Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

2013•2014
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef
Een bibliotheek van cryptografische operatie omzetten met Lava

Promotor :
dr. Kris AERTS

Promotor :
prof. dr. NELE MENTENS

Nicky Hannosset
Proefschrift ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

KU LEUVEN | universiteit hasselt

KU LEUVEN | universiteit hasselt

2013•2014
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterproef
Een bibliotheek van cryptografische operatie omzetten met Lava

Promotor :
dr. Kris AERTS

Promotor :
prof. dr. NELE MENTENS

Nicky Hannosset
Proefschrift ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

universiteit hasselt | KU LEUVEN

Cryptografische blokken genereren met Lava

Masterproef

Naam: Nicky Hannosset

Studie: Master Elektronica-ICT

Inhoudstabel

Woord vooraf 5

Abstract 7

Summary 9

Verklarende woordenlijst 11

Figurenlijst 11

Lijst van algoritmen 13

Tabellenlijst 13

1. Inleiding 15

1.1 Situering 15

1.2 Doelstelling 15

1.3 Werkomgeving 15

1.4 Aanpak 15

2 Literatuurstudie 17

2.1 Haskell en Lava 17

2.1.1 Haskell en andere functionele talen 17

2.1.2 Uitbreidingen op Haskell 18

2.1.3 De verschillende versies van Lava 19

2.2 Elliptic Curve Cryptography 21

2.2.1 Vier lagen 21

2.2.2 Laag 1: Veldoperaties 21

2.2.3 Laag 2: Puntoptelling en puntverdubbeling 23

2.2.4 Laag 3: Puntvermenigvuldiging en transformaties 25

3 Implementatie 29

3.1 Montgomery vermenigvuldiging 29

3.2 Puntverdubbeling 31

3.2.1 Signalen tussen ControlePad en DataPad 31

3.2.2 Connecties met andere bewerkingen 32

3.2.3 ControlePad 33

3.2.4 DataPad 34

3.3 Puntoptelling 37

3.3.1 ControlePad 37

3.3.2 DataPad 38

3.4 Puntvermenigvuldiging 39

3.4.1 Signalen tussen DataPad en ControlePad 39
3.4.2 Connecties met andere bewerkingen 41
3.4.3 ControlePad 42
3.4.4 DataPad 43
3.5 Modulaire machtsverheffing 45
3.5.1 Signalen tussen DataPad en ControlePad 46
3.5.2 ControlePad 47
3.5.3 DataPad 48
3.6 NtoM en MtoN transformaties 49
3.6.1 Signalen tussen DataPad en ControlePad 50
3.6.2 ControlePad 51
3.6.3 DataPad 51
3.7 AtoP en PtoA transformaties 52
3.7.1 AtoP transformatie 52
3.7.2 PtoA transformatie 53
4 Testen 59
4.1 Montgomery-vermenigvuldiging 59
4.2 Puntverdubbeling 60
4.3 Puntoptelling 61
4.4 Puntvermenigvuldiging 62
4.5 Modulaire machtsverheffing 63
4.6 NtoM transformatie 64
4.7 MtoN transformatie 65
4.8 AtoP transformatie 66
4.9 PtoA transformatie 66
5 Performantie 67
6 VHDL Conversie 71
7 Conclusie 73
8 Uitbreidingsmogelijkheden 75
9 Appendix 77
9.1 Magma code 77
Bibliografie 83

Woord vooraf

Als laatstejaars student aan de universiteit Hasselt en KULeuven heb ik de kans gekregen om een masterproef te maken bij ES&S.

Hierbij wil ik mijn promotoren dr. Kris Aerts en dr. Ir. Nele Mentens bedanken voor hun hulp tijdens de uitvoering van de masterproef, alsook mijn ouders voor de steun en de kansen die ik heb gekregen om te studeren.

Veel lees plezier

Nicky Hannosset

Abstract

De onderzoeksgroep ES&S (Embedded Systems & Security), van KULeuven campus Diepenbeek, is op het moment bezig met het ontwikkelen van de EDA-DSE-tool. Deze tool kan gebruikt worden voor het automatisch genereren van cryptografische bouwblokken in hardware. Traditioneel gebeurt dit vooral in VHDL, maar in deze tool wordt Lava gebruikt, een taal embedded in Haskell, de standaard functionele taal en kan VHDL gegenereerd worden.

Wanneer programmeurs een bepaald programma moeten schrijven, is beveiliging voor het programma een bijzaak en wordt vaak over het hoofd gezien. In dit project wordt dit probleem aangepakt door de generatie van cryptografische blokken te automatiseren.

Deze cryptografische blokken, die worden ontwikkeld, focussen op elliptische krommecryptografie (ECC of Elliptic Curve Cryptography), waarvan een eerste gedeelte al in de huidige versie van de tool zit. In dit project zullen de puntbewerkingen van elliptische krommen geschreven en gesimuleerd worden. Hierna zullen puntverdubbeling, puntoptelling en puntvermenigvuldiging geïntegreerd worden in de tool.

Binnen dit project zijn de tweede en derde laag van het ECC algoritme correct geïmplementeerd, na het aanpassen van de eerste laag. De simulatie toont dit aan. Tegelijk is de kwaliteit van de documentatie verhoogd zodat men in vervolgprojecten gemakkelijker en sneller kan voortbouwen op de resultaten.

Summary

The research group ES&S (Embedded Systems & Security), which is part of KULeuven campus Diepenbeek, is currently working on developing a tool for the automatic generation of cryptographic hardware. The description inside the tool is done in Lava, a language embedded in Haskell. From the Lava description it is possible to generate VHDL code.

When programmers have to write a certain program, the security of the program is overlooked and considered as merely a side issue. In this project the problem is tackled through the integration of the automatic generation of cryptographical blocks inside the tool.

These crypthographical blocks that are being developed focus on ECC (Elliptic Curve Cryptography), from which the first part has already been integrated inside the tool. The ECC point operations are written and simulated within this project.

Within this project the second and third layers of the ECC algorithm have been correctly implemented, after adjusting the first layer. The simulation proves this. At the same time the quality of the documentation has increased so that future projects can construct easier and faster upon the results.

Verklarende woordenlijst

EDA-DSE	Electronic Design Automation - Design Space Exploration
ES&S	Embedded Systems & Security
GHC	Glasgow Haskell Compiler
HDL	Hardware Description Language
DSL	Domain Specific Language
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
XST	Xilinx Synthesis Tool
FPGA	Field-Programmable Gate Array
FST	Finite State Machine
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDH	Elliptic Curve Diffie Hellman
MSB	Most Significant Bit

Figurenlijst

Figuur 1: De verschillende soorten programmeertalen....17
Figuur 2: De geschiedenis van Lava [22] 20
Figuur 3: De 4 lagen van ECC 21
Figuur 4: Grafische voorstelling van 2e stap van Montgomery-vermenigvuldiging [45] 22
Figuur 5: Grafische voorstelling puntverdubbeling [6] 24
Figuur 6: Grafische voorstelling puntoptelling [6]....24
Figuur 7: De vier punttransformaties....25
Figuur 8: Grafische voorstelling puntvermenigvuldiging [8] 26
Figuur 9: Projectief-naar-Affien bewerkingen [4]....27
Figuur 10: Communicatie puntverdubbeling datapad-controlepad....31
Figuur 11: Puntverdubbeling connecties met andere bewerkingen 32
Figuur 12: Puntverdubbeling controlepad 33
Figuur 13: Register van Datapad 34
Figuur 14: Functie pointDoublingReadData 35
Figuur 15: Inlezen inputdata puntverdubbeling....35
Figuur 16: Enables voor wegschrijven data puntverdubbeling....35
Figuur 17: Wegschrijven data puntverdubbeling 36
Figuur 18: Modulaire inverse 36
Figuur 19: Puntverdubbeling code uitgelegd a.d.h.v. het algoritme....37
Figuur 20: Puntoptelling controlepad....38
Figuur 21: Functie pointAddingReadData 39
Figuur 22: Communicatie puntvermenigvuldiging datapad-controlepad....40
Figuur 23: Puntvermenigvuldiging connecties met andere bewerkingen 41
Figuur 24: Puntvermenigvuldiging controlepad....42
Figuur 25: De vermenigvuldigingsfactor 43
Figuur 26: Tellen tot lengte factor doorlopen....43
Figuur 27: Bewerkingen binnen puntvermenigvuldiging....44
Figuur 28: Inlezen en wegschrijven puntvermenigvuldiging....45

Figuur 29: Puntvermenigvuldiging code uitgelegd a.d.h.v. algoritme 45
Figuur 30: Communicatie modulaire machtsverheffing datapad-controlepad 46
Figuur 31: ModExp controlepad 47
Figuur 32: De exponent 48
Figuur 33: Bewerkingen modulaire machtsverheffing 48
Figuur 34: Inlezen en wegschrijven modulaire machtsverheffing 49
Figuur 35: ModExp code uitgelegd a.d.h.v. het algoritme 49
Figuur 36: Communicatie MtoN transformatie datapad-controlepad 50
Figuur 37: MtoN transformatie controlepad 51
Figuur 38: NtoM datapad code 52
Figuur 39: AtoP transformatie code 52
Figuur 40: Communicatie PtoA transformatie datapad-controlepad 53
Figuur 41: PtoA connecties met andere bewerkingen 54
Figuur 42: PtoA transformatie controlepad 55
Figuur 43: Inlezen data PtoA transformatie 56
Figuur 44: Wegschrijven data PtoA transformatie 56
Figuur 45: Bewerkingen PtoA transformatie 57
Figuur 46: Definitie getallen in bits 59
Figuur 47: ciosMultiplierWrapper 60
Figuur 48: Main functie testBox_CIOS 60
Figuur 49: verdubbelingWrapper 61
Figuur 50: Main functie testBox_PtVd_elem 61
Figuur 51: optellingWrapper 62
Figuur 52: Main functie testBox_PtOpt_elem 62
Figuur 53: vermenigvuldigingWrapper 63
Figuur 54: Main functie testBox_PtVm_elem 63
Figuur 55: machtverheffingWrapper 64
Figuur 56: Main functie testBox_ModExp 64
Figuur 57: toMontgomeryWrapper 64
Figuur 58: Main functie testBox_NtoM_elem 65
Figuur 59: toNormalWrapper 65
Figuur 60: Main functie testBox_MtoN_elem 65
Figuur 61: toAffineWrapper 66
Figuur 62: Main functie testBox_PtoA_elem 66
Figuur 63: CIOS Heap Profile by cost-centre stack 68
Figuur 64: CIOS Heap Profile by type 68
Figuur 65: CIOS Heap Profile by closure description 69
Figuur 66: CIOS VHDL conversie error 71
Figuur 67: Magma berekeningen Montgomery vermenigvuldiging 77
Figuur 68: Magma berekeningen puntverdubbeling 78
Figuur 69: Magma berekeningen puntoptelling, deel 1 79
Figuur 70: Magma berkeningen puntoptelling, deel 2 80
Figuur 71: Magma berekeningen puntvermenigvuldiging 81

Lijst van algoritmen

Algoritme 1: Montgomery-vermenigvuldiging [45] ... 22
Algoritme 2: Verbeterde versie Montgomery-vermenigvuldiging [45] ... 22
Algoritme 3: CIOS-vermenigvuldiging [45] ... 23
Algoritme 4: Puntoptelling [4] ... 24
Algoritme 5: Puntverdubbeling [4] ... 25
Algoritme 6: Puntvermenigvuldiging dat de scalar van links naar rechts evalueert [4] ... 26
Algoritme 7: Modulaire machtsverheffing [4] ... 27

Tabellenlijst

Tabel 1: De eigenschappen van functionele talen ... 18
Tabel 2: Verschillen oude en nieuwe versie van CIOS datapad ... 30
Tabel 3: Overzichtstabel van formules voor aantal iteraties ... 67

1. Inleiding

1.1 Situering

Deze opdracht is deel van een huidig project van ES&S [1].
ES&S is een onderzoeksgroep voornamelijk bezig met ingebedde elektronische systemen.
Deze groep is nu al een redelijke tijd bezig met de ontwikkeling van de EDA-DSE tool (vroeger CREA tool), waarin men cryptografische hardware op een hoog niveau kan beschrijven [2]. Momenteel ligt de focus op de automatische generatie van cryptografische blokken met behulp van elliptische krommecryptografie (ECC of Elliptic Curve Cryptography).
Deze tool is geschreven in Lava, een HDL ingebed in Haskell. In Lava kan je FSM's beschrijven, maar die FSM's worden omgezet in combinatorische logica wat de optimisatiemogelijkheden van synthesis tools beperkt. Binnen ES&S is een versie van Lava ontwikkeld waarin FSM's omgezet worden naar FSM's in VHDL. Daarnaast kan de code gesimuleerd worden om rechtsreeks in Lava te testen of de werking van de FSM is zoals gewild, zonder te moeten exporteren naar VHDL.
Het doel van deze tool is om het ontwerp van FSM's en cryptografische blokken makkelijker te maken [3]. In de tool kunnen ook verschillende implementaties voor een bepaalde bewerking gegenereerd worden en vergeleken worden met elkaar. Hierdoor kan de implementatie gevonden worden die het beste past voor de ontwerper. Dit concept noemt men design space exploration (DSE).

1.2 Doelstelling

Het doel is, zoals de titel zegt, het genereren van cryptografische blokken in Lava met behulp van ECC [4] [5] [6]. Het project gaat hierin verder op een onderzoeksproject met BOF-CREA-middelen en de masterproef van Koen Baens [7].
Dit wordt bereikt door de verschillende lagen van ECC te integreren in de tool. De eerste laag is reeds geïmplementeerd en bevat de modulaire optelling en Montgomery vermenigvuldiging. De tweede laag bevat de puntverdubbeling en puntoptelling waarvan de puntverdubbeling al gedeeltelijk geïmplementeerd is in de masterproef van Koen Baens [7]. En uiteindelijk is er de derde laag die de puntvermenigvuldiging bevat. De laatste laag is eigenlijk de vierde laag die cryptografische protocollen zoals ECDH en ECDSA bevat [8], maar deze thesis beperkt zich tot de eerste drie lagen.
Nadat deze puntbewerkingen in code geschreven zijn, worden ze gesimuleerd in Lava, omgezet in VHDL en daarna gesimuleerd in VHDL. Uiteindelijk worden ze dan geïntegreerd in de tool.

1.3 Werkomgeving

Zoals reeds eerder gezegd is de tool geschreven in Lava, meer bepaald de versie York Lava en Lava is ingebed in Haskell. Meer informatie rond Haskell, York Lava en de verschillen met andere Lava versies zal later volgen in de literatuurstudie. Ghc wordt gebruikt om de code van York Lava te compileren. Daarnaast moet ook de VHDL generatie getest worden. Hiervoor wordt gebruikgemaakt van Xilinx.

1.4 Aanpak

Voor men kan beginnen aan het uitbreiden van de tool, moesten drie onderwerpen bestudeerd worden: Haskell en Lava, de bestaande tool en ECC.
Over Haskell was redelijk wat documentatie te vinden, maar over York Lava en Lava in het algemeen was weingig informatie beschikbaar. Over de huidige tool was informatie meegegeven door de

promotors. Over ECC was er veel documentatie te vinden.
Om de bestaande tool beter te begrijpen, was documentatie alleen niet voldoende. Daarvoor was het betere om de werking van de functies te testen. Er moest sowieso ook de code van Koen Baens nog getest worden omdat deze zijn code niet volledig heeft kunnen simuleren. Bij de testen van de huidige versie van de tool is er ontdekt dat de code van de puntverdubbeling niet werkte en dat de Montgomery-vermenigvuldiger niet het resultaat gaf dat verwacht werd.
Aanpassingen zijn gemaakt in de Montgomery-vermenigvuldiger en de puntverdubbeling is grotendeels herschreven met bepaalde stukken bewaard van de code van Baens. Daarnaast zijn de overige puntbewerkingen van laag 2 en 3 geïmplementeerd.
De puntbewerkingen en aanpassingen zijn in Lava gesimuleerd en gaven correcte resultaten.
De export van de FSM's in VHDL en het bestuderen van de genereerde hardware is niet gelukt wegens problemen in de export. Hier wordt verder op ingegaan in Sectie 6: VHDL Conversie.

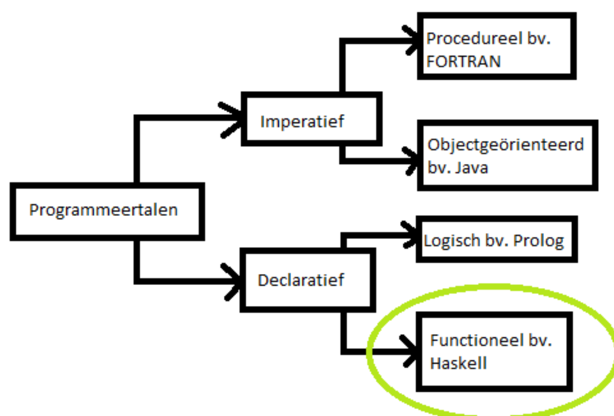
2 Literatuurstudie

Zoals de aanpak hierboven al aangaf, behandelt het literatuuronderzoek enerzijds Lava en Haskell en anderzijds Elliptische Kromme Cryptografie (ECC).
Voor Haskell en Lava toont men waarom deze talen bestaan, hun positionering binnen programmeertalen en hun verschillen met andere talen. De functionaliteiten van deze talen worden besproken en beoordeeld t.o.v. de doelstellingen van deze thesis.
Voor ECC toont men de structuur van het algoritme. Deze wordt in detail bekeken omdat de generatie van cryptografische blokken in dit project gedaan is m.b.v. dit algoritme.

2.1 Haskell en Lava

Haskell is een puur functionele 'luie' programmeertaal die gebruik maakt van monaden [9] [10]. Hiermee is ze uitzonderlijk tussen de verschillende functionele talen. Als eerste wordt in deze studie Haskell vergeleken met andere functionele programmeertalen en dieper ingegaan op wat monaden precies zijn. Daarna wordt gekeken hoe je hardware kan beschrijven in functionele programmmeertalen. Eén van de mogelijkheden is Lava. Lava is een HDL ingebed in Haskell die gebruikt wordt om sequentiële circuits, ASIC's en FPGA's te ontwerpen. Het laatste punt gaat meer gedetailleerd in op Lava en de verschillende versies ervan.

2.1.1 Haskell en andere functionele talen



Figuur 1: De verschillende soorten programmeertalen

Eerst wordt een verschil gemaakt tussen puur functionele programmeertalen [11] [12], bv. Haskell, Clean en Miranda, en niet puur functionele programmeertalen, bv. ML, Scheme en Erlang. Puur functioneel betekent dat de functies geen bijwerkingen hebben op geheugen of I/O. Puur functioneel heeft een aantal gunstige eigenschappen.

Allereerst als een pure functie twee keer dezelfde argumenten binnenkrijgt zal ze twee keer hetzelfde resultaat geven. Dit is referentiële transparantie. In een niet pure taal kan een functie oproep van f(x) twee keer een ander resultaat hebben wanneer die gebruik maakt van neveneffecten. Dat kan programma's zowel minder leesbaar maken als moeilijker om de correctheid van te bewijzen. Ten tweede kan een niet gebruikt resultaat van een pure functie verwijderd worden zonder nadelige gevolgen (garbage collection/lazyness). Ten derde als twee pure functies onafhankelijk zijn van elkaar is de volgorde van uitvoering niet belangrijk en kunnen ze ook parallel uitgevoerd worden. Ten laatste als de taal volledig puur functioneel is dan heeft de compiler meer

vrijheid in het ordenen en combineren van functies in een programma en dit maakt intelligente programmatransformaties mogelijk.

Vervolgens wordt een verschil gemaakt tussen 'strikte' programmeertalen, bv. ML, Scheme en Erlang, en 'luie' programmeertalen, bv. Haskell, Clean en Miranda. Bij een 'strikte' taal (eager evaluation) wordt elk argument van een functie geëvalueerd zelfs als het niet nodig is voor het resultaat van de functie. Bij een 'luie' taal (lazy evaluation) wordt de waarde van een bepaald argument alleen geëvalueerd als ze nodig is voor het resultaat van de functie. Bijvoorbeeld bij een functie telOp(a,b) zullen de waardes van a en b geëvalueerd worden maar bij de functie lengte([a,b]) zal dit niet het geval zijn. Daarnaast is er een verschil tussen dynamically typed programmeertalen, bv. Scheme en Erlang, en statically typed, bv. Haskell, Clean, Miranda en ML [13]. Bij dynamically typed is er veel vrijheid bij het geven van types. In een lijst mag bijvoorbeeld het eerste type een string zijn, terwijl het tweede type een integer is. Bij statically typed is het geven van types meer strikt. Het compilen zal problemen geven als je bv. een string toevoegt bij een integer. Bij statically typed is er dus meer schrijfwerk, maar zal bij runtime minder snel vastlopen door type errors dan bij dynamically typed.

Ten slotte worden monaden besproken. Bij Haskell wordt bij IO de structuur van monaden gebruikt [14]. Het punt is dat IO functies per definities tijdsafhankelijke verschillende waarden geven, bijvoorbeeld bij de functie read() die invoer van de gebruiker opvraagt. Een monade bindt verschillende operaties achter elkaar zodat de tijdsafhankelijkheid meegenomen wordt. Monad-operatoren binden het resultaat van een functie aan de invoer van de volgende functie. Dit komt goed tot uiting in IO-operaties waar de volgorde een rol speelt. Als een error plaatsvindt bij een bepaalde operatie van de monade, dan zullen de overige operaties niet worden uitgevoerd. Bijvoorbeeld als je eerst iets inleest van een bestand en dan resultaten wegschrijft naar een tweede bestand, zal dat wegschrijven niet gebeuren wanneer het inlezen mislukt. Zonder nomades ligt de volgorde van uitvoering niet vast en zou je in de problemen kunnen komen. De andere genoemde functionele talen maken geen gebruik van nomaden voor IO, maar kiezen andere oplossingen, dikwijls via niet pure functies.

