiplexer) die elke klokslag van waarde dienen te veranderen.

De eenvoudigste schakeling waarmee clock gating geïmplementeerd kan worden is te zien in Figuur 2.11. Er zijn echter enkele problemen geassocieerd met dit type schakeling. De uitgang is immers onderhevig aan onregelmatigheden op het enable signaal. Stel bijvoorbeeld dat de enable ingang al hoog wordt terwijl het kloksignaal ook nog hoog is. Dan zal het kloksignaal gepropageerd worden tot het register, wat niet de bedoeling is.

Dit probleem kan overkomen worden met de schakeling voorgesteld in Figuur 2.12. De latch zorgt er hier voor dat het enable signaal pas wordt doorgelaten nadat het kloksignaal laag geweest is. Hierdoor worden mogelijke onregelmatigheden dus tegengehouden voor de ze klokingang van het register kunnen bereiken.

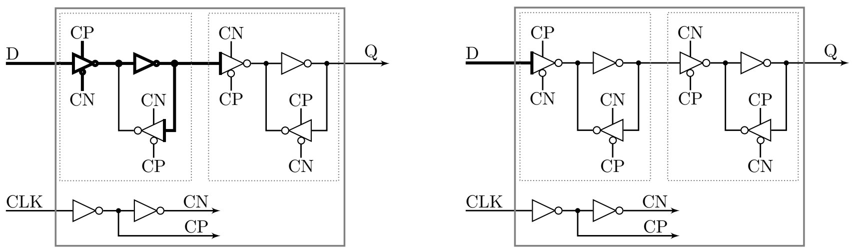


Figuur 2.11: Schakeling voor clock gating - Basis ontwerp



Figuur 2.12: Schakeling voor clock gating - Verbeterd ontwerp

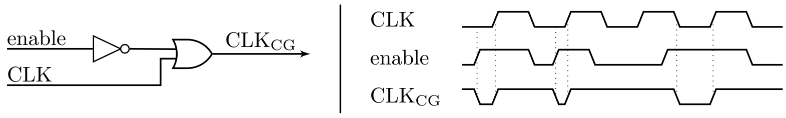
Ten slotte is er een derde oplossing die toelaat een nog grotere energiebesparing door te voeren, zoals aangetoond in [13]. In die paper wordt bewezen dat het energie verbruik van een D-type master-slave flip-flop veel hoger is wanneer het kloksignaal laag is, dan wanneer het hoog is. De reden hiervan is duidelijk te zien in Figuur 2.13: wanneer de klok laag is, veranderen telkens de ingang verandert twee interne poorten van staat en wijzigen de gate capaciteiten van twee andere poorten. Indien de klokingang hoog is, wijzigt een veranderend ingangssignaal enkel de gate capaciteit van de eerste interne poort.



(a) $CLK = 0$ (b) $CLK = 1$

Figuur 2.13: Stroomverbruikende componenten van een D-type master-slave flip-flop bij constante waarde van de klok ingang [13]

De werking van de twee vorige schakelingen is zo dat de klokingang laag gehouden wordt zolang het enable signaal laag is. De oplossing ligt in de schakeling in Figuur 2.14, die exact doet wat nodig is: de klokingang hoog houden zolang geen nieuwe waarde moet opgeslagen worden. Om het voorkomen van onregelmatigheden te garanderen, moet het enable signaal stabiel worden voor het CLK signaal hoog wordt. Indien bepaalde kloksignaalbeperkingen worden opgelegd bij het synthetiseren van het circuit is dit normaal gezien echter geen probleem.



Figuur 2.14: Schakeling voor clock gating - Laag vermogen ontwerp

Hoofdstuk 3

Resultaten

In dit hoofdstuk zullen verscheidene ASIC implementaties van de schakeling, die in Hoofdstuk 2 beschreven werd, van naderbij bestudeerd worden. Daarbij zal gelet worden op de oppervlakte en het verbruik van het resultaat. Er zal onderzocht worden wat het effect van de verschillende voorgestelde optimalisaties is op deze parameters. Ten slotte zal het ontwerp vergeleken worden met reeds bestaande implementaties.

3.1 Gebruikte software

Het ontwerp werd geprogrammeerd in GEZEL [14]. Simulaties en compilatie naar VHDL werden uitgevoerd met GEZEL 2.0. De optimalisaties werden manueel uitgevoerd in de VHDL code, aangezien dit in GEZEL niet mogelijk was.