	Evaluatie	IO	Typing	Puur
Haskell	lazy	monaden	static	ja
Clean	lazy	uniciteit	static	ja
Miranda	lazy	lazy lists	static	ja
ML	eager	bijwerkingen	static	nee
Scheme	eager	bijwerkingen	dynamic	nee
Erlang	eager	bijwerkingen	dynamic met compiler annotaties voor static typing	nee

Tabel 1: De eigenschappen van functionele talen

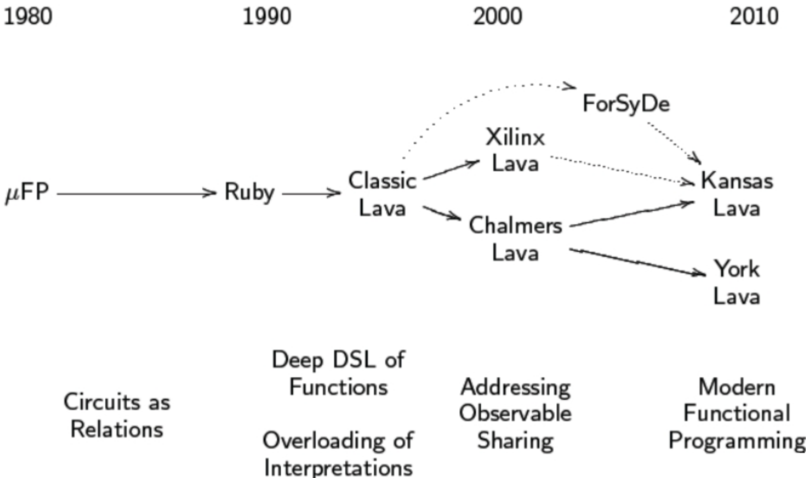
2.1.2 Uitbreidingen op Haskell

Er zijn in de industie en in onderzoeksgebied verschillende extensies gemaakt die gebruik maken van Haskell. In deze sectie zullen vier hiervan besproken worden: Bluespec, Hawk, Cryptol en Lava. Bluespec [15] [16] [17] [18] [19] wordt gebruikt voor het ontwerp van ASIC's en FPGA's net zoals Lava. Het is beheerd door het bedrijf Bluespec Inc. De taal begon als een Haskell DSL, maar heeft nu bijna geen Haskell syntax meer.

Hawk [20] [21] [22] [23] is een taal die wordt gebruikt voor het ontwerp van microprocessor architecturen. Hawk is ingebed in Haskell, net zoals Lava. Een nadeel is dat er niet veel geschreven wordt rond Hawk en ook niet zoveel gebruikt wordt in onderzoeksgroepen als Lava.
Cryptol [24] [25] [26] [27] [28] [29] [30] [31] is een taal gebruikt voor het beschrijven van cryptografische algoritmen en genereren van FPGA layouts gecreëerd door het bedrijf Galois Connections. Omdat Cryptol meer specifiek op beveiligingsaspecten focust en weinig ruimte laat voor andere uitbreidingen, wordt deze taal niet gebruikt voor de EDA-DSE tool. Cryptol is niet ingebed in Haskell, maar bevat gelijkaardige principes als Haskell.
Lava is gecreëerd door Mary Sheeran (en anderen) voor het beschrijven, genereren, simuleren en verifiëren van circuits. Deze taal wordt voornamelijk gebruikt bij onderzoeksgroepen van universiteiten. Deze groepen hebben voor hun onderzoek uitbreidingen aan de taal aangebracht waardoor er nu verschillende versies van Lava bestaan. Deze zijn Chalmers Lava, York Lava en Kansas Lava. Omdat Lava zoveel gebruikt wordt bij onderzoek in de generatie en simulatie van circuits, leek dit ook een uitstekende taal voor de tool.

2.1.3 De verschillende versies van Lava

De allereerste versie van Lava geschreven door Mary Sheeran is verschillende keren uitgebreid over de jaren heen [32] [22] [33]. Zo was er eerst de Chalmers versie geschreven door Koen Claessen en onderhouden door Emil Axelsson. Beide personen zijn deel van het departement Computer Science van de Chalmers universiteit in Gothenburg. De Chalmers versie wordt dikwijls beschouwd als de standaard voor de latere uitbreidingen van Lava. Chalmers Lava maakt gebruik van een principe genaamd 'observable sharing' [34] [35]. Dit is nodig om recursie en 'sharing' [36], het fysiek opslaan van data die dikwijls gebruikt wordt, te verenigen met de implementatie van DSL's. Naast Chalmers was er de Xilinx versie van Satnam Singh, deel van de onderzoeksgroep van Xilinx in San Jose California. De Xilinx versie is vooral gefocust op de beschrijving van circuits voor Xilinx FPGA's. De versie waarin dit project werkt, is de York versie [37] [33]. Deze versie is geschreven door Matthew Naylor van de onderzoeksgroep PLASMA van de universiteit van York. York Lava was deel van het Reduceron project [38] [39] en is een variatie op Chalmers Lava. Deze versie verwijderde overbodige functies en voegde functies toe voor veelgebruikte circuits zoals multiplexers en decoders De laatste versie van Lava is Kansas Lava geschreven door Andy Gill van de Kansas Universiteit [40] [41] [33]. Kansas Lava maakt gebruik van 'observable sharing', net zoals Chalmers Lava, voor het representeren en vangen van cycli in hardware, maar heeft een expliciete monad voor externe connectiviteit. Daarnaast is er een agressiever gebruik van types en type extensies om hardware restricties op te vangen. ES&S overweegt over te stappen op Kansas Lava.



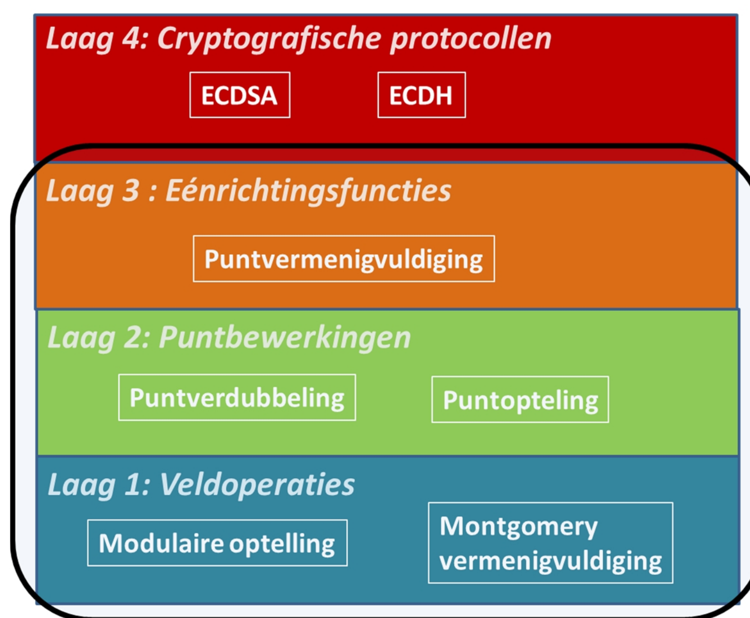
Figuur 2: De geschiedenis van Lava [22]

2.2 Elliptic Curve Cryptography

ECC maakt gebruik van een aantal wiskundige berekeningen die speciaal gedefinieerd zijn voor elliptische krommen. Bij elliptische krommen is er het verschijnsel dat als je een punt meerdere keren met zichzelf optelt, het heel moeilijk wordt om te bepalen hoeveel keer het punt met zichzelf is opgeteld wanneer je enkel het originele punt en het resulterende punt kent. Dit fenomeen kan gebruikt worden om communicatie vast te leggen tussen twee personen waarbij de originele factor alleen bekend is bij deze twee personen. Hierdoor krijg je een beveiligd communicatiekanaal [42].

2.2.1 Vier lagen

Er zijn vier lagen in ECC. De onderste laag zijn de modulaire bewerkingen zoals de modulaire optelling/aftrekking en de Montgomery-vermenigvuldiging [43]. De tweede laag zijn de puntberekeningen zoals de puntoptelling en puntverdubbeling. In de derde laag bevindt zich puntvermenigvuldiging. De bovenste laag bevat cryptografische protocollen zoals ECDSA (digitale handtekening) en ECDH (sleutelovereenkomst) [44].
De eerste laag is al aanwezig in de bestaande tool. Voor de puntverdubbeling van de tweede laag is een aanzet gedaan in de masterproef van Koen Baens [7].
Het deel waarop dit project focust is aangeduid in Figuur 3.



Figuur 3: De 4 lagen van ECC

2.2.2 Laag 1: Veldoperaties

2.2.2.1 Montgomery vermenigvuldiging

Omdat bij het testen van de tool, de resultaten niet zoals verwacht waren bij de Montgomery vermenigvuldiging, zijn er aanpassingen gemaakt in deze code. Hiervoor is er ook studie gedaan in de werking van de Montgomery-vermenigvuldiging [45] [46]. De Montgomery vermenigvuldiging is niet zo simpel als de modulaire optelling. Het resultaat van Mont(a,b) is niet (a*b mod n) maar (a*b*R′ mod n) met n de modulus. R′ is de inverse van R met R=2^(field_length van de getallen). Algoritme 1 en Algoritme 2 tonen het algoritme van de Montgomery-vermenigvuldiging en een verbeterde versie van dat algoritme [45] [47] [48].

Algorithm Montgomery multiplication

Require: $M = (M_{n-1} \dots M_0)_{2^b}$, $X = (X_{n-1} \dots X_0)_{2^b}$, $Y = (Y_{n-1} \dots Y_0)_{2^b}$ with $0 \le X, Y < M$, $R = 2^{n \cdot b}$, $\gcd(M, 2^b) = 1$ and $M' = -M^{-1} \mod 2^b$

Ensure: $(X \cdot Y \cdot R^{-1}) \mod M$

1: $T = (T_n \dots T_0)_{2^b} \leftarrow 0$
2: **for** i from 0 to $n-1$ **do**
3: $\quad U_i \leftarrow ((T_0 + X_0 \cdot Y_i) \cdot M') \mod 2^b$
4: $\quad T \leftarrow (T + X \cdot Y_i + M \cdot U_i)/2^b$
5: **end for**
6: **if** $T \ge M$ **then**
7: $\quad T \leftarrow T - M$
8: **end if**
9: Return T

Algoritme 1: Montgomery-vermenigvuldiging [45]

De verbeterde versie elimineert de laatste aftrekking in het vorige algoritme, maar de X- en Y-termen hebben een extra woord. De R-waarde is daardoor ook 2^((n+1)*b) i.p.v. 2^(n*b) met n het aantal woorden in de getallen en b de woordlengte.

Algorithm Improved Montgomery multiplication

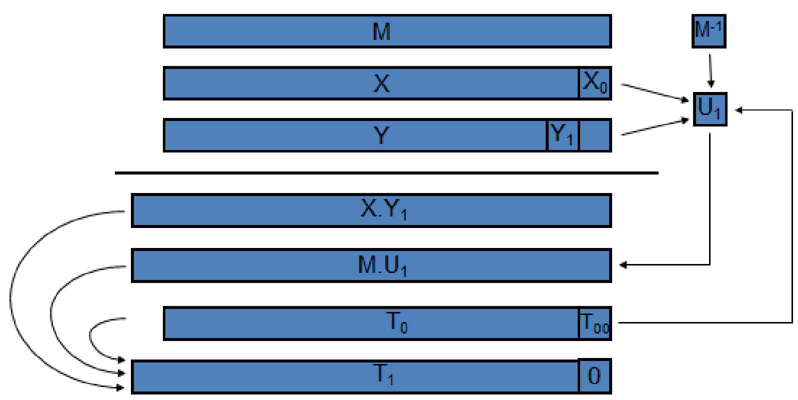
Require: $M = (M_{n-1} \dots M_0)_{2^b}$, $X = (X_n \dots X_0)_{2^b}$, $Y = (Y_n \dots Y_0)_{2^b}$ with $0 \le X, Y < 2 \cdot M$, $R = 2^{(n+1) \cdot b}$, $\gcd(M, 2^b) = 1$ and $M' = -M^{-1} \mod 2^b$

Ensure: $(X \cdot Y \cdot R^{-1}) \mod M$

1: $T = (T_n \dots T_0)_{2^b} \leftarrow 0$
2: **for** i from 0 to n **do**
3: $\quad U_i \leftarrow ((T_0 + X_0 \cdot Y_i) \cdot M') \mod 2^b$
4: $\quad T \leftarrow (T + X \cdot Y_i + M \cdot U_i)/2^b$
5: **end for**
6: Return T

Algoritme 2: Verbeterde versie Montgomery-vermenigvuldiging [45]

Om de werking van dit algoritme beter te begrijpen, is er Figuur 4. Hierin is te zien hoe U_i gemaakt wordt uit de M',T_0, X_0 en Y_i waarden. M is de modulus en M' is het inverse van de modulus. X en Y zijn de vermenigvuldigingstermen. Daarmee wordt T gemaakt. T is het tussenresultaat van elke stap van de for-lus. De figuur toont de stap voor i=1, maar dit wordt dus gedaan voor i van 0 tot n-1.



Figuur 4: Grafische voorstelling van 2e stap van Montgomery-vermenigvuldiging [45]

In de bestaande tool wordt gewerkt met de CIOS- en FIOS- versies van Montgomery-vermenigvuldiging, die frequent toegepast worden voor implementaties in ingebedde processoren, maar ook in hardware. In de literatuurstudie en de aanpassingen in de code wordt nu gefocust op de CIOS versie [45] [43]. Het algoritme hiervan staat in Algoritme 3. De FIOS versie is in dit project niet bestudeerd en ook nog niet aangepast.

Algorithm Separated Operand Scanning (SOS) method for Montgomery multiplication, where ADD propagates the carry throughout T and RED denotes the conditional final reduction step

Require: $M = (M_{n-1} \dots M_0)_{2^w}$, $X = (X_{n-1} \dots X_0)_{2^w}$, $Y = (Y_{n-1} \dots Y_0)_{2^w}$
with $0 \leq X, Y < M$, $R = 2^{n \cdot w}$ with $\gcd(M, 2^w) = 1$
and $M' = -M^{-1} \mod 2^w$
Ensure: $(X \cdot Y \cdot R^{-1}) \mod M$
1: $T = (T_{2n-1} \dots T_0)_{2^w} \leftarrow 0$
2: **for** i from 0 to $n-1$ **do**
3: $\quad C \leftarrow 0$
4: $\quad$ **for** j from 0 to $n-1$ **do**
5: $\quad\quad (C, S) \leftarrow T_{i+j} + X_j \cdot Y_i + C$
6: $\quad\quad T_{i+j} \leftarrow S$
7: $\quad$ **end for**
8: $\quad T_{i+n} \leftarrow C$
9: **end for**
10: **for** i from 0 to $n-1$ **do**
11: $\quad C \leftarrow 0$
12: $\quad U \leftarrow (T_i \cdot M') \mod 2^w$
13: $\quad$ **for** j from 0 to $n-1$ **do**
14: $\quad\quad (C, S) \leftarrow T_{i+j} + U \cdot M_i + C$
15: $\quad\quad T_{i+j} \leftarrow S$
16: $\quad$ **end for**
17: $\quad \text{ADD}(T_{i+n}, C)$
18: **end for**
19: **for** i from 0 to n **do**
20: $\quad T_i \leftarrow T_{i+n}$
21: **end for**
22: $\text{RED}(T)$
23: Return T

Algoritme 3: CIOS-vermenigvuldiging [45]

2.2.3 Laag 2: Puntoptelling en puntverdubbeling

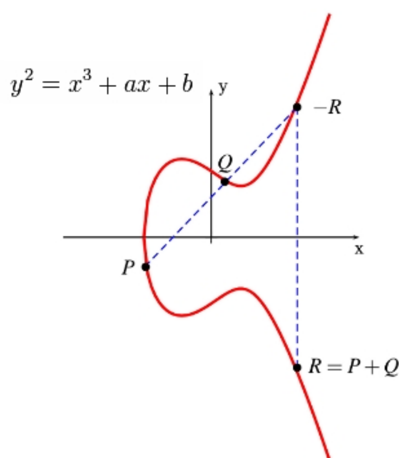
De elliptische kromme heeft als vergelijking: $E : Y^2 = X^3 + aXZ^4 + bZ^6$. Er wordt gebruikgemaakt van het modified Jacobian coördinatenstelsel waarbij naast (X,Y,Z) ook aZ^4 in rekening wordt gebracht. Een punt in dit stelsel wordt dus voorgesteld door (X,Y,Z, aZ^4) [49] [50].
De werking van de puntoptelling en puntverdubbeling zijn te zien in Algoritme 4, Algoritme 5, Figuur 5 en Figuur 6 [4] [6]. De puntverdubbeling neemt de raaklijn van de curve in het punt P die men wil verdubbelen. Waar dit punt snijdt met de curve, wordt een verticale lijn genomen. Het punt waar deze lijn de curve snijdt, is het dubbel van het punt P. De puntoptelling is gelijkaardig. Alleen wordt in het begin de snijlijn van de twee punten P en Q, waarvan men de som wilt, genomen i.p.v. de raaklijn in punt P. In de algoritmes is te zien dat deze berekeningen zijn op te splitsen in modulaire optellingen en Montgomery vermenigvuldigingen.

Algorithm EC point addition

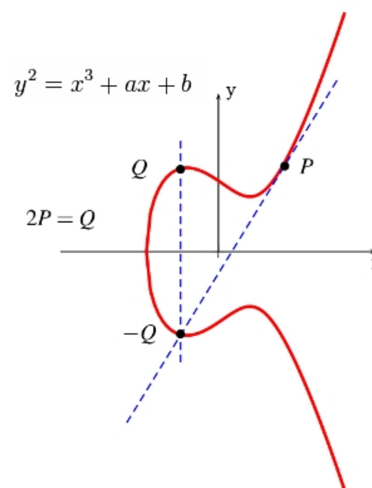
Require: $P_1 = (X_1, Y_1, 1, a)$, $P_2 = (X_2, Y_2, Z_2, aZ_2^4)$
Ensure: $P_1 + P_2 = P_3 = (X_3, Y_3, Z_3, aZ_3^4)$

1.	$T_1 \leftarrow Z_2^2$		
2.	$T_2 \leftarrow xT_1$	(U_1)	
3.	$T_1 \leftarrow T_1 Z_2$		$T_3 \leftarrow X_2 - T_2 \ (H)$
4.	$T_1 \leftarrow yT_1$	(S_1)	
5.	$T_4 \leftarrow T_3^2$		$T_5 \leftarrow Y_2 - T_1 \ (r)$
6.	$T_2 \leftarrow T_2 T_4$		
7.	$T_4 \leftarrow T_4 T_3$		$T_6 \leftarrow 2T_2$
8.	$Z_3 \leftarrow Z_2 T_3$		$T_6 \leftarrow T_4 + T_6$
9.	$T_3 \leftarrow T_5^2$		
10.	$T_1 \leftarrow T_1 T_4$		$X_3 \leftarrow T_3 - T_6$
11.	$aZ_3^4 \leftarrow Z_3^2$		$T_2 \leftarrow T_2 - X_3$
12.	$T_3 \leftarrow T_5 T_2$		
13.	$aZ_3^4 \leftarrow (aZ_3^4)^2$		$Y_3 \leftarrow T_3 - T_1$
14.	$aZ_3^4 \leftarrow a\,(aZ_3^4)$		

Algoritme 4: Puntoptelling [4]



Figuur 6: Grafische voorstelling puntoptelling [6]



Figuur 5: Grafische voorstelling puntverdubbeling [6]

Algorithm EC point doubling

Require: $P_1 = (X_1, Y_1, Z_1, aZ_1^4)$
Ensure: $2P_1 = P_3 = (X_3, Y_3, Z_3, aZ_3^4)$

1.	$T_1 \leftarrow Y_1^2$	$T_2 \leftarrow 2X_1$
2.	$T_3 \leftarrow T_1^2$	$T_2 \leftarrow 2T_2$
3.	$T_1 \leftarrow T_2T_1$ (S)	$T_3 \leftarrow 2T_3$
4.	$T_2 \leftarrow X_1^2$	$T_3 \leftarrow 2T_3$
5.	$T_4 \leftarrow Y_1Z_1$	$T_3 \leftarrow 2T_3$ (U)
6.	$T_5 \leftarrow T_3\,(aZ_1^4)$	$T_6 \leftarrow 2T_2$
7.	$T_2 \leftarrow T_6 + T_2$	
8.	$T_2 \leftarrow T_2 + (aZ_1^4)$ (M)	
9.	$T_6 \leftarrow T_2^2$	$Z_3 \leftarrow 2T_4$
10.	$T_4 \leftarrow 2T_1$	
11.	$X_3 \leftarrow T_6 - T_4$	
12.	$T_1 \leftarrow T_1 - X_3$	
13.	$T_2 \leftarrow T_2T_1$	$aZ_3^4 \leftarrow 2T_5$
14.	$Y_3 \leftarrow T_2 - T_3$	

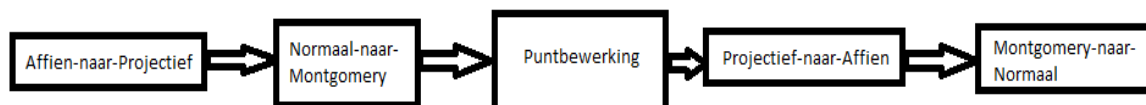
Algoritme 5: Puntverdubbeling [4]

De puntverdubbeling is geschreven in Lava in de masterproef van Koen Baens [7]. Deze code is nog niet volledig correct gesimuleerd. Er is dus een grote kans dat er nog iets moet aangepast worden. Wanneer de puntverdubbeling werkt of werkende gemaakt is, zal de puntoptelling geïmplementeerd worden.

2.2.4 Laag 3: Puntvermenigvuldiging en transformaties

Bij de puntbewerkingen denken we vooral aan de puntvermenigvuldiging van de vorm R = s*P. Hierbij zijn er 5 stappen zoals beschreven in figuur 14:

- De Affien naar Projectief transformatie
- De Normaal naar Montgomery transformatie
- De eigenlijke puntbewerking
- De Projectief naar Affien transformatie
- De Montgomery naar Normaal transformatie



Figuur 7: De vier punttransformaties

We bespreken eerst de puntvermenigvuldiging en dan de vier transformaties [4].

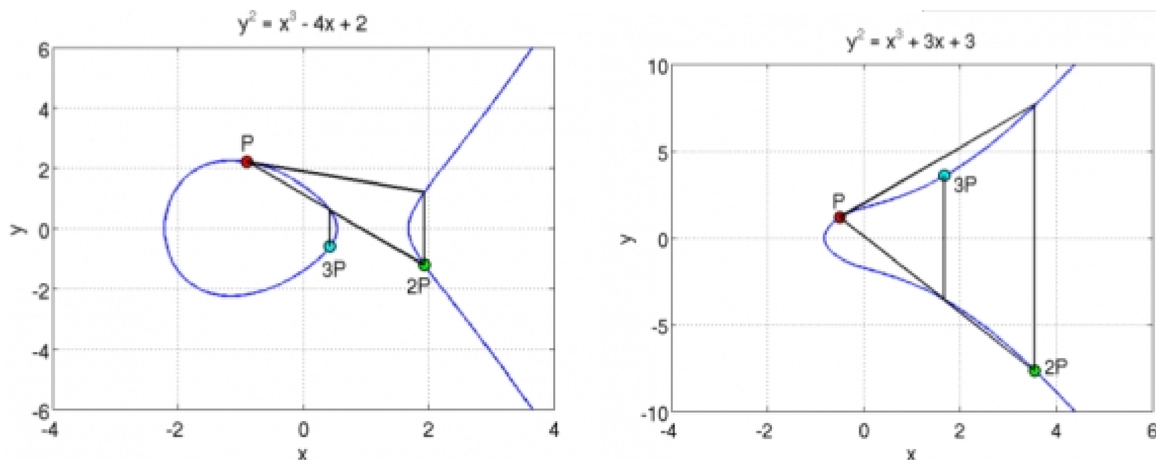
2.2.4.1 Puntvermenigvuldiging

Het algoritme dat na de puntoptelling en –verdubbeling moet geïmplementeerd worden is de puntvermenigvuldiging. Dit algoritme is hieronder te zien [4]. De werking hiervan wordt ook getoond in twee grafieken [8]. Zoals te zien is in Algoritme 6 en Figuur 8 is de puntvermenigvuldiging een combinatie van puntverdubbelingen en –optellingen.

Algorithm Elliptic Curve Point Multiplication

Require: EC point $P = (x, y)$, integer k, $0 < k < M$, $k = (k_{l-1}, k_{l-2}, \cdots, k_0)_2$, $k_{l-1} = 1$ and M
Ensure: $Q = (x', y')$
1: $Q \leftarrow P$
2: **for** i from $l - 2$ downto 0 **do**
3: $Q \leftarrow 2Q$
4: **if** $k_i = 1$ **then**
5: $Q \leftarrow Q + P$
6: **end if**
7: **end for**
8: Return (Q)

Algoritme 6: Puntvermenigvuldiging dat de scalar van links naar rechts evalueert [4]



Figuur 8: Grafische voorstelling puntvermenigvuldiging [8]

2.2.4.2 Modulaire machtsverheffing

De transformatie van projectieve coördinaten naar affiene coördinaten vereist de berekening van een inverse machtsverheffing. We bespreken dan ook eerst deze bewerking vooraleer we de transformaties aanpakken. Volgens de kleine stelling van Fermat [51] [52] heeft de bewerking a^(-1) hetzelfde resultaat als de bewerking a^(p-2) mod p, met p de modulus. Hiervoor wordt de modulaire machtsverheffing geïmplementeerd. Deze is gebaseerd op onderstaand algoritme [4].

Algorithm Modular exponentiation

Require: integers $0 \leq M < N$, $0 < E < N$, $E = (e_{t-1}, e_{t-2}, \cdots, e_0)_2$, $e_{t-1} = 1$ and N

Ensure: $M^E \bmod N$

1: $A \rightarrow M$
2: **for** i from $t-2$ to 0 **do**
3: $\quad A \rightarrow AA \bmod N$
4: $\quad$ **if** $e_i = 1$ **then**
5: $\quad\quad A \rightarrow AM \bmod N$
6: $\quad$ **end if**
7: **end for**
8: Return (A)

Algoritme 7: Modulaire machtsverheffing [4]

2.2.4.3 Affien-naar-Projectief transformatie

Deze transformatie dient om de affiene coördinaten van de punten om te zetten naar projectieve coördinaten. De affiene coördinaten zijn van de vorm (x,y) en de affiene vergelijk van de curve is y^2=x^3+a*x+b. De projectieve coördinaten zijn van de vorm (X,Y,Z,a*Z^4) en de projectieve vergelijking is Y^2=X^3+a*X*(Z^4)+b*(Z^6). Hierin is x gelijk aan X/(Z^2) en y gelijk aan Y/(Z^3). De transformatie zelf is redelijk simpel. Neem gewoon Z=1. Dan is X=x, Y=y en a*(Z^4)=a.

2.2.4.4 Normaal-naar-Montgomery transformatie

Deze transformatie is nodig om de Montgomery waarden van de input te verkrijgen. De Montgomery waarde van een getal 'a' wordt bekomen door een Montgomery vermenigvuldiging te berekenen van dit getal met R^2. R is 2^(field_length van de getallen). Deze vermenigvuldiging vindt dus vier keer plaats, één keer voor elke coördinaat.

2.2.4.5 Projectief-naar-Affien transformatie

Voor deze transformatie zijn volgende vier bewerkingen nodig:

$$Z^{-2}R = Mont(Z^{-1}R, Z^{-1}R)$$
$$xR = XZ^{-2}R = Mont(XR, Z^{-2}R)$$
$$Z^{-3}R = Mont(Z^{-1}R, Z^{-2}R)$$
$$yR = YZ^{-3}R = Mont(YR, Z^{-3}R)$$

Figuur 9: Projectief-naar-Affien bewerkingen [4]

Nu om aan deze bewerkingen te beginnen, is Z^(-1) nodig. En dit is de reden dat de modulaire machtsverheffing is geïmplementeerd. De xR en yR waarden zijn de Montgomery waarden van de x en y coördinaten van het resultaat.

2.2.4.6 Montgomery-naar-Normaal transformatie

Bij deze laatste transformatie zijn maar twee Montgomery vermenigvuldigingen nodig, namelijk de Montgomery vermenigvuldigingen van xR en yR met 1. Dit is omdat de Montgomery vermenigvuldiging van een getal (a*R) met 1 geeft (a*R)*1*R^(-1) en dit is gelijk aan a.

3 Implementatie

In sectie drie 'Implementatie' gaat de werking van de geïmplementeerde bewerkingen uitgelegd worden. Allereerst komen de aanpassingen van de Montgomery-vermenigvuldiger aan bod en waarom deze nodig zijn. In sectie twee en drie zijn de implementaties van de puntverdubbeling en puntoptelling van de 2e laag van ECC aan de beurt. Uiteindelijk wordt de werking van de puntvermenigvuldiging getoond.

3.1 Montgomery vermenigvuldiging

Om de werking van de 1e laag bewerkingen beter te begrijpen werden simulaties uitgevoerd van de modulaire optelling en Montgomery-vermenigvuldiging. Maar bij de simulatie van de Montgomery-vermenigvuldiging werd niet het verwachte resultaat bekomen. Na dieper onderzoek naar de Montgomery-vermenigvuldiging werden er aanpassingen gemaakt aan de versie van de Montgomery-vermenigvuldiging binnen de tool. Tot nu toe zijn er wel alleen nog maar aanpassingen gemaakt aan de CIOS versie en niet de FIOS versie van de Montgomery-vermenigvuldiging. Er zal nu dieper ingegaan worden welke deze aanpassingen zijn.

Allereerst beginnen we met de veranderingen binnen het controlepad. Er werd een extra state 'sRED' toegevoegd voor de 'sDone' state waarin de modulus afgetrokken wordt van het eindresultaat wanneer dit resultaat groter is dan de modulus. Dit kan ook anders aangepakt worden door de 'verbeterde' versie van Montgomery te implementeren in de CIOS vermenigvuldiger, maar de eerste pogingen om dit te bereiken gaven niet het gewenste resultaat en er is dan verdergegaan met de versie die wel het juiste resultaat gaf. Daarnaast werden de 'Last' states van fase1 en fase2 van de vermenigvuldiging één cyclus groter gemaakt. Hierdoor worden T(n+1) en T(n) elk apart in een cyclus toegevoegd aan de variabele T.
Deze twee veranderingen hebben als gevolg dat er twee extra inputs en twee extra outputs zijn aan het controlepad, nl. iCntr3Tec en oCntr3Clr voor het tellen van de cycli van de 'Last' states en iModDone en oModAdd om te zeggen wanneer de 'sRED' state begint en gedaan is.

In het datapad zijn er vier veranderingen. In het begin wordt aan de hand van field_width (het aantal bits in een getal) en datapad_width (het aantal bits in een blok) een derde variabele gemaakt genaamd s_width. Deze variabele geeft het aantal blokken in een getal. Het resultaat neemt nu de eerste (s_width*dp_width=f_width) waardes van het register dataT in plaats van het originele (f_width+dp_width) aantal waardes. In de originele versie was dit (f_width+dp_width) omdat men de verbeterde versie van de Montgomery-vermenigvuldiging (zie sectie 2.2.2.1: Montgomery vermenigvuldiging) wilde implementeren waarbij alle waardes een bitwoord extra kregen, maar zoals eerder vermeld wordt nu de extra state 'sRED' gebruikt.
Daarnaast worden de gehele getallen x, y en n ingevoerd en de LSB's geplaatst in registers dataXLSB, dataYLSB en dataNLSB binnen de core van de vermenigvuldiging en zo gebruikt in de vermenigvuldigingen. De vroeger versie gebruikte shiftReg buiten de core om de i-waardes van x, y en n te bepalen. De nieuwe manier is handiger om de berekeningen van het algoritme (zie Algoritme 3) te implementeren.
Een derde verandering is dat het register dataT nu grootte ((s_width+2)*dp_width = f_width + 2*dp_width) heeft i.p.v. het originele (f_width+3*dp_width). Deze grootte is veranderd om dezelfde reden als de eerste verandering: de verbeterde versie van Montgomery-vermenigvuldiging wordt niet geïmplementeerd. Daarom was het vroeger optellen met 3*dp_width en is het nu optellen met 2*dp_width). Het register dataT heeft normaal twee bitwoorden extra volgens het algoritme en bij

de verbeterde versie dus nog één extra.
Als vierde verandering is er nu de 'sRED' state zoals eerder al gezegd waarbij het tussenresultaat z modulair opgeteld wordt met nul zodanig dat het uiteindelijk resultaat kleiner is dan de modulus. Deze state is nodig omdat we niet de verbeterde versie van Montgomery-vermenigvuldiging gebruiken. Doordat er nu een modulaire optelling is toegevoegd, moet er bij de parameters ook een adderArch worden toegevoegd.

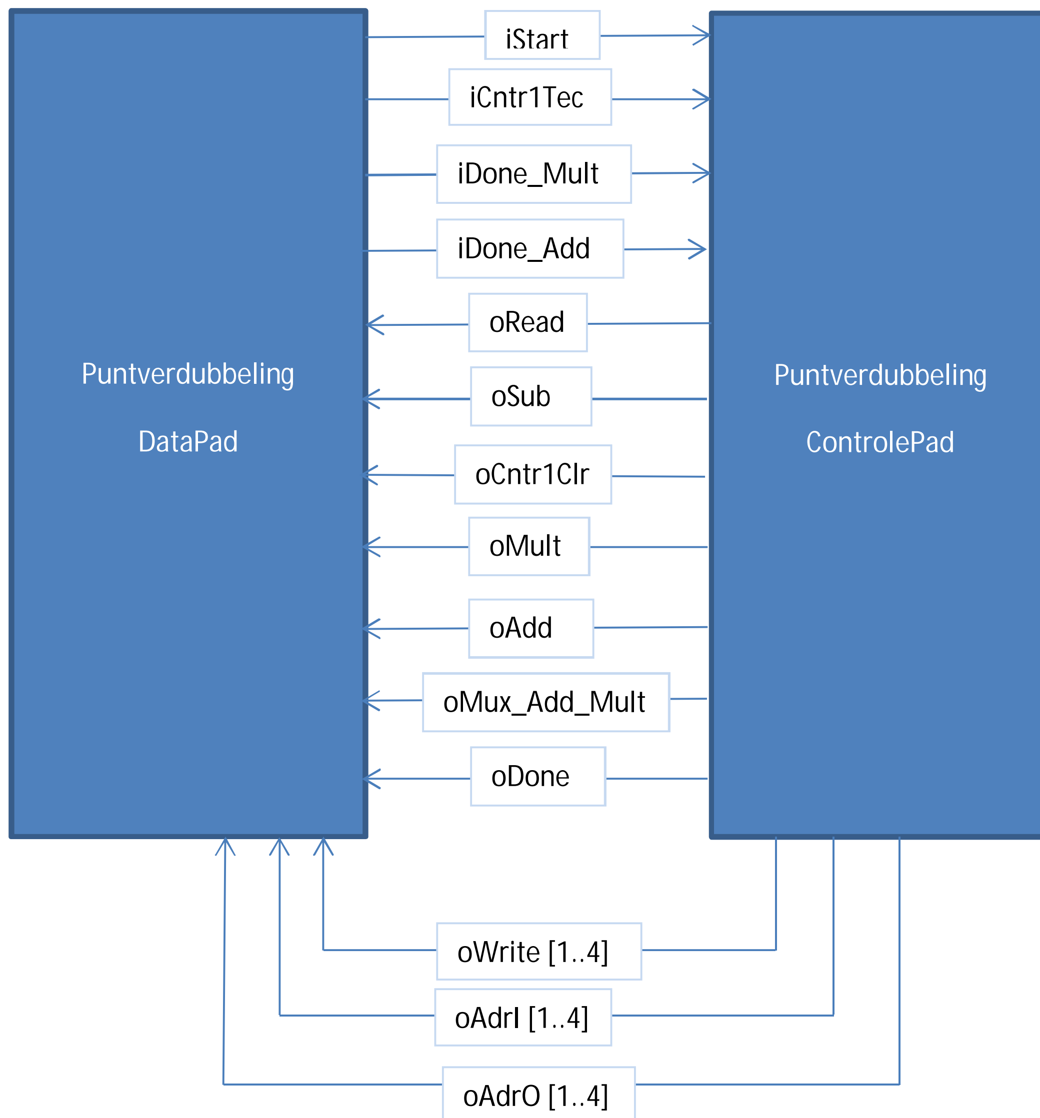
	x[i], y[i], n[i] bepalen	grootte resultaat	grootte register dataT	modulair resultaat
oude versie CIOS	via shiftReg buiten de core	f_width	f_width+3*dp_width	via verbeterde Montgomery
nieuwe versie CIOS	via registers binnen de core	f_width+1	f_width+2*dp_width	via extra 'RED' state

Tabel 2: Verschillen oude en nieuwe versie van CIOS datapad

3.2 Puntverdubbeling

In deze sectie worden eerst de communicatie tussen het controlepad en datapad getoond. En daarna wordt de werking van het controlepad en datapad uitgelegd. De code staat in de bestanden PtVd_elem_CP.hs en PtVd_elem_DP.hs.

3.2.1 Signalen tussen ControlePad en DataPad

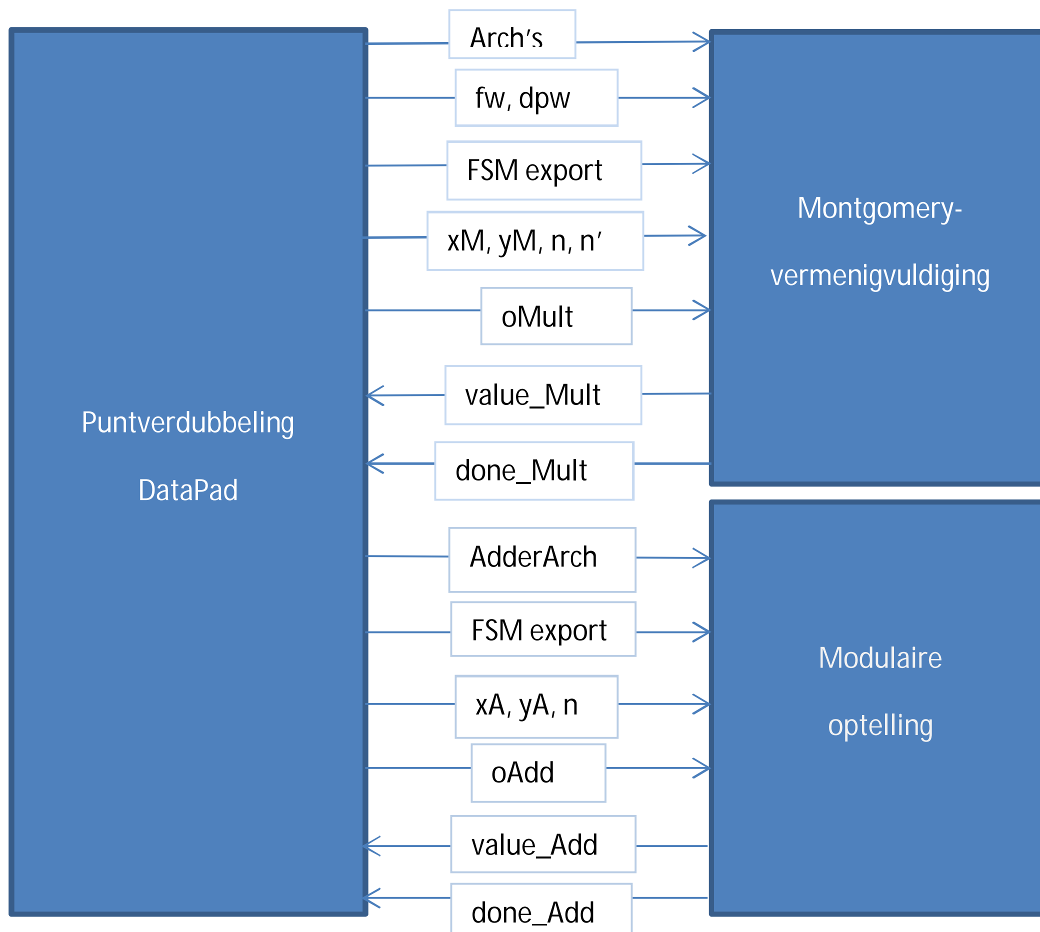


Figuur 10: Communicatie puntverdubbeling datapad-controlepad

Het controlepad heeft 4 input signalen (iStart, iCntr1Tec, iDone_Mult, iDone_Add) en 19 output signalen (oRead, oSub, oCntr1Clr, oMult, oAdd, oMux_Add_Mult, oDone, oWrite [1 .. 4], oAdrI [1 .. 4], oAdrO [1 .. 4]).

Waarvoor deze signalen gebruikt worden, zal besproken worden in de volgende twee secties.