Alle ontwerpen werden gesynthetiseerd met behulp van Synopsys Design Vision (versie Y-2006.06). De gebruikte bibliotheek was de $0.13\mu m$ *low leakage* bibliotheek van Faraday Technology [4]. De maximale oppervlakte werd ingesteld op nul, wat als netto effect een resultaat met minimum oppervlakte gaf. Verder werd voor het kloksignaal, indien niet anders vermeld, een frequentie van 10kHz gebruikt.

3.2 Gerapporteerd verbruik

Voor de resultaten in verband met energieverbruik worden steeds twee waarden gegeven. De eerste waarde, dynamisch verbruik, geeft weer hoeveel vermogen verbruikt wordt door veranderende CMOS in- en uitgangen. De tweede waarde, leakage verbruik, is verbruik dat voorkomt zelfs indien een transistor niet geleidt. De impact hiervan hangt onder meer af van de gebruikte bibliotheek. Het is voor het synthese programma zeer moeilijk een nauwkeurige schatting voor het verbruik te geven, dus de gegeven waarden moeten niet als exact worden beschouwd.

Indien gewenst kan het verbruik voor hogere kloksnelheiden geschat worden. Gegeven de standaard frequentie $f = 10$kHz, de formule voor het dynamisch verbruik van een CMOS schakeling:

$$P_d = V^2 \cdot C \cdot f$$

en het leakage verbruik P_l, kan men het totale verbruik omrekenen naar dat van een willekeurige kloksnelheid via

$$P' = \frac{P_d \cdot f'}{10 \cdot 10^3} + P_l.$$

Het verkregen resultaat zal niet volledig zal kloppen, maar komt zeer dicht in de buurt van het door Design Vision geschatte verbruik.

3.3 Benodigde rekentijd

Alvorens de syntheseresultaten te presenteren, wordt even uitgewijd over het noodzakelijk aantal klokcycli c om één pairing te berekenen. Merk op dat de formules die volgen enkel kloppen voor willekeurige m indien $\mathsf{Hamm}(l) = 3$, wat het geval is voor de gekozen parameters (zie Paragraaf 2.1).

Het totaal aantal cycli wordt gegeven door:

$$c = 21681 + 4322 + 2998 \cdot \left\lceil \frac{m}{d} \right\rceil.$$

Daarbij staat het eerste getal voor het aantal klokcycli waarin geheugen van plaats verschoven wordt. Het tweede getal is het aantal optellingen dat uitgevoerd dient te worden en de coëfficient voor de vermenigvuldiging is het aantal vermenigvuldigingen. Het aantal klokcycli c kan verder opgesplitst worden in het aantal cycli c_f voor de for-lus en c_m voor de finale machtsverheffing. In dat geval zijn de formules:

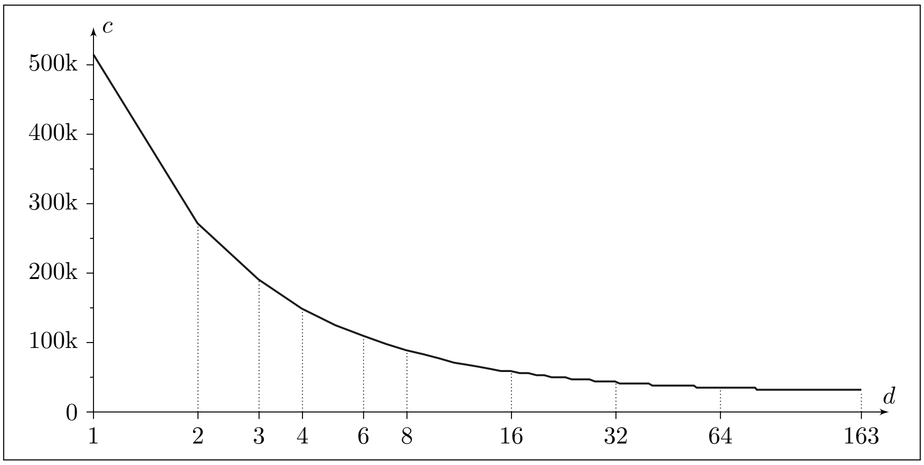
$$c_f = 18731 + 3937 + 2463 \cdot \left\lceil \frac{m}{d} \right\rceil$$
$$c_m = 2950 + 385 + 535 \cdot \left\lceil \frac{m}{d} \right\rceil,$$

waarbij de coëfficienten opnieuw voor respectievelijk geheugenverschuivingen, optellingen en vermenigvuldigingen staan.