3.2.2 Connecties met andere bewerkingen

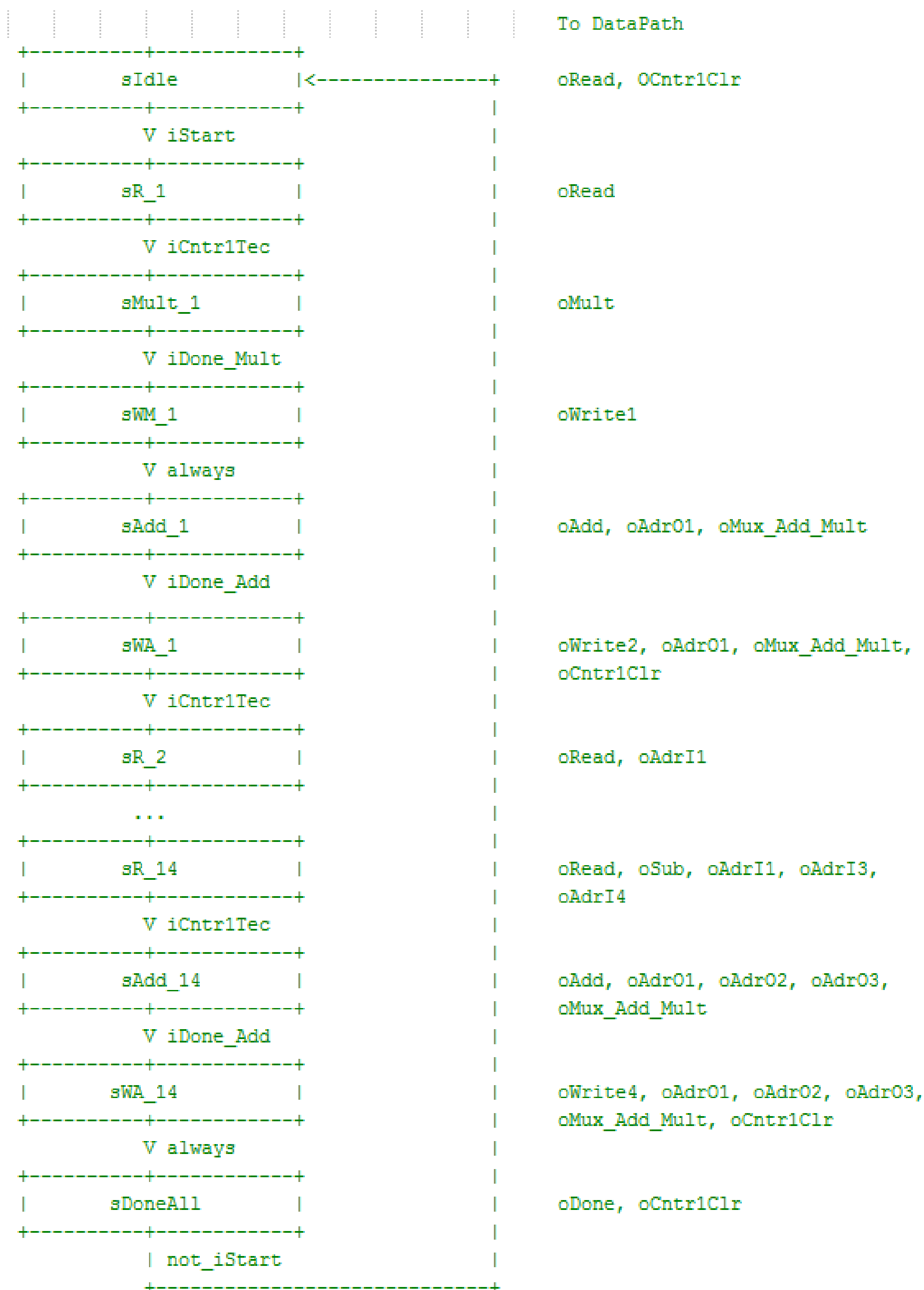


Figuur 11: Puntverdubbeling connecties met andere bewerkingen

De puntverdubbeling maakt gebruik van de Montgomery vermenigvuldiging en modulaire optelling. Hetgene hij stuurt naar die bewerkingen zijn: de architecturen waarvan de bewerkingen gebruik moeten maken (bv. rippleCarryAdder als AdderArch of CIOS voor MMArch), de field_width en datapath_width, de manier van FSM export (bv. NetList), de inputwaarden voor de bewerkingen (xM, Ym, n, …), en het signaal dat zegt dat de bewerking mag beginnen (oMult voor Montgomery, oAdd voor modulaire optelling). Wat de puntverdubbeling terugkrijgt is het resultaat van de bewerking (value_Mult voor Montgomery, value_Add voor modulaire optelling) en een signaal dat zegt wanneer de bewerking gedaan is (done_Mult voor Montgomery, done_Add voor modulaire optelling).

3.2.3 ControlePad

De werking van het puntverdubbeling controlepad:



Figuur 12: Puntverdubbeling controlepad

Er zijn in totaal 60 states: de ‘idle’ state en de ‘doneAll’ state voor het begin en einde (2 states), een ‘read’ state voor elke rij van het puntverdubbeling algoritme (14 states), een ‘mult’ state en ‘writeMult’ state voor elke modulaire vermenigvuldiging (2x8 = 16 states) en een ‘add’ state en ‘writeAdd’ state voor elke modulaire optelling (2x14 = 28 states).

De iStart input laat de bewerkingen van het algoritme beginnen vanuit de 'Idle' state. Wanneer de-bewerkingen gedaan zijn ('DoneAll' state) en iStart is low, keert het terug naar de 'Idle' state.
De iCntr1Tec zegt wanneer het inlezen van de termen voor de bewerkingen gedaan is.
De iDone_Mult en iDone_Add signalen zeggen wanneer een modulaire vermenigvuldiging of optelling gedaan is en men naar het wegschrijven van het resultaat kan overgaan.

De uitgaande 'adresOuput' en 'write' signalen worden bepaald aan de hand van het algoritme uit sectie 2.2.2. Deze bepalen in welk register de tussen- en eindresultaten van de modulaire bewerkingen worden opgeslagen. Meer hierover wordt verteld in volgende sectie. De 'adresInput' signalen vertellen aan het datapad welke termen moeten ingelezen worden voor de bewerkingen. Het 'oRead' signaal zegt dat data moet worden ingelezen en 'oSub' zegt dat er een aftrekking plaatsvindt i.p.v. een optelling. Het 'oCntr1Clr' signaal zet de teller voor de 'Read' states terug op nul. De 'Read' states hebben 2 cycli nodig voor het inlezen van de data en een teller wordt gebruikt om te zeggen wanneer deze cycli voorbij zijn. De 'oMult' en 'oAdd' signalen zeggen respectievelijk wanneer een modulaire vermenigvuldiging of optelling plaatsvindt. Het 'oMux_Add_Mult' signaal zegt bij het wegschrijven of het resultaat van een vermenigvuldiging of een optelling wordt weggeschreven en het 'oDone' signaal vertelt het datapad dat alle bewerkingen gedaan zijn.

3.2.4 DataPad

Register:

Er zijn tien tussenregisters waar de tussenresultaten van modulaire optellingen of vermenigvuldigingen worden opgeslaan. Dit wordt gedaan aan de hand van de write en adresOutput signalen van het ControlePad.

```
-- keuze tussen adder of multiplier waarde
value = mux1 oMux_Add_Mult value_Mult value_Add

-- de enables voor het register
oWrite = or2(oWrite1, or2(oWrite2, or2(oWrite3,oWrite4)))
en1  = and2(inv(oWrite), and2(inv(oAdr01), and2(inv(oAdr02), and2(inv(oAdr03), inv(oAdr04)))))
en2  = and2(inv(oWrite), and2(oAdr01,      and2(inv(oAdr02), and2(inv(oAdr03), inv(oAdr04)))))
en3  = and2(inv(oWrite), and2(inv(oAdr01), and2(oAdr02,      and2(inv(oAdr03), inv(oAdr04)))))
en4  = and2(inv(oWrite), and2(oAdr01,      and2(oAdr02,      and2(inv(oAdr03), inv(oAdr04)))))
en5  = and2(inv(oWrite), and2(inv(oAdr01), and2(inv(oAdr02), and2(oAdr03,      inv(oAdr04)))))
en6  = and2(inv(oWrite), and2(oAdr01,      and2(inv(oAdr02), and2(oAdr03,      inv(oAdr04)))))
en7  = and2(inv(oWrite), and2(inv(oAdr01), and2(oAdr02,      and2(oAdr03,      inv(oAdr04)))))
en8  = and2(inv(oWrite), and2(oAdr01,      and2(oAdr02,      and2(oAdr03,      inv(oAdr04)))))
en9  = and2(inv(oWrite), and2(inv(oAdr01), and2(inv(oAdr02), and2(inv(oAdr03),     oAdr04))))
en10  = and2(inv(oWrite),and2(oAdr01,      and2(inv(oAdr02), and2(inv(oAdr03),     oAdr04))))

-- het register
(t1, t2, t3, t4, t5, t6, t7, t8, t9, t10) = register value (en1, en2, en3, en4, en5, en6, en7, en8, en9, en10)
```

Figuur 13: Register van Datapad

De functie register plaatst de 'value' waarde in het juiste tussenregister met de 'delayEn' functie.

Input modulaire bewerkingen:

De input wordt gekozen uit de vier parameters (x,y,z,az^4) en de tussenregisters (t1 .. t10) aan de hand van de read en adresInput signalen van het ControlePad. Hiervoor is een extra functie gemaakt genaamd pointDoublingReadData.

```
pointDoublingReadData (oRead, oAdr1, oAdr2, oAdr3, oAdr4)
  (x1, y1, z1, az1, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11)
  = (r1,r2,r3,r4)
  where
    w1 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [y1,t1,t2,x1,y1,t3,t11,t11,t2,t11,t11,t11,t2,t11,t11,t11]
    w2 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [y1,t1,t1,x1,z1,az1,t11,t11,t2,t11,t11,t11,t1,t11,t11,t11]
    w3 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [x1,t2,t3,t3,t3,t2,t6,t2,t4,t1,t6,t1,t5,t2,t11,t11]
    w4 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [x1,t2,t3,t3,t3,t2,t2,az1,t4,t1,t4,t7,t5,t3,t11,t11]
    (r1,r2,r3,r4) = delayEn (y1,y1,x1,x1) oRead (w1,w2,w3,w4)
```

Figuur 14: Functie pointDoublingReadData

Deze functie wordt gebruikt in de verdubbeling functie. Daarnaast wordt het modulaire inverse genomen van de vierde inleeswaarde als er een aftrekking plaatsvindt, dit wordt bepaald door oSub van het ControlePad.

```
-- inlezen van de elementen
(r1, r2, r3, r4) = pointDoublingReadData (oRead, oAdrI1, oAdrI2, oAdrI3, oAdrI4)
  (x1, y1, z1, az1, rT1, rT2, rT3, rT4, rT5, rT6, rT7, rT8, rT9, rT10, t11)
r4_S = mux1 oSub r4 (modInverse r4 n low)
xM = mux1 oRead dataXM r1
yM = mux1 oRead dataYM r2
xA = mux1 oRead dataXA r3
yA = mux1 oRead dataYA r4_S
dataXM = delayEn zeroWord high xM
dataYM = delayEn zeroWord high yM
dataXA = delayEn zeroWord high xA
dataYA = delayEn zeroWord high yA
```

Figuur 15: Inlezen inputdata puntverdubbeling

Wegschrijven data:

Wanneer een modulaire optelling of vermenigvuldiging gedaan is wordt het resultaat weggeschreven. Dit wordt gedaan aan de hand van de vier write signalen van het ControlePad.

```
-- de enables voor het wegschrijven
wr1 = and2(oWrite1,       and2(inv(oWrite2), and2(inv(oWrite3), inv(oWrite4))))
wr2 = and2(inv(oWrite1), and2(oWrite2,       and2(inv(oWrite3), inv(oWrite4))))
wr3 = and2(oWrite1,       and2(oWrite2,       and2(inv(oWrite3), inv(oWrite4))))
wr4 = and2(inv(oWrite1), and2(inv(oWrite2), and2(oWrite3,       inv(oWrite4))))
wr5 = and2(oWrite1,       and2(inv(oWrite2), and2(oWrite3,       inv(oWrite4))))
wr6 = and2(inv(oWrite1), and2(oWrite2,       and2(oWrite3,       inv(oWrite4))))
wr7 = and2(oWrite1,       and2(oWrite2,       and2(oWrite3,       inv(oWrite4))))
wr8 = and2(inv(oWrite1), and2(inv(oWrite2), and2(inv(oWrite3),      oWrite4)))
wr9 = and2(oWrite1,       and2(inv(oWrite2), and2(inv(oWrite3),      oWrite4)))
wr10 = and2(inv(oWrite1),and2(oWrite2,       and2(inv(oWrite3),      oWrite4)))
```

Figuur 16: Enables voor wegschrijven data puntverdubbeling

Met deze enables wordt het resultaat weggeschreven naar de juiste register van de registers (rT1 .. rT10).

```
-- resultaat wegschrijven
rT1 = muxl wr1 dataT1 t1
rT2 = muxl wr2 dataT2 t2
rT3 = muxl wr3 dataT3 t3
rT4 = muxl wr4 dataT4 t4
rT5 = muxl wr5 dataT5 t5
rT6 = muxl wr6 dataT6 t6
rT7 = muxl wr7 dataT7 t7
rT8 = muxl wr8 dataT8 t8
rT9 = muxl wr9 dataT9 t9
rT10 = muxl wr10 dataT10 t10
dataT1 = delayEn zeroWord high rT1
dataT2 = delayEn zeroWord high rT2
dataT3 = delayEn zeroWord high rT3
dataT4 = delayEn zeroWord high rT4
dataT5 = delayEn zeroWord high rT5
dataT6 = delayEn zeroWord high rT6
dataT7 = delayEn zeroWord high rT7
dataT8 = delayEn zeroWord high rT8
dataT9 = delayEn zeroWord high rT9
dataT10 = delayEn zeroWord high rT10
```

Figuur 17: Wegschrijven data puntverdubbeling

Hierna kan deze waarde dan gebruikt worden voor de volgende modulaire bewerking.

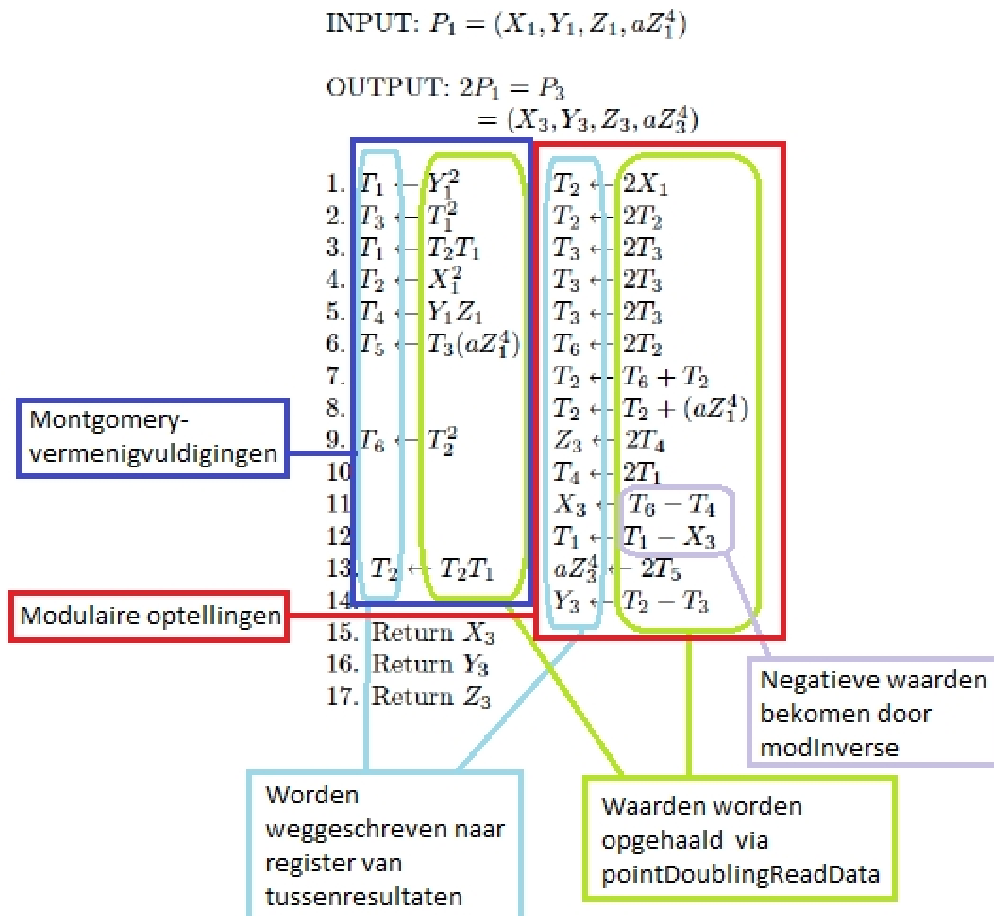
Modulaire inverse:

Voor de aftrekking wordt de modulaire inverse genomen van de tweede opteltterm. Dit wordt bereikt met behulp van een nieuwe functie 'modInverse'.

```
modInverse :: [Bit] -> [Bit] -> Bit -> [Bit]
modInverse [] [] cin = []
modInverse (b:bs) (n:ns) cin = (i:is)
  where
    [i] = muxlTree [b,n,cin] [[low],[high],[high],[low],[high],[low],[low],[high]]
    [cout] = muxlTree [b,n,cin] [[low],[high],[low],[low],[high],[high],[low],[high]]
    is = modInverse bs ns cout
```

Figuur 18: Modulaire inverse

De bedoeling van al deze code is ook weergegeven via Figuur 19 m.b.v het puntverdubbeling algoritme.



Figuur 19: Puntverdubbeling code uitgelegd a.d.h.v. het algoritme

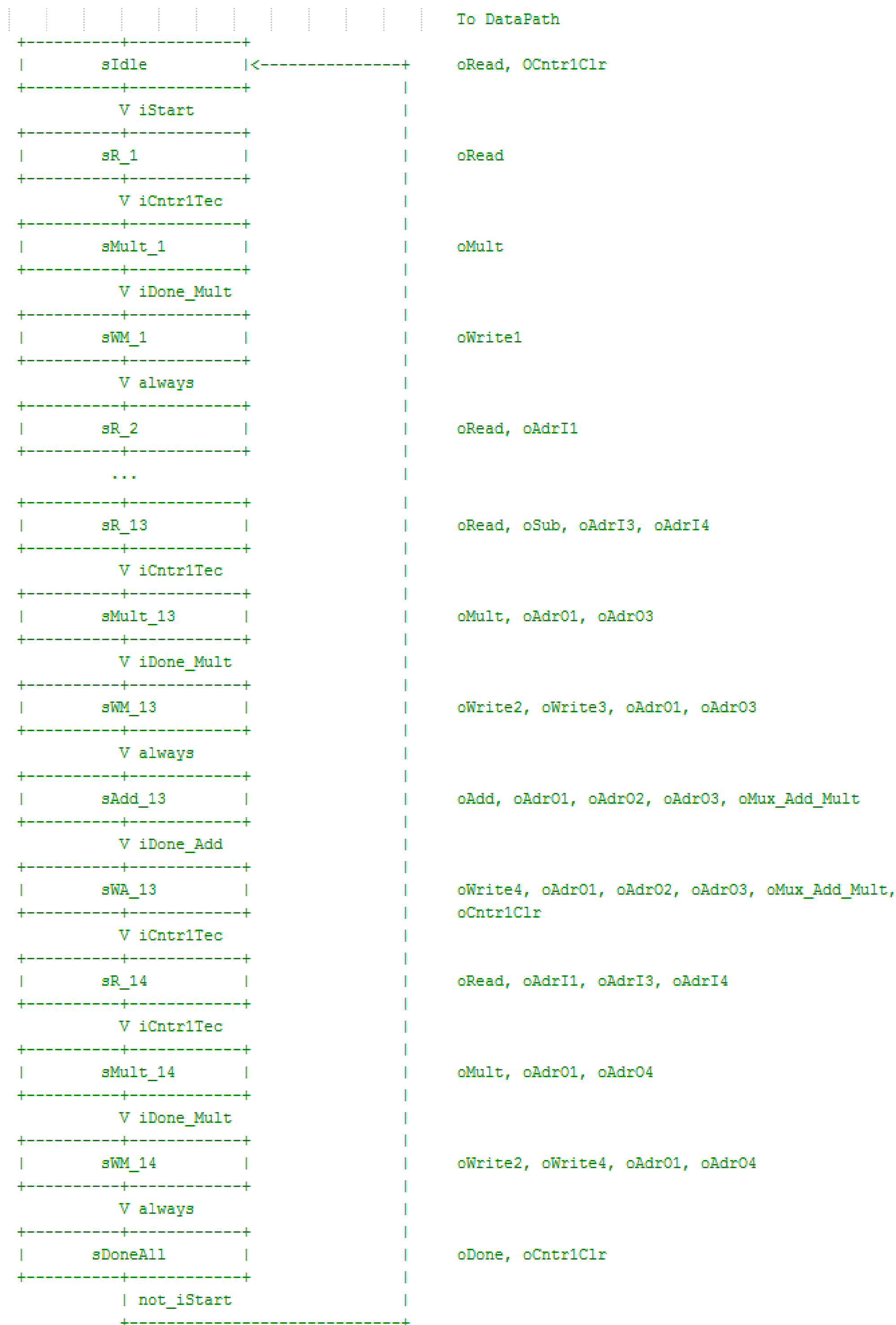
3.3 Puntoptelling

De puntoptelling is erg gelijkaardig aan de puntverdubbeling qua structuur. De communicatie tussen het controlepad en datapad is hetzelfde als bij de puntverdubbeling, alsook zijn de verbindingen met andere bewerkingen. Deze sectie gaat deze dus niet opnieuw tonen en gaat direct over naar de werking van het controlepad en datapad. De code staat in de bestanden PtOpt_elem_CP.hs en PtOpt_elem_DP.hs.

3.3.1 ControlePad

Het controlepad is erg gelijkaardig aan het controlepad van de puntverdubbeling. Het grootste verschil is dat er meer modulaire vermenigvuldigingen zijn dan optellingen. Bij de puntverdubbeling was het omgekeerde het geval. En daarnaast zijn de ‘write’ en ‘adresOutput’ signalen die worden gestuurd naar het datapad ook verschillend. Deze zijn bepaald aan de hand van het algoritme dat te zien was in sectie 2.2.2. Op het volgend blad vindt u het schema.

De werking van het puntoptelling controlepad:



Figuur 20: Puntoptelling controlepad

3.3.2 DataPad

Het datapad van de puntoptelling is weer gelijkaardig aan die van de puntverdubbeling. De enige echte grote verandering is de aanpassing in het inlezen van de data met de functie ‘pointAddingReadData’.

```
pointAddingReadData (oRead, oAdr1, oAdr2, oAdr3, oAdr4)
  (x1, y1, z1, az1, x2, y2, z2, az2, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11)
  = (r1,r2,r3,r4)
  where
    w1 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [z2,x1,t1,y1,t3,t2,t4,z2,t5,t1,t9,t5,t6,az1,t11,t11]
    w2 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [z2,t1,z2,t1,t3,t4,t3,t3,t5,t4,t9,t2,t6,t6,t11,t11]
    w3 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [t11,t11,x2,t11,y2,t11,t2,t4,t11,t3,t2,t11,t3,t11,t11,t11]
    w4 = mux1Tree [oAdr1,oAdr2,oAdr3,oAdr4]
      [t11,t11,t2,t11,t1,t11,t2,t6,t11,t6,t7,t11,t1,t11,t11,t11]
    (r1,r2,r3,r4) = delayEn (y1,y1,x1,x1) oRead (w1,w2,w3,w4)
```

Figuur 21: Functie pointAddingReadData

3.4 Puntvermenigvuldiging

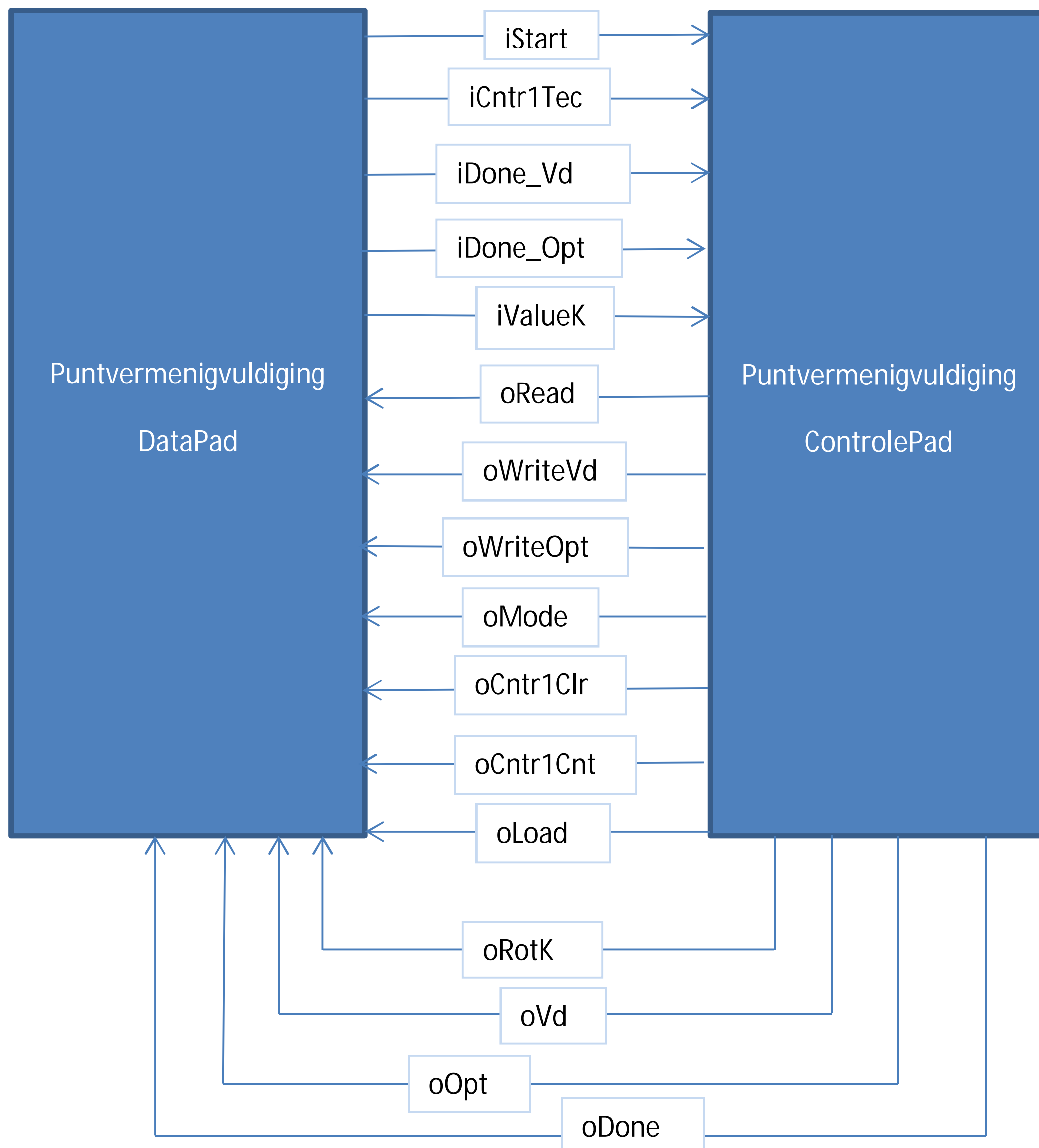
In deze sectie worden eerst de communicatie tussen het controlepad en datapad van de puntvermenigvuldiging getoond. Daarna wordt weer de werking van het controlepad en datapad uitgelegd, net zoals bij de vorige twee bewerkingen. De code staat in de bestanden PtVm_elem_CP.hs en PtVm_elem_DP.hs.

3.4.1 Signalen tussen DataPad en ControlePad

Het controlepad heeft 5 input signalen (iStart, iCntr1Tec, iDone_Vd, iDone_Opt, iValueK) en 11 output signalen (oRead, oWriteVd, oWriteOpt, oMode, oCntr1Clr, oCntr1Cnt, oLoad, oRotK, oVd, oOpt, oDone) .

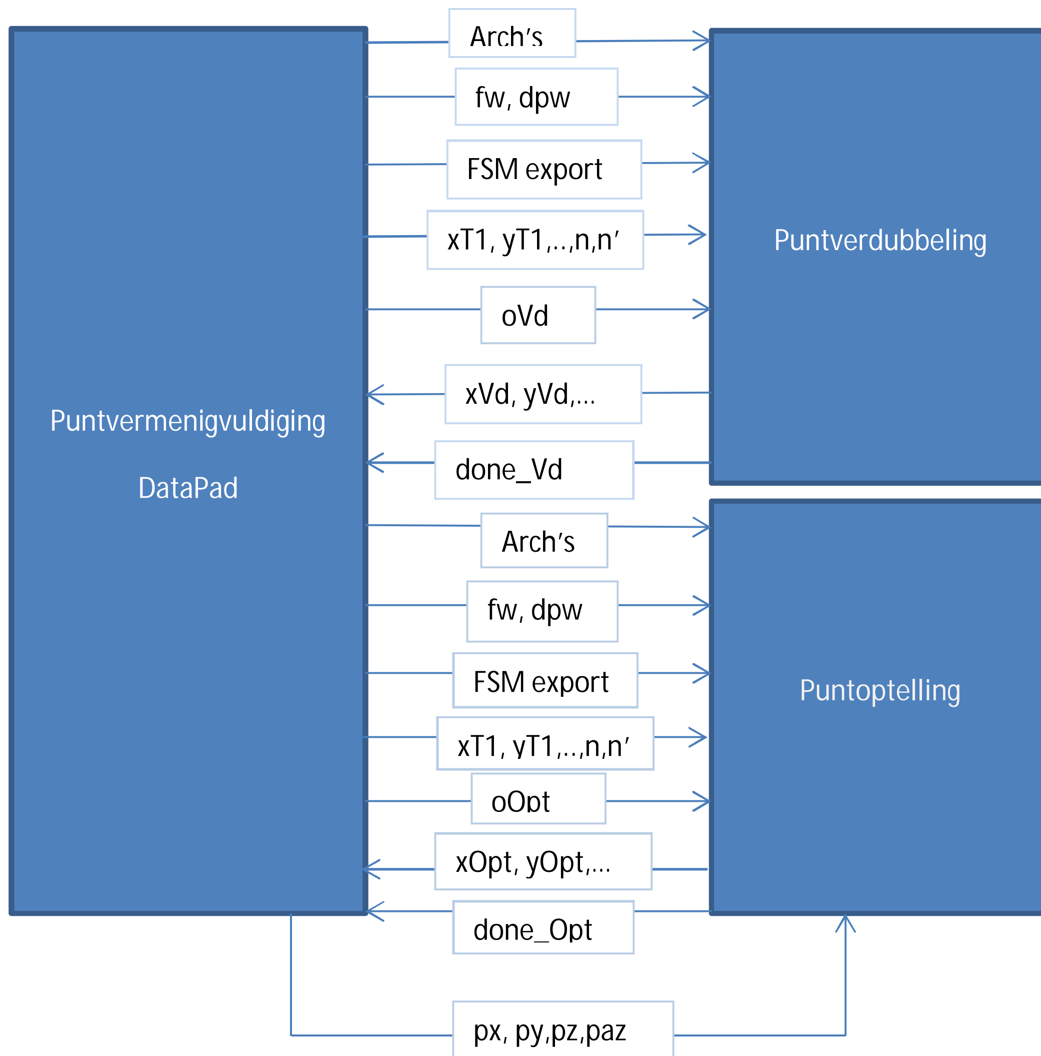
Deze wisselwerking wordt getoond in figuur 19.

Waarvoor deze signalen gebruikt worden, zal besproken worden in de volgende twee secties.



Figuur 22: Communicatie puntvermenigvuldiging datapad-controlepad

3.4.2 Connecties met andere bewerkingen

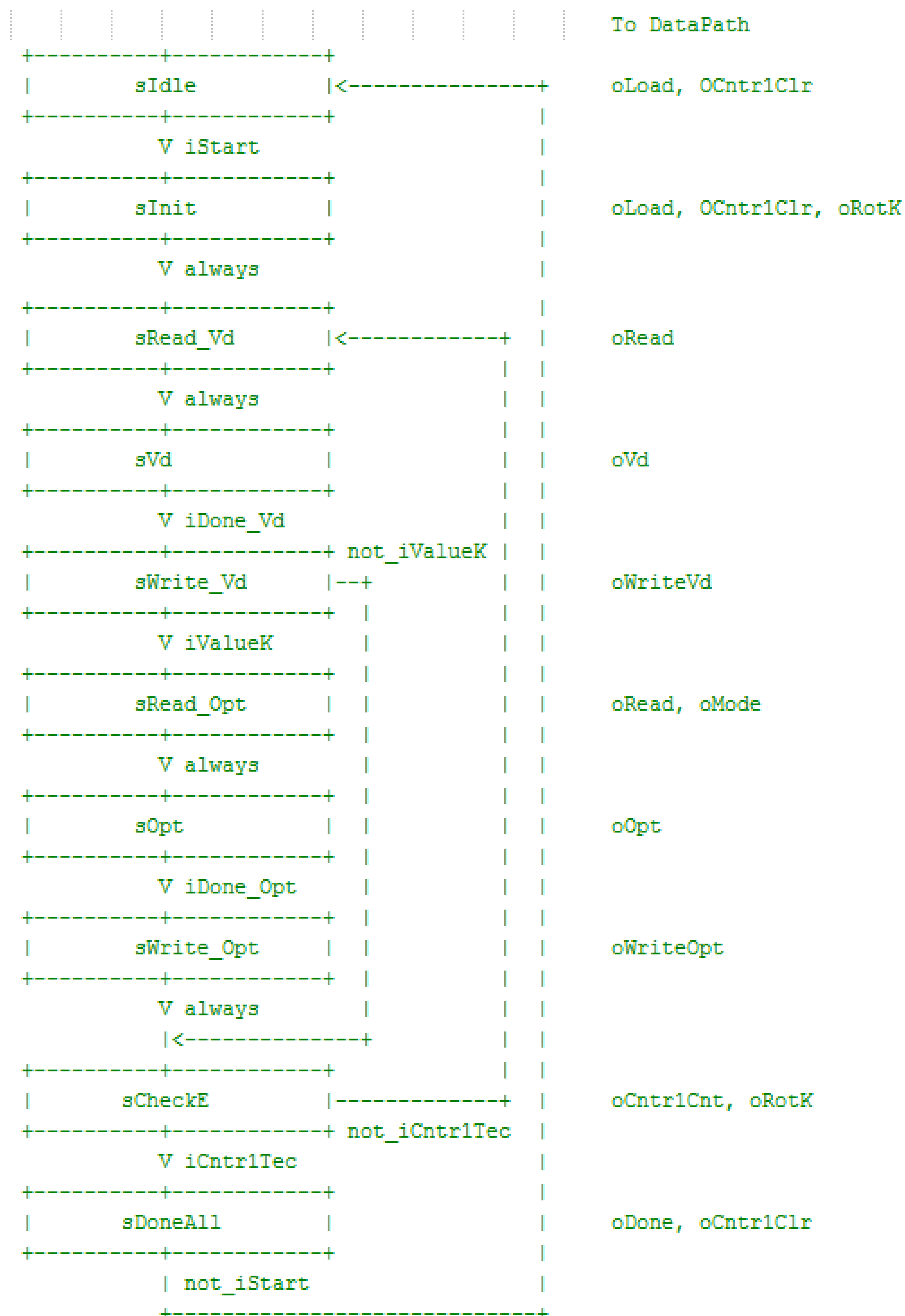


Figuur 23: Puntvermenigvuldiging connecties met andere bewerkingen

De waarden die puntvermenigvuldiging geeft aan de andere bewerkingen zijn nog altijd gelijkaardig aan de waarden bij puntverdubbeling. De verschillen zijn dat er nu meer inputwaarden zijn voor de bewerkingen nl. de vier coördinaten (xT1,yT1,zT1,azT1) van het tussenresultaat en de vier coördinaten (px, py, pz, paz) van het originele punt. Deze laatste vier coördinaten alleen bij de puntoptelling. Hetgene dat de puntvermenigvuldiging terugkrijgt zijn het resultaat van de bewerking en het signaal dat zegt dat de bewerking is afgelopen.

3.4.3 ControlePad

De werking van het puntvermenigvuldiging controlepad:



Figuur 24: Puntvermenigvuldiging controlepad

Er zijn in totaal 10 states. Dit is dus veel minder dan in de puntverdubbeling en –optelling controlepaden. Net zoals voorheen is er weer 'idle' en 'doneAll' state en de 'iStart' input die zegt wanneer ze naar de volgende state gaan. Dan is er een 'init' state waarin de input geladen wordt en de vermenigvuldigingsfactor k één keer geroteerd wordt. Waarom dit gedaan wordt, vindt u in de

sectie over het datapad. Dan is er een 'Vd' en 'Opt' state om de puntverdubbeling en –optelling uit te voeren. Er is ook een 'read' ('read_Vd' en 'read_Opt') en 'write ('write_Vd' en 'write_Opt') state voor en achter deze twee states voor het inlezen en wegschrijven van de data. De laatste nog niet besproken state is 'checkE'. In deze state wordt de factor k geroteerd en er wordt gecheckt of de lengte van k helemaal doorlopen is. Zo niet, gaat hij terug naar 'read_Vd'. Zo wel, gaat hij naar de 'doneAll' state. Dit laatste wordt gedaan met een teller dat aan het controlepad via de input 'iCntr1Tec' vertelt of die lengte doorlopen is. De 'iDone_Vd' en 'iDone_Opt' inputs vertellen wanneer de puntverdubbeling en puntoptelling gedaan zijn en de resultaten ervan mogen worden weggeschreven. De 'iValueK' input geeft de i-de MSB van de factor k. Afhankelijk van het feit of die high of low is, wordt er wel of niet een puntverdubbeling gedaan in de i-de stap van de lus van het puntvermenigvuldigingsalgoritme.
De 'oRead' output vertelt aan het datapad wanneer de data moeten worden ingelezen voor de bewerkingen. De 'oMode' output zegt of het inlezen voor een puntverdubbeling of voor een puntoptelling is. De 'oWriteVd' en 'oWriteOpt' outputs zeggen dat de resultaten van de respectievelijke bewerkingen mogen worden weggeschreven. De 'oVd' en 'oOpt' outputs zeggen aan het datapad dat die bewerkingen mogen uitgevoerd worden. 'oCntr1Clr' en 'oCntr1Cnt' zijn voor respectievelijk het leegmaken en het tellen van de teller van de lus van het algoritme. 'oLoad' is voor het inlezen van de vermenigvuldigingsfactor k en het punt P. 'oRotK' is voor het roteren van de factor k in het datapad. En 'oDone' is zoals bij vorige controlepaden om weer te geven dat alle bewerkingen gedaan zijn.

3.4.4 DataPad

<u>Vermenigvuldigingsfactor:</u>

De factor waarmee vermenigvuldigt wordt, wordt hier voorgesteld door de letter k en als input meegegeven. Er is een voorwaarde voor het algoritme dat zegt dat de MSB van het getal high moet zijn. Bij de huidige implementatie mag de meegegeven vermenigvuldigingsfactor daarom geen insignificante bits hebben. In het algoritme (te zien in sectie 2.2.4, Algoritme 6) is er een for lus waarbij de i-de term van de factor wordt gebruikt. In de code wordt dit gedaan door het getal te roteren na een stap van de lus en altijd de hoogste term te nemen.

```
dataKLD = muxl oLoad dataK k
[dataKMSB] = drop ((length dataKLD) - 1) dataKLD
dataKLast = init dataKLD
dataKNS = (dataKMSB:dataKLast)
dataKPD = muxl oRotK dataKLD dataKNS
dataK = delayEn k (oRotK) dataKPD
```

Figuur 25: De vermenigvuldigingsfactor

Om te weten wanneer de lus gedaan is, wordt een teller bijgehouden die stopt wanneer de lengte van de factor minus 2 doorlopen is. De minus 2 is door twee redenen nl.: om n termen te doorlopen moet (n-1) keer geteld worden en de MSB van het getal wordt niet gebruikt in het algoritme.

```
lengthK = (length k)
iCntr1Tec = stepCntr (lengthK - 2) (oCntr1Clr, oCntr1Cnt)
```

Figuur 26: Tellen tot lengte factor doorlopen

De bewerkingen:

In het algoritme wordt elke stap van de lus een puntverdubbeling (tussenresultaat*2) gedaan en als de i-de bit van de factor k high is wordt er ook een puntoptelling (tussenresultaat + input) gedaan. Deze bewerkingen zijn op volgende manier geïmplementeerd:

```
x_T1 = muxl oRead dataXT1 dataQX
y_T1 = muxl oRead dataYT1 dataQY
z_T1 = muxl oRead dataZT1 dataQZ
az_T1 = muxl oRead dataAZT1 dataQAZ
dataXT1 = delayEn zeroWord high x_T1
dataYT1 = delayEn zeroWord high y_T1
dataZT1 = delayEn zeroWord high z_T1
dataAZT1 = delayEn zeroWord high az_T1
(xVd, yVd, zVd, azVd, oRVd, oWVd, oMult_Vd, oAdd_Vd
  , done_Mult_Vd, done_Add_Vd, done_Vd) =
  verdubbeling adderArch mmArch multiplierArch fsmExport fw dpw
  (x_T1, y_T1, z_T1, az_T1, n, nAccent, oVd)
(xOpt, yOpt, zOpt, azOpt, oROpt, oWOpt, oMult_Opt, oAdd_Opt
  , done_Mult_Opt, done_Add_Opt, done_Opt) =
  optelling adderArch mmArch multiplierArch fsmExport fw dpw
  (px, py, pz, paz, x_T1, y_T1, z_T1, az_T1, n, nAccent, oOpt)
```

Figuur 27: Bewerkingen binnen puntvermenigvuldiging

In de puntoptelling wordt P als 1e opteltermgebruikt omdat volgens een voorwaarde van het puntoptellingsalgoritme de eerste term een één als z - coördinaat heeft en a als az - coördinaat.

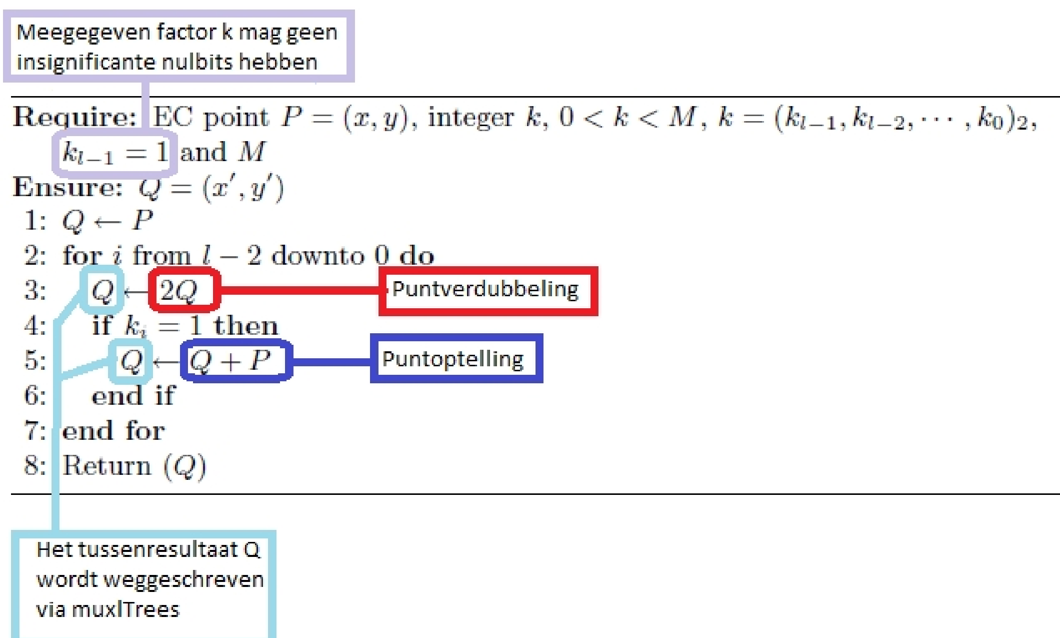
Inlezen en wegschrijven data:

Initieel worden de coördinaten van het input punt P (px, py, pz, paz) opgeslaan in de coördinaten van het punt Q (qx, qy, qz, qaz). Na elke bewerking (puntoptelling of –verdubbeling) word het resultaat van de bewerkingen opgeslaan in Q met behulp van de write enables: oWriteOpt en oWriteVd.

```
dataQXLD = muxl oLoad dataQX px
dataQXPD = muxlTree [oWriteVd,oWriteOpt] [dataQXLD, xVd, xOpt, dataQXLD]
dataQX = delayEn px (high) dataQXPD
qx = dataQX

dataQYLD = muxl oLoad dataQY py
dataQYPD = muxlTree [oWriteVd,oWriteOpt] [dataQYLD, yVd, yOpt, dataQYLD]
dataQY = delayEn py (high) dataQYPD
qy = dataQY

dataQZLD = muxl oLoad dataQZ pz
dataQZPD = muxlTree [oWriteVd,oWriteOpt] [dataQZLD, zVd, zOpt, dataQZLD]
dataQZ = delayEn pz (high) dataQZPD
qz = dataQZ

dataQAZLD = muxl oLoad dataQAZ paz
dataQAZPD = muxlTree [oWriteVd,oWriteOpt] [dataQAZLD, azVd, azOpt, dataQAZLD]
dataQAZ = delayEn paz (high) dataQAZPD
qaz = dataQAZ
```

Figuur 28: Inlezen en wegschrijven puntvermenigvuldiging

In Figuur 29 is de werking van deze code te zien t.o.v. het algoritme.