In Tabel 3.1 wordt voor enkele waarden getoond hoeveel klokcycli nodig zijn om een berekening te voltooien in het geval $m = 163$. In Figuur 3.1 wordt hetzelfde weergegeven, maar dan voor elke d van 1 t.e.m. 163. Het is duidelijk dat de tijdsbesparing waar extra MALU's voor zorgen vrij snel teniet wordt gedaan door het aantal cycli dat niet door d beïnvloed wordt.

Tabel 3.1: Aantal klokcycli c nodig voor één pairing i.f.v. het aantal MALU's d voor $m = 163$

d	1	2	3	4	6	8	16	32
c	514 677	271 839	190 893	148 921	109 947	88 961	58 981	43 991



Figuur 3.1: Aantal klokcycli c nodig voor één pairing i.f.v. het aantal MALU's d voor $m = 163$

3.4 Basisimplementatie & register optimalisaties

Logischerwijs wordt eerst gekeken naar de syntheseresultaten voor een implementatie met één MALU. Verder zal in deze paragraaf ook onderzocht worden wat de effecten zijn van de optimalisaties voorgesteld in Paragraaf 2.6. Bij de implementaties met clock gating werden steeds ook de reset ingangen van zoveel mogelijk registers verwijderd. Ten slotte zal onderzocht worden hoeveel oppervlakte de individuele onderdelen van het ontwerp innemen.

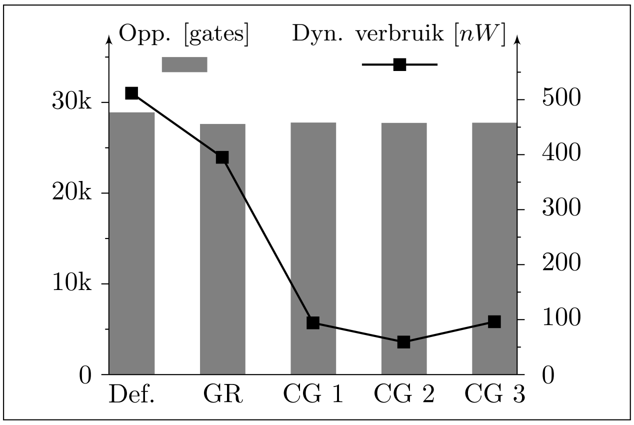
De synthese resultaten voor de vijf verschillende implementaties worden gegeven in Tabel 3.2. De versies met clock gating (CG n) implementeren de schakelingen in de volgorde waarin ze voorkomen in Paragraaf 2.6.2. Ter verduidelijking zijn deze resultaten ook nog eens uitgezet in Figuur 3.2.

Zoals verwacht resulteert het toepassen van clock gating in een zeer grote besparing op het verbruik. Het is zelfs zo dat na toepassing van clock gating het dynamisch verbruik lager is dan het leakage verbruik. Daarbij dient opgemerkt te worden dat het dynamisch verbruik zal stijgen bij een hogere kloksnelheid terwijl het leakage verbruik constant zal blijven.

Het is opmerkelijk dat het dynamisch verbruik van CG 2 zo veel lager wordt gerapporteerd dan dat van de andere twee CG implementaties. Dit mogelijk als te verklaren door het feit dat de synthese software in de CG 2

Tabel 3.2: Syntheseresultaten voor implementaties met één MALU

Ontwerp	Opp. [gates]		Verbruik @ 10kHz [nW]			
			Dynamisch		Leakage	
Basis	28 876		512		117	
Geen Reset	27 596	96%	395	77%	107	92%
CG 1	27 751	96%	94	18%	109	94%
CG 2	27 713	96%	59	12%	102	88%
CG 3	27 734	96%	96	19%	110	94%



Figuur 3.2: Syntheseresultaten voor implementaties met één MALU en $f =$ 10kHz

implementatie de clock gating schakeling herkent, aangezien dit de de facto manier is om clock gating toe te passen. Waarschijnlijk wordt daar dus bij de berekening van het verbruik ook rekening mee gehouden. In werkelijkheid zal implementatie CG 3 het minste verbruiken.

Verder valt het ook op dat de implementaties met clock gating niet kleiner zijn als die zonder. Normaal gezien zou de toepassing van deze schakeling per register één multiplexer ingang moeten elemineren. Dat dit niet gebeurd is opnieuw te wijten aan de synthese software die niet doorheeft dat er clock gating wordt toegepast en het register dus sowieso naar zichzelf terugkoppelt. Hoe het dan komt dat het verbruik van CG 2 toch lager wordt gerapporteerd is niet helemaal duidelijk.