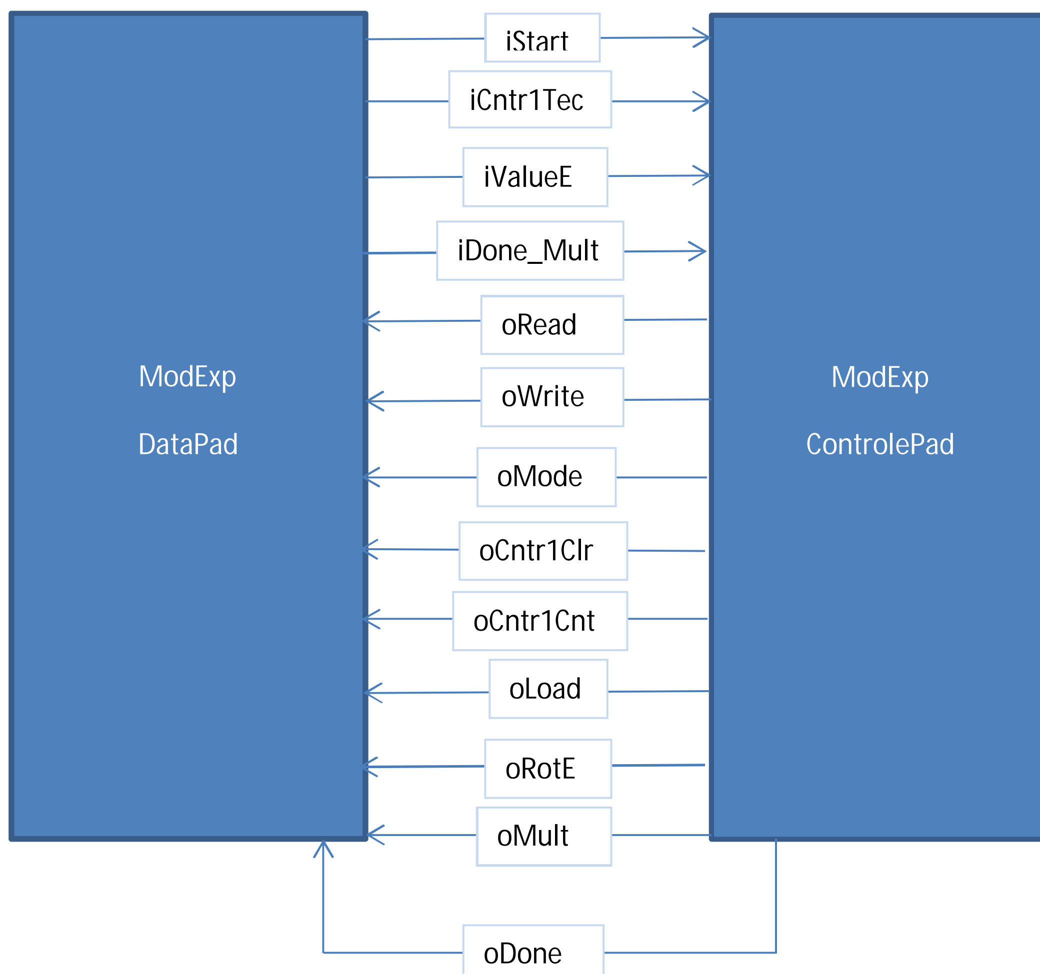


Figuur 29: Puntvermenigvuldiging code uitgelegd a.d.h.v. algoritme

3.5 Modulaire machtsverheffing

Modulaire machtsverheffing is (e-1) keren, met e een bepaalde macht, de Montgomery vermenigvuldiging uitvoeren. Deze modulaire machtsverheffing wordt ook gebruikt voor de transformatie van projectieve coördinaten naar affiene coördinaten, want bij die transformatie is R^(-1) nodig voor de berekeningen en R^(-1) is gelijk aan R^(p-2), met p de modulus. De code staat in de bestanden ModExp_CP.hs en ModExp_DP.hs.

3.5.1 Signalen tussen DataPad en ControlePad



Figuur 30: Communicatie modulaire machtsverheffing datapad-controlepad

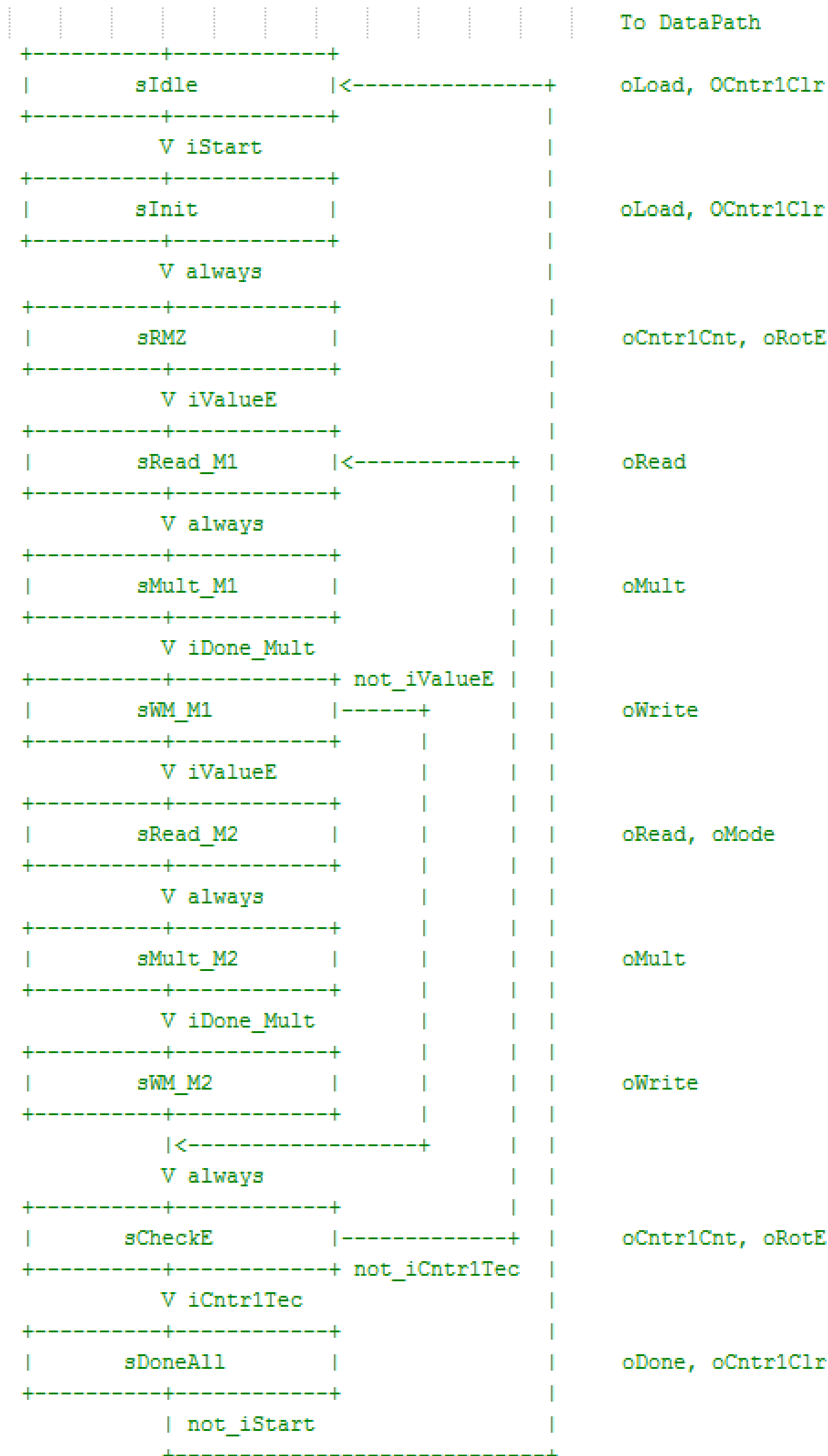
Het controlepad heeft 4 input signalen (iStart, iCntr1Tec, iValueE, iDone_Mult) en 9 output signalen (oRead, oWrite, oMode, oCntr1Clr, oCntr1Cnt, oLoad, oRotE, oMult, oDone) .

Dit wordt getoond in Figuur 30.

De modulaire machtsverheffing heeft ook een connectie met de Montgomery-vermenigvuldiging, maar deze is hetzelfde als de connectie van puntverdubbeling met deze vermenigvuldiging. Dus deze connectie wordt niet opnieuw getoond.

Zoals voorheen zal dit besproken worden in de volgende twee secties.

3.5.2 ControlePad



Figuur 31: ModExp controlepad

Er zijn in totaal 11 states. Er is een 'init' state waarin de input ingeladen wordt. Dan is er een 'Mult_M1' en 'Mult_M2' state om twee verschillende montgomery vermenigvuldigingen uit te voeren. De eerste vermenigvuldigt het tussenresultaat met zichzelf en de tweede vermenigvuldigt

het tussenresultaat met de originele input waarde. Voor de rest is dit controlepad erg gelijkaardig aan het puntvermenigvuldiging controlepad en daarom zal niet alles terug opnieuw worden uitgelegd. Er is wel nog het verschil dat er een 'RMZ' state is waar de macht e verschillende keren geroteerd wordt voor de berekeningen gebeuren. Dit is omdat in het algoritme de voorwaarde wordt gesteld dat de macht e geen overbodige zerobits heeft in zijn MSB. In het controlepad van de ModExp worden die dan weggewerkt in de 'RMZ' stat. Bij de puntvermenigvuldiging wordt dit niet gedaan en mag de ingegeven vermenigvuldigingsfactor geen overbodige zerobits hebben.

3.5.3 DataPad

Exponent:

De exponent e wordt erg gelijkaardig behandeld als de vermenigvuldigingsfactor k bij de puntvermenigvuldiging.

```
dataELD = muxl oLoad dataE e
[dataEMSB] = drop ((length dataELD) - 1) dataELD
dataELast = init dataELD
dataENS = (dataEMSB:dataELast)
dataEPD = muxl oRotE dataELD dataENS
dataE = delayEn e (high) dataEPD
```

Figuur 32: De exponent

Bewerkingen:

Er worden twee verschillende soorten Montgomery vermenigvuldigingen uitgevoerd: vermenigvuldiging van tussenresultaat met zichzelf en vermenigvuldigd met de originele waarde. De vermenigvuldigingstermen zijn xM en yM. De xM term is altijd het tussenresultaat en de yM term is ofwel het tussenresultaat ofwel de originele waarde. Dit wordt gekozen m.b.v. een muxTree.

```
xM = muxl oRead dataXM dataA
yM = muxlTree [oRead,oMode] [dataYM,dataA,dataYM,m]
dataXM = delayEn zeroWord high xM
dataYM = delayEn zeroWord high yM
(value_Mult, rotX, rotY, rotN, done_Mult) =
  montgomeryMultiplierCore fw dpw mmArch adderArch
  multiplierArch fsmExport (xM, yM, n, nAccent, oMult)
```

Figuur 33: Bewerkingen modulaire machtsverheffing

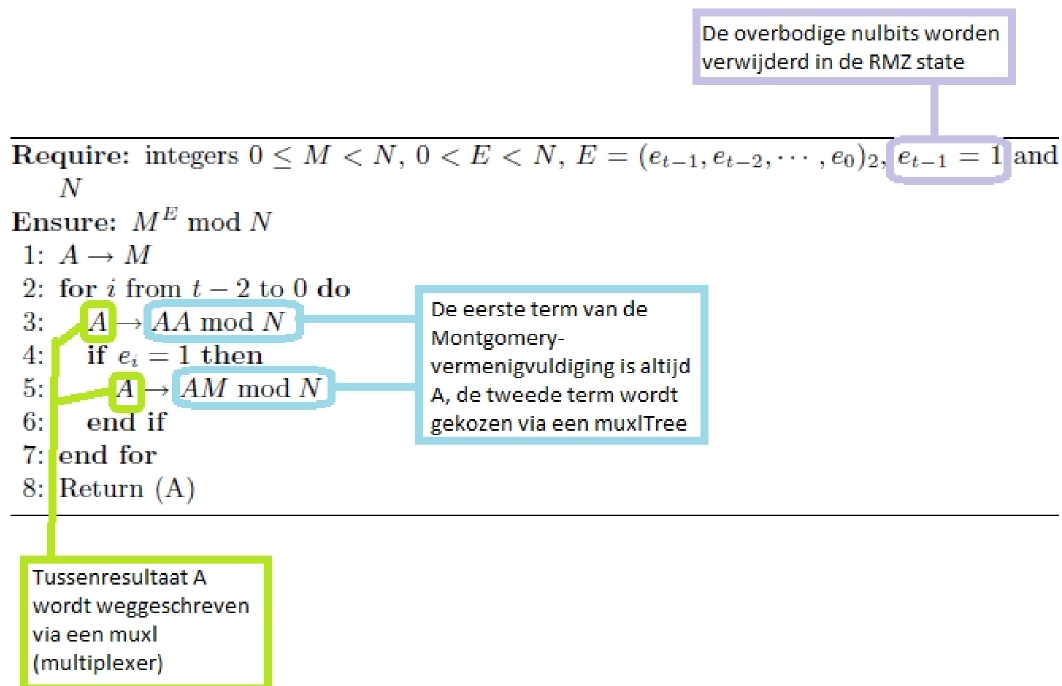
Inlezen en wegschrijven:

Het tussenresultaat 'a' is in het begin de originele waarde 'm' en elke keer als er een vermenigvuldiging gedaan is, wordt het resultaat van die bewerking opgeslagen in 'a'.

```
dataALD = muxl oLoad dataA m
dataAPD = muxl oWrite dataALD value_Mult
dataA = delayEn m (high) dataAPD
a = dataA
```

Figuur 34: Inlezen en wegschrijven modulaire machtsverheffing

In Figuur 35 is de werking van de code te zien t.o.v. het modulaire machtsverheffing algoritme.

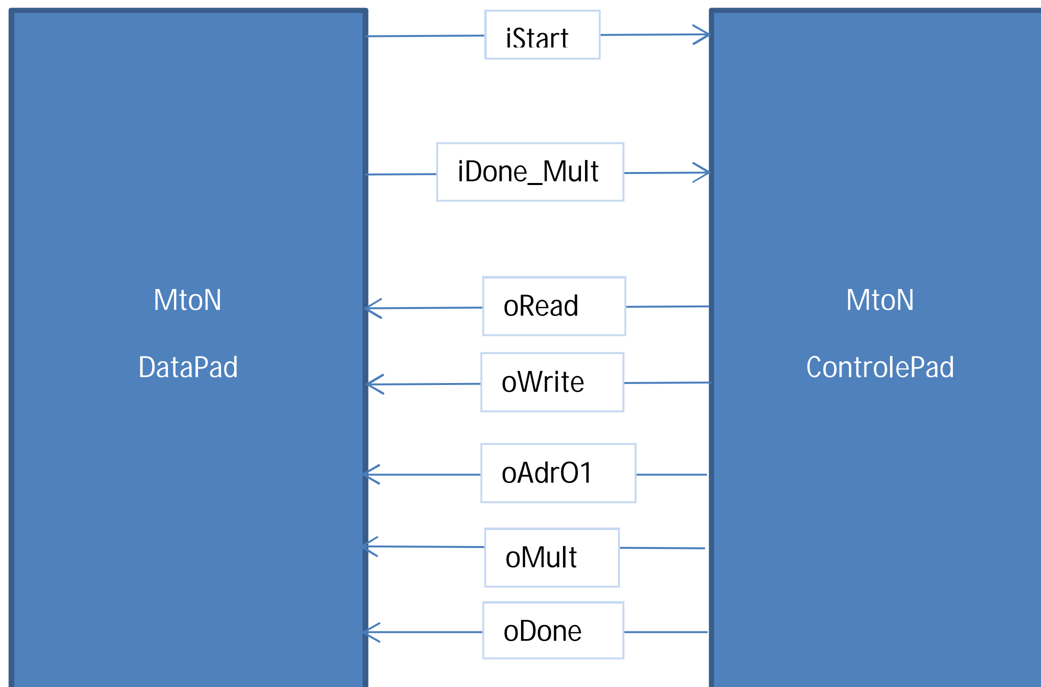


Figuur 35: ModExp code uitgelegd a.d.h.v. het algoritme

3.6 NtoM en MtoN transformaties

Deze sectie gaat over de Normaal-naar-Montgomery en Montgomery-naar-Normaal transformaties. Deze twee zijn erg gelijkaardig en daarom worden ze in één sectie gestoken. Deze transformaties hebben ook een connectie met de Montgomery vermenigvuldiging, maar deze is al eens getoond bij puntverdubbeling. Dus deze wordt net zoals bij modulaire machtsverheffing niet opnieuw getoond. De code staat in de bestanden NtoM_elem_CP.hs, NtoM_elem_DP.hs, MtoN_elem_CP.hs en MtoN_elem_DP.hs.

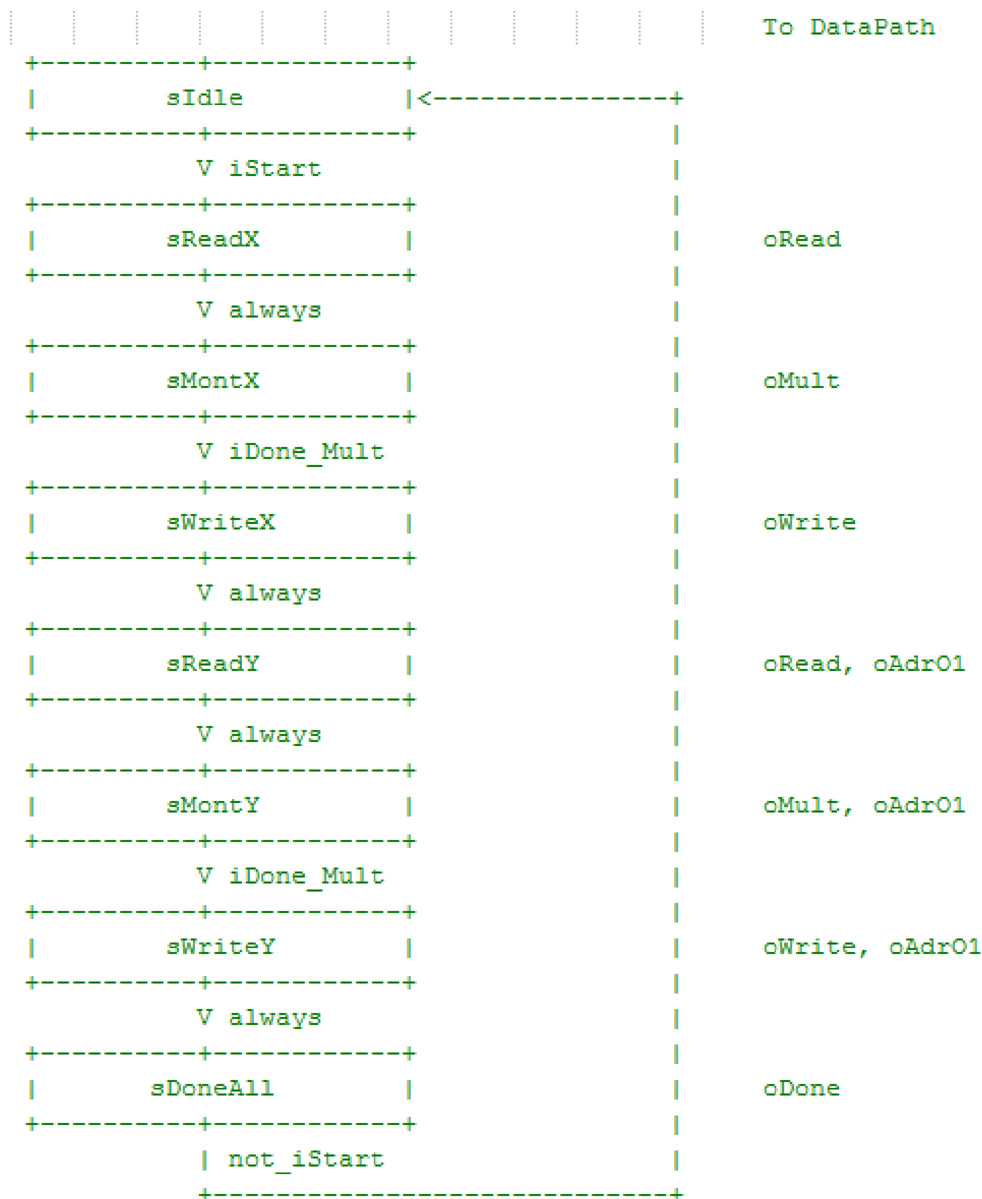
3.6.1 Signalen tussen DataPad en ControlePad



Figuur 36: Communicatie MtoN transformatie datapad-controlepad

Het MtoN controlepad heeft 2 input signalen (iStart, iDone_Mult) en 5 output signalen (oRead, oWrite, oAdrO1, oMult, oDone) . Bij NtoM is er één output signaal meer nl. oAdrO2. Dit is omdat er bij NtoM vier coördinaten omgezet worden en bij MtoN maar 2 coördinaten. Daarom is er een extra signaal nodig voor wegschrijven en inlezen.

3.6.2 ControlePad



Figuur 37: MtoN transformatie controlepad

Er zijn in totaal 8 states voor MtoN en 14 states voor NtoM. Deze states omvatten de altijd aanwezige 'Idle' en 'DoneAll' states en een 'Read', 'Mont' en 'Write' state voor elke coördinaat. De 'Mont' state is de state waarin de Montgomery vermenigvuldiging plaatsvindt.

3.6.3 DataPad

Het datapad is redelijk simpel in beide gevallen. Telkens een coördinaat inlezen, Montgomery vermenigvuldiging uitvoeren en het resultaat wegschrijven naar de nieuwe coördinaat. Een verschil is wel dat bij NtoM een extra input nodig is nl. R^2. Dit is nodig voor de Montgomery vermenigvuldigingen die in deze transformatie worden uitgevoerd.

```
sw = div fw dpw
zeroWord = zeroList (sw*dpw)

en1 = and2(oWrite, and2(inv(oAdrO1), inv(oAdrO2)))
en2 = and2(oWrite, and2(oAdrO1, inv(oAdrO2)))
en3 = and2(oWrite, and2(inv(oAdrO1), oAdrO2))
en4 = and2(oWrite, and2(oAdrO1, oAdrO2))

-- resultaat wegschrijven
qx = muxl en1 dataT1 value_Mult
qy = muxl en2 dataT2 value_Mult
qz = muxl en3 dataT3 value_Mult
qaz = muxl en4 dataT4 value_Mult
dataT1 = delayEn zeroWord high qx
dataT2 = delayEn zeroWord high qy
dataT3 = delayEn zeroWord high qz
dataT4 = delayEn zeroWord high qaz

dataX = muxlTree [oAdrO1,oAdrO2] [px,py,pz,paz]

(value_Mult, rotX, rotY, rotN, done_Mult) =
  montgomeryMultiplierCore fw dpw mmArch adderArch
  multiplierArch fsmExport (dataX, rsquare, n, nAccent, oMult)
```

Figuur 38: NtoM datapad code

3.7 AtoP en PtoA transformaties

In deze sectie worden de overige twee transformaties besproken nl. de Affiene-naar-Projectieve en de Projectieve-naar-Affiene transformaties. De code staat in de bestanden AtoP_elem.hs, PtoA_elem_CP.hs en PtoA_elem_DP.hs.

3.7.1 AtoP transformatie

Deze transformatie is heel kort uitgelegd. De input zijn de affiene coördinaten x en y en de parameter 'a' uit de affiene vergelijking van de curve. De output zijn de projectieve coördinaten X, Y, Z en aZ^4, met X=x, Y=y, Z=1 en aZ^4=a. Waarom dit laatste zo is, is terug te vinden in de sectie 1.2.6.1.

```
toProjective :: Int                              -- ^ fw
             -> Int                              -- ^ dpw
             -> ([Bit], [Bit], [Bit])            -- ^ (ax, ay, a)
             -> ([Bit], [Bit], [Bit], [Bit])     -- ^ (px, py, pz, paz)

toProjective fw dpw (ax, ay, a) = (px, py, pz, paz)
  where

    sw = div fw dpw
    zeroWord = zeroList (sw*dpw)
    zeroWordM = zeroList ((sw*dpw)-1)
    oneM = (high:zeroWordM)

    px = ax
    py = ay
    pz  = oneM
    paz = a
```

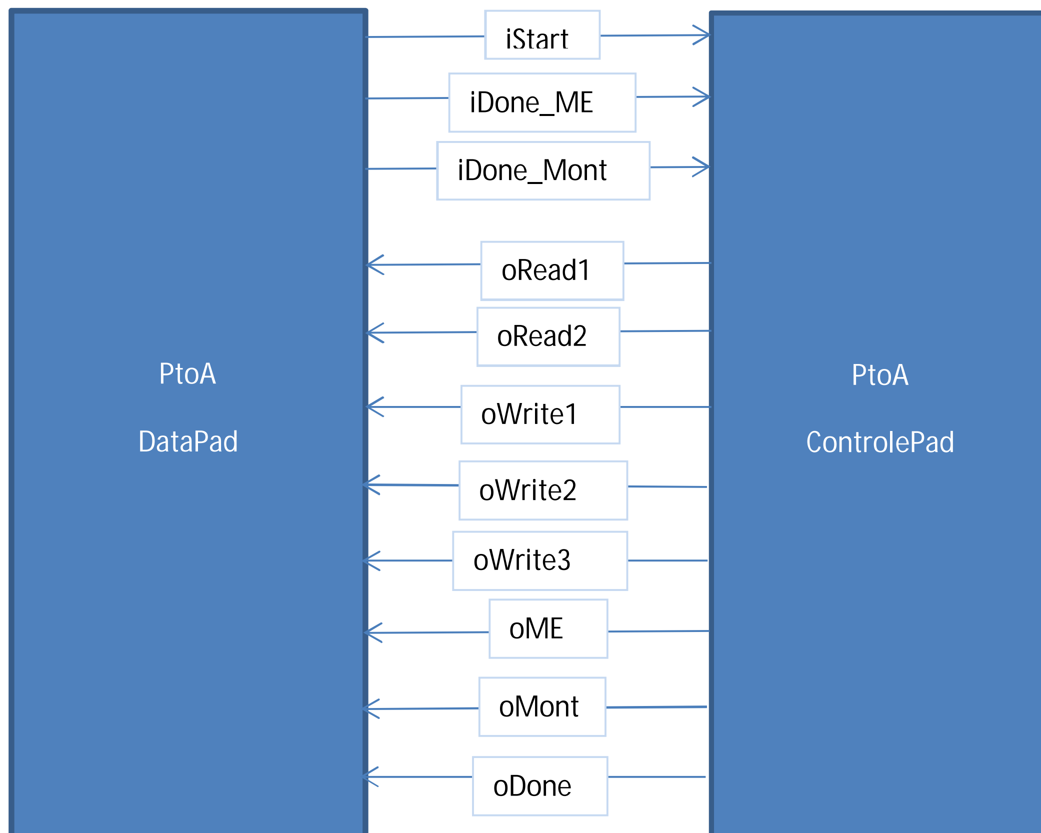
Figuur 39: AtoP transformatie code

3.7.2 PtoA transformatie

Deze transformatie is ingewikkelder dan de vorige. Hierbij worden 4 Montgomery vermenigvuldigingen en 1 modulaire machtsverheffing gebruikt.

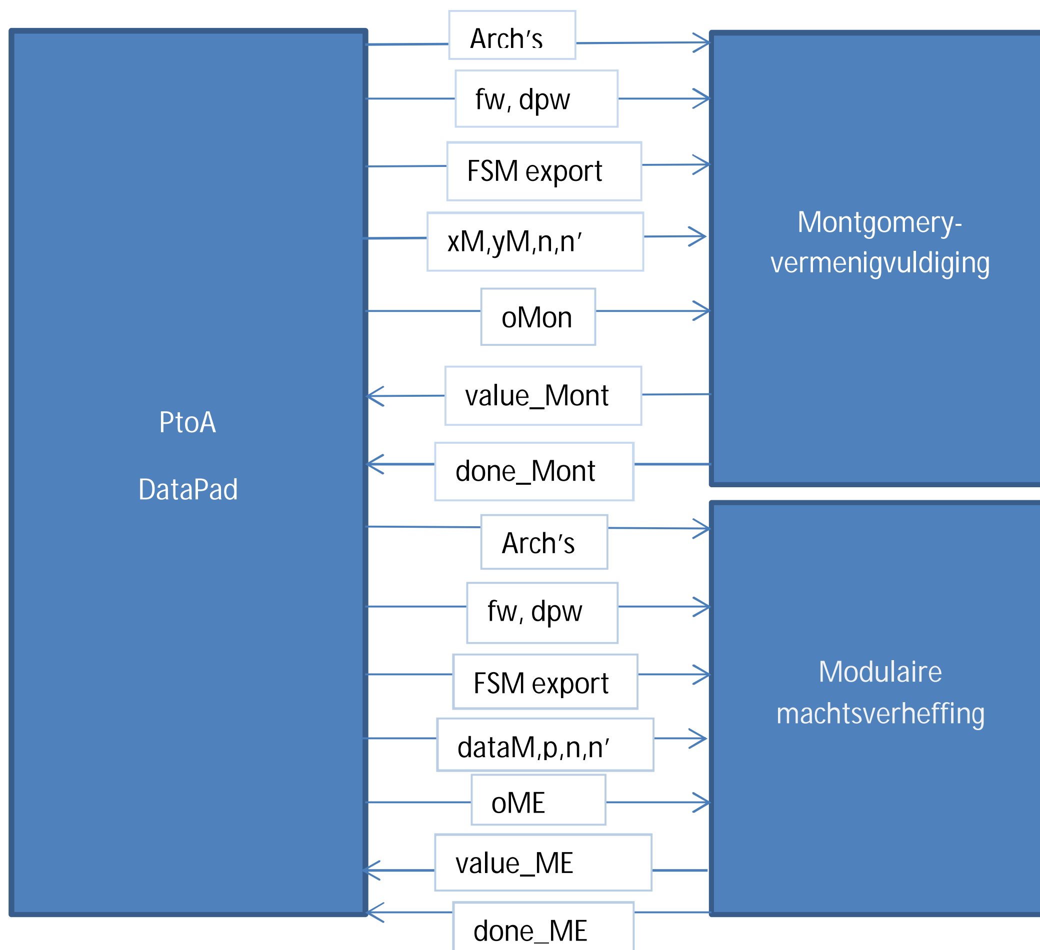
3.7.2.1 Signalen tussen DataPad en ControlePad

Het PtoA controlepad heeft 3 input signalen (iStart, iDone_ME, iDone_Mont) en 8 output signalen (oRead1, oRead2, oWrite1, oWrite2, oWrite3, oME, oMult, oDone) .



Figuur 40: Communicatie PtoA transformatie datapad-controlepad

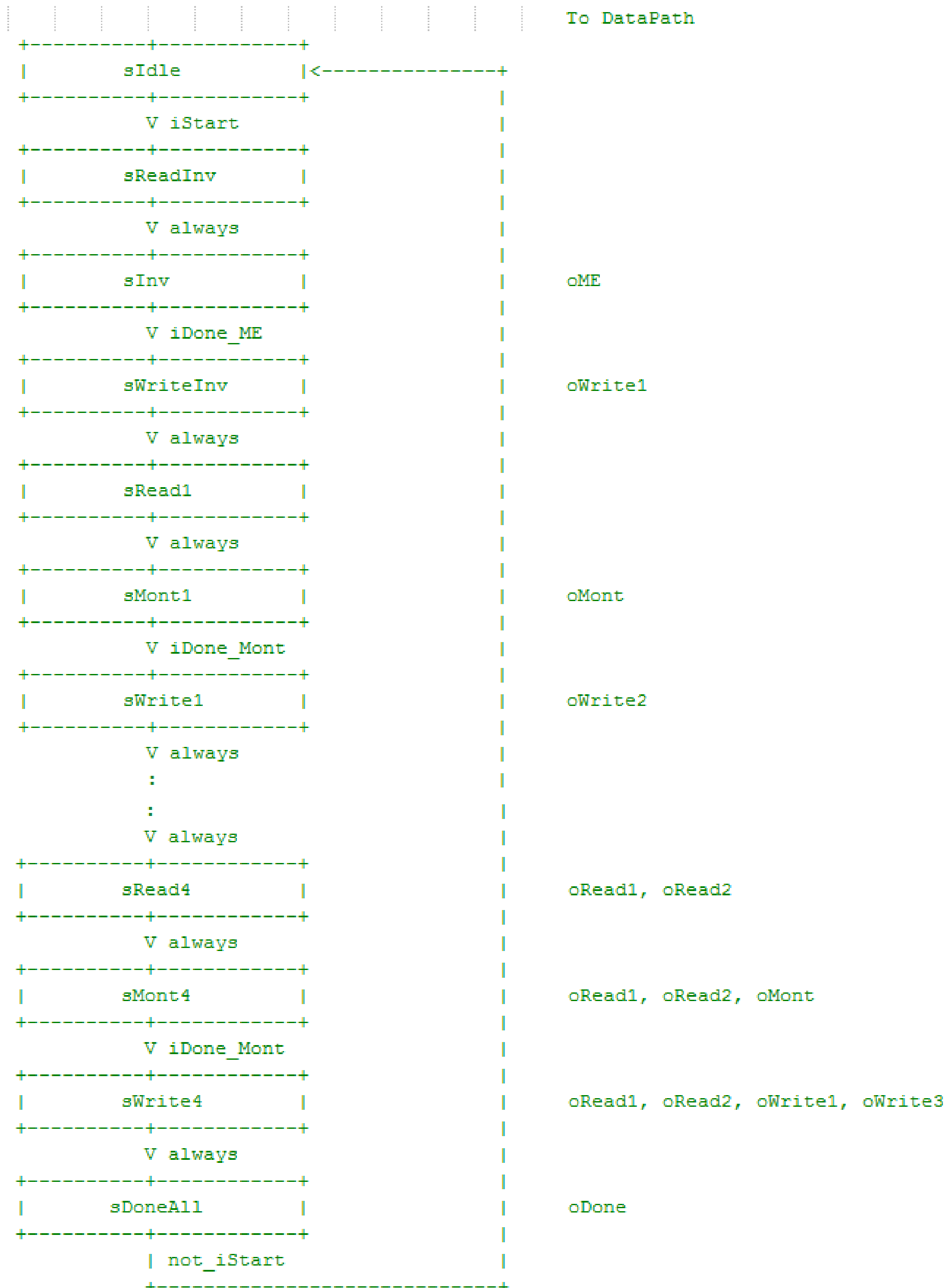
3.7.2.2 Connecties met andere bewerkingen



Figuur 41: PtoA connecties met andere bewerkingen

De PtoA transformatie heeft een connectie met de Montgomery-vermenigvuldiging en met de modulaire machtsverheffing. Deze is weer gelijkaardig als in vorige gevallen met uitzondering op de inputwaarden van de modulaire machtsverheffing. Deze krijgt als input de waarde waarop men een machtsverheffing wilt uitvoeren 'dataM', de exponent 'p', de modulus n en inverse modulus n'.

3.7.2.3 ControlePad



Figuur 42: PtoA transformatie controlepad

Er zijn in totaal 17 states voor PtoA. Deze states omvatten de 'Idle' en 'DoneAll' states en een 'Read', ('Mont'/'ME') en 'Write' state voor elk van de vijf bewerkingen. Deze bewerkingen zijn de modulaire machtsverheffing en de vier Montgomery vermenigvuldigingen. De 'Mont' state is de state waarin de Montgomery vermenigvuldiging plaatsvindt en de 'ME state is de state waarin de modulaire machtsverheffing gebeurt.

3.7.2.4 DataPad

Inlezen data:

De projectieve Z-coordinaat wordt gebruikt als input voor de modulaire machtsverheffing. De macht p (= n-2, met n de modulus) moet op het moment ook meegegeven worden als input. Dit kan misschien nog veranderd worden door deze aftrekking mee in deze transformatie te implementeren. Het nadeel van de aftrekking implementeren zou zijn dat deze bewerking dan meer iteraties nodig heeft om te voltooien. Het voordeel is dat je de factor p niet zelf moet berekenen. De waardes die aan de vier Montgomery vermenigvuldigingen worden meegegeven zijn zoals te zien in sectie 1.2.6.3.

```
dataM = pz
dataXM = mux1Tree [oRead1,oRead2] [dataZ1,dataZ1,px,py]
dataXMT = delayEn zeroWord high dataXMT
dataYM = mux1Tree [oRead1,oRead2] [dataZ1,dataZ2,dataZ2,dataZ3]
dataYMT = delayEn zeroWord high dataYMT
```

Figuur 43: Inlezen data PtoA transformatie

Wegschrijven data:

Er worden 5 enables (wrz1, wrz2, wrz3, wrx, wry) gemaakt, één voor elk van de vijf bewerkingen. Deze enables worden gebruikt om de resultaten van de bewerkingen weg te schrijven naar de tussenresultaten (z1T, z2T, z3T) en de uiteindelijke resultaten ax en ay. De ax en ay waarden zijn de affiene coördinaten.

```
wrz1 = and2(oWrite1, and2(inv(oWrite2), inv(oWrite3)))
wrz2 = and2(inv(oWrite1), and2(oWrite2, inv(oWrite3)))
wrz3 = and2(oWrite1, and2(oWrite2, inv(oWrite3)))
wrx = and2(inv(oWrite1), and2(inv(oWrite2), oWrite3))
wry = and2(oWrite1, and2(inv(oWrite2), oWrite3))

dataZ1 = mux1 wrz1 z1T value_ME
z1T = delayEn zeroWord (high) dataZ1
dataZ2 = mux1 wrz2 z2T value_Mont
z2T = delayEn zeroWord (high) dataZ2
dataZ3 = mux1 wrz3 z3T value_Mont
z3T = delayEn zeroWord (high) dataZ3
dataX = mux1 wrx xT value_Mont
xT = delayEn zeroWord (high) dataX
ax = xT
dataY = mux1 wry yT value_Mont
yT = delayEn zeroWord (high) dataY
ay = yT
```

Figuur 44: Wegschrijven data PtoA transformatie

Bewerkingen:

Er zijn 5 bewerkingen: de modulaire machtsverheffing en de vier Montgomery vermenigvuldigingen.

```
(value_ME, oRead_ME, oWrite_ME, oMode_ME, oLoad_ME, oRotE
  , oMult, done_Mult, done_ME) =
  machtverheffing adderArch mmArch multiplierArch fsmExport
  fw dpw (dataM, p, n, nAccent, oME)
(value_Mont, rotX, rotY, rotN, done_Mont) =
  montgomeryMultiplierCore fw dpw mmArch adderArch multiplierArch
  fsmExport (dataXM, dataYM, n, nAccent, oMont)
```

Figuur 45: Bewerkingen PtoA transformatie

4 Testen

Om de Lava code te testen heb ik gebruikt gemaakt van de magma calculator, terug te vinden op de site http://magma.maths.usyd.edu.au/calc/. Hierop kon ik zien wat de juiste resultaten waren van de Montgomery vermenigvuldigingen en puntbewerkingen en kon ik zo testen of de geïmplementeerde code dezelfde resultaten geeft. In volgende secties wordt één voorbeeld gebruikt om de bewerkingen te simuleren. De bewerkingen zijn doorheen het project voor meerdere voorbeelden gesimuleerd, maar voor dit hoofdstuk zal alleen één simpel voorbeeld uitgelegd worden.

4.1 Montgomery-vermenigvuldiging

Eerst worden twee willekeurige waarden genomen, bv. 5 en 4, en een modulus bv. 13. In dit voorbeeld is de veldlengte van de waarden 4 en de woordbreedte 1. In de magma calculator worden de commando's ingegeven die te zien zijn in de Appendix. Dit is om de Montgomery-waarden en het resultaat te vinden die dan worden gebruikt om de Lava-code te testen.

De waarde p is de modulus, in ons geval 13 zoals te zien in de Appendix, en wordt geschreven in machten van 2. De modulus moet een priemgetal zijn. Daarna wordt het veld F gemaakt om te gebruiken in volgende vergelijkingen om altijd modulus p te bekomen van de resultaten. R is 2 tot de macht (veldlengte) en w is de woordlengte. Daarnaast zijn ook de inverse van R en p nodig en het kwadraat van R voor de bewerkingen. De Montgomery waarde van een getal is dit getal maal R. En om een Montgomery om te zetten naar de normale waarde, wordt de waarde vermenigvuldigd met Rinv. Voor de Montgomery vermenigvuldiging te vergelijken wordt SM= AM*BM*Rinv berekend. AM en BM zijn de waarden die moeten ingegeven worden in de Montgomery vermenigvuldiging Lava code. En SM is het resultaat dat uit die code moet worden bekomen. S is de normale waarde van dat resultaat. Stest is gewoon om aan te tonen dat het resultaat S uit de Montgomery vermenigvuldiging hetzelfde is als de gewone vermenigvuldiging a*b.