Ten slotte wordt nog ontleed hoeveel oppervlakte de individuele onderdelen van de schakeling innemen. Tabel 3.3 geeft een overzicht van de grootte van de verschillende delen van implementatie CG 3. Het valt op dat de zestien registers en de FSM, die bestaat uit 553 states, samen goed zijn voor 95.5% van de oppervlakte. Het effect daarvan zal duidelijk worden in de volgende paragraaf, wanneer meerdere MALU's geïmplementeerd worden.

Tabel 3.3: Oppervlakte van de individuele onderdelen in de implementatie met één MALU

Onderdeel	Opp.	[gates]
MALU	458	1.7%
$\mathbb{F}_{2^m}$ kern		
Logica	783	2.8%
Registers	962	3.5%
Controller		
Logica	13 044	47%
Registers	12 487	45%
Totaal	27 734	100%

3.5 Meerdere MALU's

Mits de toevoeging van extra MALU's is het, zoals aangetoond, mogelijk de totale rekeningtijd drastisch te verlagen. Hoewel het gebruik van meerdere MALU's de uiteindelijke schakeling vergroot en dat dus enigszins in gaat tegen de originele doelstelling, wordt hier toch onderzocht in welke mate de interessante parameters hierdoor juist worden beïnvloed.

Op de implementaties met meerdere MALU's werd steeds de derde clock gating techniek (en het verwijderen van reset ingangen) toegepast, aangezien dit in werkelijkheid de grootste energiebesparing teweeg zou moeten brengen. Implementaties met een aantal MALU's gaande van twee t.e.m. tweeëndertig werden gesynthetiseerd. Een nog hoger aantal MALU's zou immers compleet ingaan tegen de originele doelstelling.

De resultaten van de synthese zijn te zien in Tabel 3.4 en Figuur 3.3. Het valt op dat, zoals verwacht, het toevoegen van een klein aantal extra MALU's weinig extra oppervlakte en verbruik met zich meebrengt en dus een aangeraden manier is om de snelheid van de schakeling op te drijven. Dit wordt verder onderzocht in de volgende paragraaf.

3.6 Hogere kloksnelheid vs. meerdere MALU's

Aan de gegeven kloksnelheid $f = 10\text{kHz}$ doet een schakeling met één MALU er 51.5 seconden over om één pairing te berekenen. Dit zal in de meeste gevallen onaanvaardbaar zijn. Daarom wordt hier onderzocht wat de effecten op de schakeling zijn indien de kloksnelheid wordt opgedreven en er eventueel meerdere MALU's gebruikt worden. Ter illustratie zal voor een im-

Tabel 3.4: Syntheseresultaten voor implementaties met d MALU's

d	Opp. [gates]		Verbruik @ 10kHz [nW]				Tijds-winst
			Dynamisch		Leakage		
1	27 734		96		110		
2	28 423	102%	90	94%	113	103%	47.2%
3	29 071	105%	103	107%	118	107%	62.9%
4	30 278	109%	108	113%	122	111%	71.1%
6	30 956	112%	112	117%	127	115%	78.6%
8	32 782	118%	122	127%	136	124%	82.7%
16	37 798	136%	162	169%	163	148%	88.5%
32	47 833	172%	212	221%	213	194%	91.5%

plementatie met meerdere MALU's die met twee nader onderzocht worden. Opnieuw wordt in beide gevallen clock gating schakeling drie gebruikt en worden zoveel mogelijk reset ingangen verwijderd.

Stel een maximale rekentijd $t_{\max} \approx 50ms$. Voor een implementatie met één MALU moet de klokfrequentie f_1 dan 1030 maal verhoogd worden. Wanneer men de schakeling met twee MALU's even snel wenst te maken als die met één dan dient de kloksnelheid f_2 van die eerste vermenigvuldigd te worden met $\Delta f = 1 - 0.472$. De kloksnelheden van de respectievelijke schakelingen zijn dan:

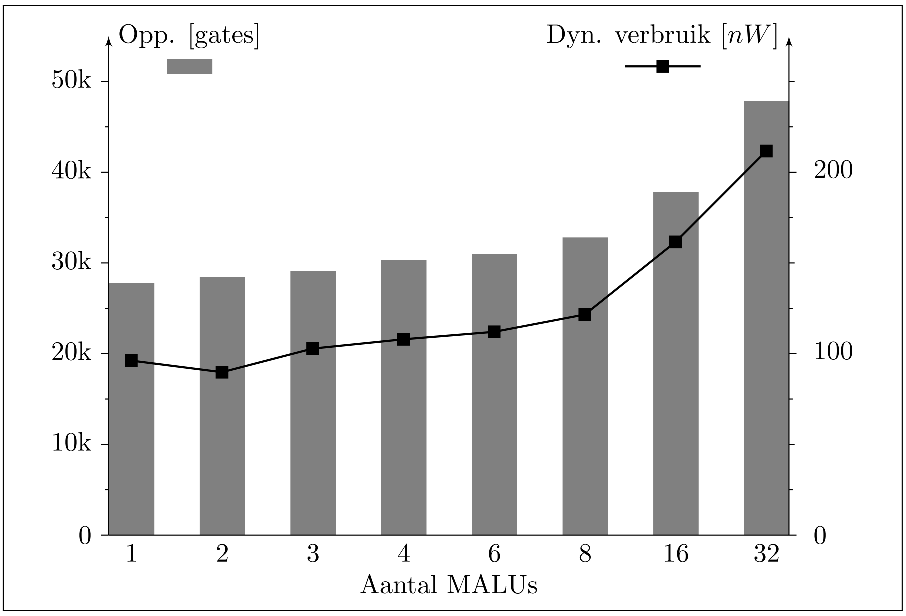
$$f_1 \approx 10.3\text{MHz}$$
$$f_2 \approx 5.44\text{MHz}.$$

Ter vergelijking worden de resulterende parameters van beide implementaties gegeven in Tabel 3.5. Beiden werden gehersynthetiseerd met aangepaste parameters voor de klok. Dat de schakeling met één MALU in dit geval kleiner is dan in het geval $f = 10$kHz, is te wijten aan enige willekeurigheid in de algoritmes van de synthesesoftware.

Hoewel de implementatie met twee MALU's 3% groter is dan die met één, verbruikt ze slechts de helft zoveel vermogen. Het lijkt dus vanzelfsprekend om steeds voor een implementatie met meerdere MALU's te kiezen indien de snelheid moet opgedreven worden. Hoeveel MALU's ideaal zijn, zal afhankelijk zijn van toepassing tot toepassing.

3.7 Vergelijking met bestaande implementaties

Gezien de vrij recente ontdekking van pairings is het beschikbare aantal implementaties voor ASICs ook vrij beperkt. Er werden slechts drie papers in de literatuur gevonden waarin het voorgestelde ontwerp naar een ASIC gesynthetiseerd werd. Enkele papers beschrijven een ontwerp voor



Figuur 3.3: Syntheseresultaten voor implementaties met meerdere MALU's en $f = 10$kHz

gebruik in sensor netwerken. Helaas hebben deze als doelplatform allemaal een microprocessor. Zowat alle andere gepubliceerde implementaties werden ontwikkeld in software of voor FPGA's. Verder wordt in het geval van de FPGA ontwerpen, waar een zekere controle over de uiteindelijke grootte mogelijk is, zowat steeds geoptimaliseerd naar een minimum oppervlakte rekentijd product. Voor laag vermogensschakeling is dit echter geen interessante waarde. Het is dus zowat onmogelijk een grondige vergelijking te maken tussen de verschillende bestaande implementaties. Toch zal zo goed als mogelijk getracht worden een overzicht te geven, zodat het in deze thesis voorgestelde ontwerp beter geplaatst kan worden.

3.7.1 Microchip implementaties

Implementaties van pairings voor gebruik op een MICA node, specifiek ontwikkeld voor gebruik in sensor netwerken, worden voorgesteld in [15], [16] en [17]. De processor op deze nodes is een ATMega128L microchip. Een overzicht van de resultaten is gegeven in Tabel 3.6. Rekening houdend met het stroomverbruik gegeven in [17] wordt het verbruik geschat op ongeveer $23.60mW$. Uiteraard zijn de oppervlakte en het verbruik van een microchip implementatie niet te vergelijken met de in deze thesis voorgestelde ASIC implementatie, gezien de zeer verschillende architectuur van en filosofie achter de keuze voor beiden.

Tabel 3.5: Vergelijking van syntheseresultaten voor twee verschillende implementaties die er even lang over doen één pairing te berekenen

	1 MALU	2 MALU's	
f [MHz]	10.3	5.44	53%
Opp. [gates]	27 430	28 155	103%
Verbruik [μW]			
Dynamisch	98.2	48.5	49%
Leakage	$107 \cdot 10^{-3}$	$111 \cdot 10^{-3}$	104%

Tabel 3.6: Resultaten uit de literatuur voor implementaties met focus op sensor netwerken

	TinyTate [15]	TinyPBC [16]	NanoECC [17]	
			Binair	Priem
Veld	$\mathbb{F}_p$ 256 bit	$\mathbb{F}_{2^{271}}$	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_p$ 160 bit
Pairing	Tate	η_T	Tate	Tate
Rekentijd (s)	30.21	5.45	10.96	17.93

3.7.2 FPGA implementaties

In de literatuur zijn vrij veel ontwerpen voor FPGA's terug te vinden. Het probleem is echter dat men zich bij het ontwerp hiervan steeds toelegt op het behalen van een zo hoog mogelijke snelheid, wat resulteert in een grote oppervlakte. Ook wordt vaak gerekend over grotere velden dan hetgeen waarover gerekend wordt in deze thesis (bv. $\mathbb{F}_{2^{313}}$ of $\mathbb{F}_{3^{197}}$), wat meer veiligheid biedt, maar resulteert in nog grotere ontwerpen.

Grootte van een FPGA ontwerp wordt vermeld in *slices* (Xilinx) of *logic elements* (Altera). Deze eenheden zijn onmogelijk om te zetten naar een aantal gates. Zodoende is het dus niet mogelijk de grootte van deze ontwerpen te vergelijken met die van een ASIC schakeling.

Toch wordt in Tabel 3.7 een sumier overzicht gegeven van een zeer beperkt aantal ontwerpen. Bij de selectie hiervan werd vooral gekozen voor ontwerpen waarin in een vrij klein veld gerekend werd. Er dient in acht te worden genomen dat bij al deze implementaties snelheid, en niet oppervlakte, het voornaamste doel is. Het gebruik van een vermenigvuldigingsschakeling equivalent aan het gebruik van honderd of meer MALU's is in deze gevallen eerder regel dan uitzondering. Ook worden steeds meerdere berekeningen in parallel uitgevoerd.

Tabel 3.7: Resultaten uit de literatuur voor implementaties ontwikkeld voor FPGA's

	Veld	Pairing	Opp. [slices]	f [MHz]	Reken-tijd [μs]
Ronan *et al.* [18]	$\mathbb{F}_{2^{103}}$	Mod. Tate	21021	51	206
Shu *et al.* [19]	$\mathbb{F}_{2^{239}}$	Mod. Tate	25287	84	41
Keller *et al.* [20]	$\mathbb{F}_{2^{251}}$	Mod. Tate	27725	40	2370
Grabher en Page [21]	$\mathbb{F}_{3^{97}}$	Mod. Tate	4481	150	432
Beuchat *et al.* [22]	$\mathbb{F}_{3^{97}}$	η_T	1833	145	192

3.7.3 ASIC implementaties

Ten slotte rest dan nog de vergelijking met de tot nu toe gepubliceerde ASIC ontwerpen. De drie ontwerpen worden voorgesteld in respectievelijk [23], [24] en [25]. Zowel in [23] als [25] wordt met het oog op het behalen van zo hoog mogelijke snelheden ontworpen. De implementatie uit [24] bevat naast de schakeling voor pairings tevens een RISC processor. In [25] wordt de finale machtsverheffing niet uitgevoerd. Tabel 3.8 geeft een vergelijkend overzicht van het ontwerp voorgesteld in deze thesis en de vermelde ontwerpen. Aangezien het ontwerp uit [23] het enige is waarvan alle gegevens bekend zijn, zal in de tekst die volgt enkel met dat ontwerp vergeleken worden.

Tabel 3.8: Vergelijking van de implementatie voorgesteld in deze thesis met ASIC implementaties uit de literatuur

	Thesis (1 MALU, CG 3)		Pairing-Lite [23]	Kammler *et al.* [24]	Kömürcü en Savas [25]
	$d = 1$, $f = 10$kHz	$d = 2$, $f = 5.44$MHz			
Veld	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_p$ 256 bit	$\mathbb{F}_{3^{97}}$
Pairing	Tate	Tate	η_T	Optimal Ate	Tate
Technologie	$0.13\mu m$	$0.13\mu m$	$0.18\mu m$	$0.13\mu m$	$0.25\mu m$
Opp. [gates]	27 734	28 155	193 765	97 000	$10mm^{2\dagger}$
f [MHz]	0.01	5.44	200	338	78
Rekentijd [μs]	$51.5 \cdot 10^6$	$50 \cdot 10^3$	46.7	$15.8 \cdot 10^3$	$250^\ddagger$
Verbruik [mW]	$206 \cdot 10^{-6}$	$48.6 \cdot 10^{-3}$	672	?	?
Score $\left[\frac{\text{fJ}}{\text{gate}}\right]^\S$	0.74	0.32	17.34	?	?

† De gegeven oppervlakte is die van de complete schakeling inclusief routing.
‡ Exclusief de finale machtsverheffing.
§ Des te lager de score, des te conservatiever springt de implementatie om met energie. De score wordt berekend als score $= \frac{\text{verbruik}}{\text{frequentie} \cdot \text{oppervlakte}}$.

De hier voorgestelde implementaties zijn niet alleen ongeveer een factor zeven kleiner dan dat voorgesteld in [23], hun verbruik per gate (genormaliseerd voor frequentie) is ook beduidend lager. Uiteraard moet er wel op gewezen worden dat het ontwerp uit [23] door het gebruik van een groter veld een grotere veiligheid biedt en, zoals reeds vermeld, niet ontworpen is met compactheid of laag verbruik in gedachten. Het is dus zeker niet de bedoeling het ontwerp af te breken. Het is echter wel duidelijk dat de ontwerpen voorgesteld in deze thesis met kop en schouders uitsteken boven dat van [23] indien compactheid en laag vermogenverbruik prioriteit genieten.

Hoofdstuk 4

Algemeen besluit

4.1 Besluit

Deze thesis handelde over pairings, een recente ontwikkeling op gebied van cryptografie. Pairings laten identiteits-gebaseerde cryptografie toe, wat zeer interessant is voor bijvoorbeeld netwerken van sensoren.

Er werd onderzocht hoe de Tate pairing geïmplementeerd kan worden in hardware. Meer specifiek werd de nadruk gelegd op een compacte implementatie die daarenboven nog eens zo weinig mogelijk vermogen verbruikte. Een implementatie van dat type zou toegepast kunnen worden in de eerder vermelde netwerken van sensoren of toestellen met een beperkt vermogen.

Verscheidene bestaande algoritmen werden aangepast en geoptimaliseerd zodat ze met een minimum gebruik aan geheugen uitgevoerd konden worden. Er werd een geheugenontwerp voorgesteld dat een goed compromis gaf tussen grootte en verbruik. Tevens werden verschillende oppervlakte- en vermogensbesparende technieken toegepast om het uiteindelijke circuit zo goed mogelijk aan de doelstellingen te laten voldoen. Door aanpassing van de kloksnelheid en het aantal gebruikte rekenschakelingen (MALUs) kunnen de parameters van het ontwerp aangepast worden aan de individuele vereisten van een toepassing.

Het resulterende ontwerp is uniek in zijn soort. Ten eerste is de ASIC implementatie met zijn ¡30k gates zeer klein. Verder is het mogelijk een verbruik te behalen van enkele tientallen nanowatt, indien rekensnelheid geen punt is. Ook aan hogere snelheiden blijft het verbruik zeer laag vergeleken met het enige andere gedocumenteerde ontwerp uit de literatuur. Ten tijde van dit schrijven waren nog geen andere ontwerpen gepubliceerd waarbij de nadruk op compactheid en laag vermogenverbruik lag.

4.2 Toekomstig onderzoek

Als toekomstig onderzoek kan het interessant zijn na te gaan of er nog significante optimalisaties aan het ontwerp mogelijk zijn. Daarbij is zou onderzocht moeten worden of de FSM verder te vereenvoudingen valt, aangezien die zowat de helft van de oppervlakte van de totale implementatie inneemt. Het zou ook nuttig zijn te onderzoeken wat de ideale plaatsing van variabelen in het geheugen is, om zo het verbruik en de rekentijd nog verder te verlagen. Ook dient nagegaan te worden waarom bij de synthese de oppervlakte van de schakeling niet kleiner wordt na het toepassen van clock gating technieken en hoe dit opgelost kan worden.

Verder zou ook onderzocht moeten worden in welke mate werken over een groter Galois veld de oppervlakte en het verbruik wijzigt. Ook de implementatie van andere types pairings kan interessant zijn. Zo bestaan er bijvoorbeeld de η_T en de modified Tate pairing die beiden berekend kunnen worden in ongeveer de helft van de tijd nodig voor de berekening van de Tate pairing.

Bibliografie

[1] G. Bertoni, L. Breveglieri, P. Fragneto, G. Pelosi, and L. Sportiello, "Software implementation of Tate pairing over GF(2^m)," in *DATE 06: Proceedings of the conference on Design, automation and test in Europe.* European Design and Automation Association, 2006, pp. 7–11.

[2] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodrguez-Henrquez, "A Comparison Between Hardware Accelerators for the Modified Tate Pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$," in *Lecture Notes in Computer Science*, vol. 5209. Springer, 2008, pp. 297–315.

[3] Certicom Corporation, *SEC 2: Recommended Elliptic Curve Domain Parameters*, september 2000. [Online]. Available: http://www.secg.org

[4] Faraday Technology Corporation, *0.13μm Platinum Standard Cell Databook*, 2004. [Online]. Available: http://www.faraday-tech.com

[5] K. Sakiyama, "Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems," Ph.D. dissertation, KU Leuven, december 2007.

[6] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks," *Security and Privacy in Ad-Hoc and Sensor Networks*, vol. 4357, pp. 6–17, 2006.

[7] D. Hankerson, A. Menezes, and S. Vanstone, *Guide To Elliptic Curve Cryptography.* Springer, 2004.

[8] L. Batina, "Arithmetic And Architectures For Secure Hardware Implementations Of Public-Key Cryptography," Ph.D. dissertation, KU Leuven, 2005.

[9] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases," *Information and Computation*, vol. 78, no. 3, pp. 171–177, 1988.

[10] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady*, vol. 7, p. 595, 1963.

[11] D. Zuras, "On squaring and multiplying large integers," in *Proceedings of IEEE 11th Symposium on Computer Arithmetic ARITH-93*, 1993, p. 260.

[12] Y. K. Lee and I. Verbauwhede, "A Compact Architecture for Montgomery Elliptic Curve Scalar Multiplication Processor." in *WISA*, ser. Lecture Notes in Computer Science, S. Kim, M. Yung, and H.-W. Lee, Eds., vol. 4867. Springer, 2007, pp. 115–127.

[13] M. Müller, A. Wortmann, S. Simon, M. Kugel, and T. Schoenauer, "The impact of clock gating schemes on the power dissipation of synthesizable register files," in *ISCAS (2)*, 2004, pp. 609–612.

[14] P. Schaumont, D. Ching, H. Chan, J. Steensgaard-Madsen, A. Vad-Lorentzen, and E. Simpson, *GEZEL User Manual*, 2007. [Online]. Available: http://rijndael.ece.vt.edu/gezel2

[15] L. B. Oliveira, D. F. Aranha, E. Morais, F. Daguano, J. López, and R. Dahab, "TinyTate: Computing the Tate Pairing in Resource-Constrained Sensor Nodes," in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on.* IEEE Computer Society, 2007, pp. 318–323.

[16] L. B. Oliveira, M. Scott, J. Lopez, and R. Dahab, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," in *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, 2008, pp. 173–180.

[17] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks." *European conference on Wireless Sensor Networks (EWSN08)*, 2008.

[18] R. Ronan, C. OEigeartaigh, C. C. Murphy, M. Scott, and T. Kerins, "Hardware acceleration of the Tate pairing on a genus 2 hyperelliptic curve," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 85–98, 2007.

[19] C. S. Soonhak, C. Shu, S. Kwon, and K. Gaj, "FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields," in *Cryptology ePrint Archive, Report 2006/179*, 2006.

[20] M. Keller, T. Kerins, F. M. Crowe, and W. P. Marnane, "FPGA Implementation of a GF(2^m) Tate Pairing Architecture." in *ARC*, ser. Lecture Notes in Computer Science, K. Bertels, J. a. M. P. Cardoso, and S. Vassiliadis, Eds., vol. 3985. Springer, 2006, pp. 358–369.

[21] P. Grabher and D. Page, "Hardware Acceleration of the Tate Pairing in Characteristic Three." in *CHES*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 398–411.

[22] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi, "Algorithms and Arithmetic Operators for Computing the η_T Pairing in Characteristic Three." *IEEE Trans. Computers*, vol. 57, no. 11, pp. 1454–1468, 2008.

[23] J.-L. Beuchat, H. Doi, K. Fujita, A. Inomata, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, T. Takagi, A. Vithanage, and H. Yamamoto, "FPGA and ASIC Implementations of the η_T Pairing in Characteristic Three," in *Cryptology ePrint Archive, Report 2008/280*, 2008.

[24] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, R. Leupers, R. Mathar, and H. Meyr, "Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves," in *Cryptology ePrint Archive, Report 2009/056*, 2009.

[25] G. Kömürcü and E. Savas, "An Efficient Hardware Implementation of the Tate Pairing in Characteristic Three." in *ICONS*. IEEE Computer Society, 2008, pp. 23–28.