Uit de resultaten van de calculator is te zien dat AM=2, BM=12, p=13 en p_inv=1. Dit wordt gebruikt voor de input. Er is ook te zien dat SM=8 en S=7. Dit is om met de output te vergelijken. In de Lava code is er een testBox_CIOS.hs file aangemaakt waarin het volgende is geschreven:

```
zero  = [low,low,low,low]
one = [high,low,low,low]
two   = [low,high,low,low]
three = [high,high,low,low]
four = [low,low,high,low]
five  = [high,low,high,low]
six = [low,high,high,low]
seven = [high,high,high,low]
eight = [low,low,low,high]
nine = [high,low,low,high]
ten = [low,high,low,high]
eleven = [high,high,low,high]
twelve = [low,low,high,high]
thirteen = [high,low,high,high]
fourteen = [low,high,high,high]
fifteen = [high,high,high,high]
```

Figuur 46: Definitie getallen in bits

Dit stuk code is om de getallen in bit-formaat te definiëren. De getallen zijn geordend met minst significante bit eerst. Dit noemt little-endian [53]. Gewoonlijk werken we volgens big-endian met minst significante bit laatst, maar de bewerkingen die al geïmplementeerd waren in de tool werkten volgens little-endian. Daarom zijn de nieuwe bewerkingen ook geïmplementeerd zodat ze werken met getallen volgens dit principe.

```
ciosMultiplierWrapper2 field_length word_length (a,b,n,nAccent,enable) =
  ciosMultiplier field_length word_length rippleCarryAdder
  fullProdRCAMultiplier NetList (a, b, n, nAccent, enable)
```

Figuur 47: ciosMultiplierWrapper

```
main = do
      putStrLn $ show (testSeqMMTimes 59 4 1 two twelve thirteen one)

testSeqMMTimes times field_length word_length a b n n' =
  simulateN times (ciosMultiplierWrapper2 field_length word_length (a,b,n,n',high))
```

Figuur 48: Main functie testBox_CIOS

De functie testSeqMMTimes voert de CIOS Montgomery Multiplier een ‘times’ aantal cycli uit met enable ‘high’. De waarden ‘times’, ‘field_length’, ‘word_length’ en de input (a,b,n,n’) worden in deze functie ingegeven. Als dit uitgevoerd wordt, wordt het volgende op het einde verkregen:

[[low,low,low,high],high)]

De eerste vier bits is het resultaat SM en is 2^3=8. Dit klopt met het resultaat van de magma calculator. De laatste bit is de ‘DoneAll’ bit en zegt of de berekening gedaan is.

Voor de waarde ‘times’ is gekozen voor 59 aan de hand van volgende formule:
Times = 2*s^2+5*s+7 met s het aantal woorden ofwel s=field_length/word_length. In dit geval is gelijk aan 4 en wordt 59 verkregen. Deze formule is bekomen door:
(1+(2+s)+(2+s))*s+4+3=2*s^2+5*s+7.
Dit is gelijk aan (sRotateT+(sStoreAndShiftPh1Last+sStoreAndShiftPh1)+(sStoreAndShiftPh2Last+sStoreAndShiftPh2))*s+sRED+(sIdle,sInit,sProdDone)
De waarde ‘times’ is zo berekend om het aantal iteraties te bekomen tot de ‘done’ bit high is.

Om te zien of S ook juist bekomen wordt, kan in de main functie ‘two’ en ‘twelve’ vervangen worden door ‘eight’ en ‘one’. Dit is omdat voor de normale waarde te bekomen de Montgomery vermenigvuldiging tussen de Montgomery waarde en 1 wordt berekend. Het resultaat is:

([high,high,high,low],high)]

De eerste vier bits geven het getal 7 weer en dit komt overeen met S. Er is dus te zien dat de CIOS Montgomery vermenigvuldiging werkt.

4.2 Puntverdubbeling

In de Appendix zijn de magma commando’s te vinden. Er is gekozen voor coordinaten (5,4,1,2) en vergelijking van de curve Y^2=X^3+a*X*Z^4+b*Z^6, met (a,b) = (2,11). Er wordt gecheckt of deze coordinaten liggen op de curve E en het punt wordt G genoemd. De resultaten van de

puntverdubbeling in Lava worden geschreven in (x2M,y2M,z2M,az2M). Deze worden dan omgezet van Montgomery, projectieve coördinaten naar normale, affien coördinaten. Het resultaat hiervan wordt punt H genoemd. Daarna wordt het dubbele van G genomen, punt J, en dit punt wordt vergeleken met punt H. In dit voorbeeld zijn de punten ook hetzelfde.

Nu wordt de Lava code bekeken. De test voor de puntverdubbeling staat in het bestand testBox_PtVd_elem.hs.

Voor 'times' wordt volgende formule gebruikt:
times = 16*s^2+40*s+164 = 14*(4+1)+8*(2s^2+5s+7+1)+14*2+2
Dit is 14*(sAdd+sWA)+8*(sMult+sWM)+14*(sRead)+(sIdle,sDoneAll).

Times is 580 in het voorbeeld.

```
verdubbelingWrapper2 field_length word_length (a,b,c,d,n,nAccent,enable) =
  verdubbeling rippleCarryAdder CIOS fullProdRCAMultiplier NetList
  field_length word_length (a, b, c, d, n, nAccent, enable)
```

Figuur 49: verdubbelingWrapper

```
main = do
        putStrLn $ show (testSeqVDTimes 580 4 1 two twelve three six thirteen one)

testSeqVDTimes times field_length word_length x y z az n n' =
  simulateN times (verdubbelingWrapper2 field_length word_length (x,y,z,az,n,n',high))
```

Figuur 50: Main functie testBox_PtVd_elem

De functie testSeqVDTimes voert de verdubbeling een 'times' aantal cycli uit met enable 'high'. Zoals voorheen worden de waarden 'times', 'field_length', 'word_length' meegegeven. De overige input zijn (x,y,z,az,n,n'). Dit zijn de 4 coordinaten, de modulus en inverse modulus. Als dit uitgevoerd wordt, wordt het volgende op het einde verkregen:

([high,high,high,low],[high,low,low,low],[high,high,low,high],[low,high,high,low],low,low,low,low,low,low,high)

Dit geeft (7,1,11,6) in de groepjes van 4 bits. Dit zijn de waarden die eerder in magma werden ingevoerd. De overige bits zijn (oRead, oWrite, oMult, oAdd, done_Mult, done_Add, done_All). Deze geven weer wanneer er welke operatie wordt gedaan. Bijvoorbeeld bij oWrite wordt er een resultaat weggeschreven en bij oMult wordt er een Montgomery vermenigvuldiging uitgevoerd. De laatste bit zegt wanneer alle bewerkingen van de puntverdubbeling gedaan zijn.

4.3 Puntoptelling

De magma commando's zijn weer te vinden in de Appendix. Er is gekozen voor coördinaten (5,4,1,2) en (2,7,1,2). Er wordt gecheckt of deze coördinaten liggen op de curve E en worden G en K genoemd. De resultaten van de puntoptelling in Lava worden geschreven in (x3M,y3M,z3M,az3M). Deze worden dan omgezet van Montgomery, projectieve coördinaten naar normale, affien coördinaten. Het resultaat hiervan wordt punt H genoemd. Daarna wordt de optelling van G en K genomen, punt J, en dit punt wordt vergeleken met punt H. En in dit voorbeeld zijn de punten hetzelfde.

Nu wordt de Lava code bekeken. De test voor de puntoptelling staat in het bestand testBox_PtOpt_elem.hs.

Voor 'times' wordt volgende formule gebruikt:
times = 28*s^2+70*s+177 = 7*(4+1)+14*(2s^2+5s+7+1)+14*2+2
Dit is 7*(sAdd+sWA)+14*(sMult+sWM)+14*(sRead)+(sIdle,sDoneAll).

Times is 905 in het voorbeeld.

```
optellingWrapper2 field_length word_length (a,b,c,d,a2,b2,c2,d2,n,nAccent,enable) =
  optelling rippleCarryAdder CIOS fullProdRCAMultiplier NetList field_length
  word_length (a, b, c, d, a2, b2, c2, d2, n, nAccent, enable)
```

Figuur 51: optellingWrapper

```
main = do
    putStrLn $ show (testSeqOptTimes 905 4 1 two twelve three six six eight
    three six thirteen one)

testSeqOptTimes times field_length word_length x1 y1 z1 az1 x2 y2 z2 az2 n n' =
  simulateN times (optellingWrapper2 field_length word_length
  (x1,y1,z1,az1,x2,y2,z2,az2,n,n',high))
```

Figuur 52: Main functie testBox_PtOpt_elem

De functie testSeqOptTimes voert de optelling een 'times' aantal cycli uit met enable 'high'. Voor de input is het gelijkaardig aan de puntverdubbeling. Het enige verschil is dat de coördinaten van 2 punten worden meegegeven. Het resultaat is:

([high,high,high,low],[low,high,high,low],[low,low,high,low],[high,low,high,low],low,low,low,low,low,low,high)

Dit geeft (7,6,4,5) in de groepjes van 4 bits. Dit zijn de waarden die eerder in magma werden ingevoerd. De overige bits zijn weer hetzelfde als bij de puntverdubbeling.

4.4 Puntvermenigvuldiging

De magma commando's zijn weer gelijkaardig als bij vorige puntbewerkingen en zijn te vinden in de Appendix. De coördinaten zijn nog steeds (5,4,1,2) en het punt noemt G. Voor de vermenigvuldigingsfactor wordt hier 3 genomen. De resultaten van de puntvermenigvuldiging in Lava worden geschreven in (x2M,y2M,z2M,az2M). Deze worden weer omgezet van Montgomery, projectieve coördinaten naar normale, affien coördinaten. Het resultaat hiervan wordt punt H genoemd. Daarna wordt de vermenigvuldiging van G met 3 genomen, punt J, en dit punt wordt vergeleken met punt H. En de punten zijn ook weer in dit voorbeeld hetzelfde.

Nu wordt de Lava code bekeken. De test voor de puntvermenigvuldiging staat in het bestand testBox_PtVm_elem.hs.

Voor 'times' wordt volgende formule gebruikt:
times = (28*n+16*k-44)*s^2+(70*n+40*k-110)*s+(179*n+167*k-343) =
((2+16*s^2+40*s+164)*(k-1)+(2+28*s^2+70*s+177)*(n-1)+(k-1))+3
Dit is ((sRead_Vd+sWrite_Vd+sVd)+(sRead_Opt+sWrite_Opt+sOpt)*(n-

1)+sCheckE)+(sIdle,sInit,sDoneAll) met n het aantal enen in de vermenigvuldigingsfactor, k het aantal significante bits in de factor en s nog steeds het aantal woorden.

Times is 1493 in het voorbeeld.

```
vermenigvuldigingWrapper2 field_length word_length (px,py,pz,paz,k,n,nAccent,enable) =
  vermenigvuldiging rippleCarryAdder CIOS fullProdRCAMultiplier NetList field_length
  word_length (px, py, pz, paz, k, n, nAccent, enable)
```

Figuur 53: vermenigvuldigingWrapper

```
main = do
       putStrLn $ show (testSeqVMTimes 1493 4 1 two twelve three
       six [high,high] thirteen one)

testSeqVMTimes times field_length word_length x y z az k n n' =
  simulateN times (vermenigvuldigingWrapper2 field_length
  word_length (x,y,z,az,k,n,n',high))
```

Figuur 54: Main functie testBox_PtVm_elem

De functie testSeqVMTimes voert de vermenigvuldiging een 'times' aantal cycli uit met enable 'high'. Voor de input is het gelijkaardig aan de puntverdubbeling. Het enige verschil is dat de vermenigvuldigingsfactor wordt meegegeven. Deze factor mag geen overbodige nul bits hebben. Dit kan wel nog aangepast worden vergelijkbaar als in de ModExp code. Maar dit is nog niet gebeurd. Het resultaat is:

([low,high,high,low],[high,low,low,high],[high,high,high,low],[high,low,high,low],low,low,low,low,low,low,low,high)

Dit geeft (6,9,7,5) in de groepjes van 4 bits. Dit zijn de waarden die eerder in magma werden ingevoerd. De overige bits zijn (oRead, oWrite, oMode, oLoad, oRotK, oOpt, done_Opt, done_All). Deze bits geven weer met welke bewerking het programma bezig is.

4.5 Modulaire machtsverheffing

Als voorbeeld voor deze code wordt gekeken naar het getal 'R' en zijn inverse 'Rinv' uit de magma berekeningen van vorige secties. Deze R was 3 en Rinv was 9. De modulus is 13 en de factor p (=modulus-2) nodig voor deze bewerking is dus 11. . Er mag niet vergeten worden dat de input een Montgomery waarde moet zijn en 3*R is in dit geval 9. Er zal nu getest worden in Lava of hetzelfde resultaat zal bekomen worden. Dit resultaat is 9*R=1, de Montgomery waarde van 9. De test gebeurt in bestand testBox_ModExp.hs.

Voor 'times' wordt volgende formule gebruikt:
times = (2*k+2*n-4)*s^2+(5*k+5*n-9)*s+(9*k+9*n-15) = (k+n-2)*(2*s^2+5*s+9)+(s+3) =
((2+2*s^2+5*s+7)*(k-1)+(2+2*s^2+5*s+7)*(n-1)+(k-1))+(z+1)+3
Dit is ((sRead_M1+sWrite_M1+sMult_M1)*(k-1)+(sRead_M2+sWrite_M2+sMult_M2)*(n-1)+sCheckE)+sRMZ+(sIdle,sInit,sDoneAll) met z het aantal overbodige nul bits in de exponent, n het aantal enen in de exponent, k het aantal significante bits in de exponent en s het aantal woorden. Omdat z+k=s wordt dit weggewerkt in de vergelijking.

Times is 312 in het voorbeeld.

```
machtverheffingWrapper2 field_length word_length (m,e,n,nAccent,enable) =
  machtverheffing rippleCarryAdder CIOS fullProdRCAMultiplier NetList
  field_length word_length (m, e, n, nAccent, enable)
```

Figuur 55: machtverheffingWrapper

```
main = do
      putStrLn $ show (testSeqMETimes 312 4 1 nine eleven thirteen one)

testSeqMETimes times field_length word_length m e n n' =
  simulateN times (machtverheffingWrapper2 field_length
  word_length (m,e,n,n',high))
```

Figuur 56: Main functie testBox_ModExp

De functie testSeqMETimes voert de machtverheffing een 'times' aantal cycli uit met enable 'high'. Voor de input worden het basisgetal, de exponent, de modulus en inverse modulus meegegeven. De exponent mag overbodige nul bits hebben in tegenstelling tot de vermenigvuldigingsfactor bij puntvermenigvuldiging. Zoals eerder gezegd kan de puntvermenigvuldiging aangapast worden om op deze manier geïmplementeerd te worden. Het resultaat is:

([high,low,low,low],low,low,low,low,low,low,low,high)

Dit geeft 1 in een groepje van 4 bits. Dit komt overeen met het besproken resultaat. De overige bits zijn (oRead, oWrite, oMode, oLoad, oRotE, oMult, done_Mult, done_All) en geven weer met welke bewerking het programma bezig is.

4.6 NtoM transformatie

Voor deze test worden dezelfde coördinaten als bij vorige voorbeelden gebruikt nl. (5,4,1,2). Er is al geweten uit de magma berekeningen dat (2,12,3,6) de Montgomery waarden zijn. De test gebeurt in bestand testBox_NtoM_elem.hs.

Voor 'times' wordt volgende formule gebruikt:
times = 8*s^2+20*s+38 = 4*(2*s^2+5*s+7)+4*2+2
Dit is 4*(sMont)+4*(sRead,sWrite)+(sIdle,sDoneAll).

Times is 246 in het voorbeeld.

```
toMontgomeryWrapper2 field_length word_length (px,py,pz,paz,rsquare,n,nAccent,enable) =
  toMontgomery rippleCarryAdder CIOS fullProdRCAMultiplier NetList field_length
  word_length (px, py, pz, paz, rsquare, n, nAccent, enable)
```

Figuur 57: toMontgomeryWrapper

```
main = do
        putStrLn $ show (testSeqNtoMTimes 246 4 1 five four one
        two nine thirteen one)

testSeqNtoMTimes times field_length word_length x y z az rsquare n n' =
  simulateN times (toMontgomeryWrapper2 field_length word_length
  (x,y,z,az,rsquare,n,n',high))
```

Figuur 58: Main functie testBox_NtoM_elem

De functie testSeqNtoMTimes voert de NtoM transformatie een 'times' aantal cycli uit met enable 'high'. Voor de input worden de projectieve coördinaten, het getal R in het kwadraat, de modulus en inverse modulus meegegeven. Het getal R is nog steeds 2^(field_length). Het resultaat is:

([low,high,low,low],[low,low,high,high],[high,high,low,low],[low,high,high,low],low,low,low,low,high)

Dit geeft (2,12,3,6) in groepjes van 4 bits en is hetzelfde als de uitkomst van de magma berekeningen.

4.7 MtoN transformatie

Voor deze test worden de Montgomery waarden uit de vorige sectie gebruikt nl. (2,12). Dit zijn alleen de eerste twee waarden omdat de MtoN transformatie wordt gebruikt op affiene coördinaten en de NtoM transformatie op projectieve coördinaten. Er is dus ook geweten uit de vorige sectie dat (5,4) de normale waarden zijn. De test gebeurt in bestand testBox_MtoN_elem.hs.

Voor 'times' wordt volgende formule gebruikt:
times = 4*s^2+10*s+20 = 2*(2*s^2+5*s+7)+2*2+2
Dit is 2*(sMont)+2*(sRead,sWrite)+(sIdle,sDoneAll).

Times is 124 in het voorbeeld.

```
toNormalWrapper2 field_length word_length (px,py,n,nAccent,enable) =
  toNormal rippleCarryAdder CIOS fullProdRCAMultiplier NetList
  field_length word_length (px, py, n, nAccent, enable)
```

Figuur 59: toNormalWrapper

```
main = do
        putStrLn $ show (testSeqMtoNTimes 124 4 1 two
        twelve thirteen one)

testSeqMtoNTimes times field_length word_length x y n n' =
  simulateN times (toNormalWrapper2 field_length word_length
  (x,y,n,n',high))
```

Figuur 60: Main functie testBox_MtoN_elem

De functie testSeqMtoNTimes voert de MtoN transformatie een 'times' aantal cycli uit met enable 'high'. Voor de input worden de affiene coördinaten in Montgomery waarden, de modulus en inverse modulus meegegeven. Het resultaat is:

([high,low,high,low],[low,low,high,low],low,low,low,low,high)

Dit geeft (5,4) in groepjes van 4 bits en is hetzelfde als de verwachte uitkomst.

4.8 AtoP transformatie

De AtoP transformatie heeft geen controle- en datapad. Het is een gewone, simpele functie, waarin bepaalde input waarden naar bepaalde output waarden weggeschreven worden. Als men deze functie uitvoert via ghc is te zien dat deze werkt zoals gewenst.

4.9 PtoA transformatie

Voor deze test wordt er gekeken naar de magma berekeningen van de puntverdubbeling in sectie 3.2. Daar waren de coördinaten (x2M,y2M,z2M,az2M)=(7,1,11,6) verkregen als resultaat van de puntverdubbeling. Dit zijn projectieve Montgomery coördinaten. En de affiene, Montgomery waarden zijn (x2noZM,y2noZM)=(6,8). Er wordt nu gekeken of deze transformatie in de Lava code hetzelfde resultaat bekomt. De test gebeurt in bestand testBox_PtoA_elem.hs.

Voor 'times' wordt volgende formule gebruikt:
times = (2*k+2*n+4)*s^2+(5*k+5*n+11)*s+(9*k+9*n+25) = 4*(2*s^2+5*s+7)+1*(2*k+2*n-4)*s^2+(5*k+5*n-9)*s+(9*k+9*n-15)+5*2+2
Dit is 4*(sMont)+1*(sME)+5*(sRead,sWrite)+(sIdle,sDoneAll).

Times is 560 in het voorbeeld.

```
toAffineWrapper2 field_length word_length (px,py,pz,paz,p,n,nAccent,enable) =
  toAffine rippleCarryAdder CIOS fullProdRCAMultiplier NetList field_length
  word_length (px, py, pz, paz, p, n, nAccent, enable)
```

Figuur 61: toAffineWrapper

```
main = do
      putStrLn $ show (testSeqPtoATimes 560 4 1 seven one eleven
      six eleven thirteen one)

testSeqPtoATimes times field_length word_length x y z az p n n' =
  simulateN times (toAffineWrapper2 field_length word_length
  (x,y,z,az,p,n,n',high))
```

Figuur 62: Main functie testBox_PtoA_elem

De functie testSeqPtoATimes voert de PtoA transformatie een 'times' aantal cycli uit met enable 'high'. Voor de input worden de projectieve coördinaten in Montgomery waarden, de waarde p(= modulus-2), de modulus en inverse modulus meegegeven. Zoals eerder gezegd in sectie 2.7.2 kan de code nog aangepast worden zodat p wordt berekend en niet moet worden meegegeven. Maar dit is nog niet geïmplementeerd. Het resultaat is:

([low,high,high,low],[low,low,low,high],low,low,low,low,low,low,high)

Dit geeft (6,8) in groepjes van 4 bits en is hetzelfde als de verwachte uitkomst.

5 Performantie

In Tabel 3 is het overzicht te zien van de formules die we berekend hebben in de vorige sectie. In de laatste kolom zijn ook de big O notaties te vinden [54].

	aantal iteraties	O(n) notatie
Montgomery-vermenigvuldiging	**2*s^2+5*s+7**	O(s^2)
Puntverdubbeling	**16*s^2+40*s+164**	O(s^2)
Puntoptelling	**28*s^2+70*s+177**	O(s^2)
Puntvermenigvuldiging	**(28*n+16*k-44)*s^2+(70*n+40*k -110)*s+(179*n+167*k-343)**	O(s^2*(n+k))
Modulaire machtsverheffing	**(2*k+2*n-4)*s^2+(5*k+5*n-9)*s +(9*k+9*n-15)**	O(s^2*(n+k))
NtoM transformatie	**8*s^2+20*s+38**	O(s^2)
MtoN transformatie	**4*s^2+10*s+20**	O(s^2)
PtoA transformatie	**(2*k+2*n+4)*s^2+(5*k+5*n+11)*s +(9*k+9*n+25)**	O(s^2*(n+k))

Tabel 3: Overzichtstabel van formules voor aantal iteraties

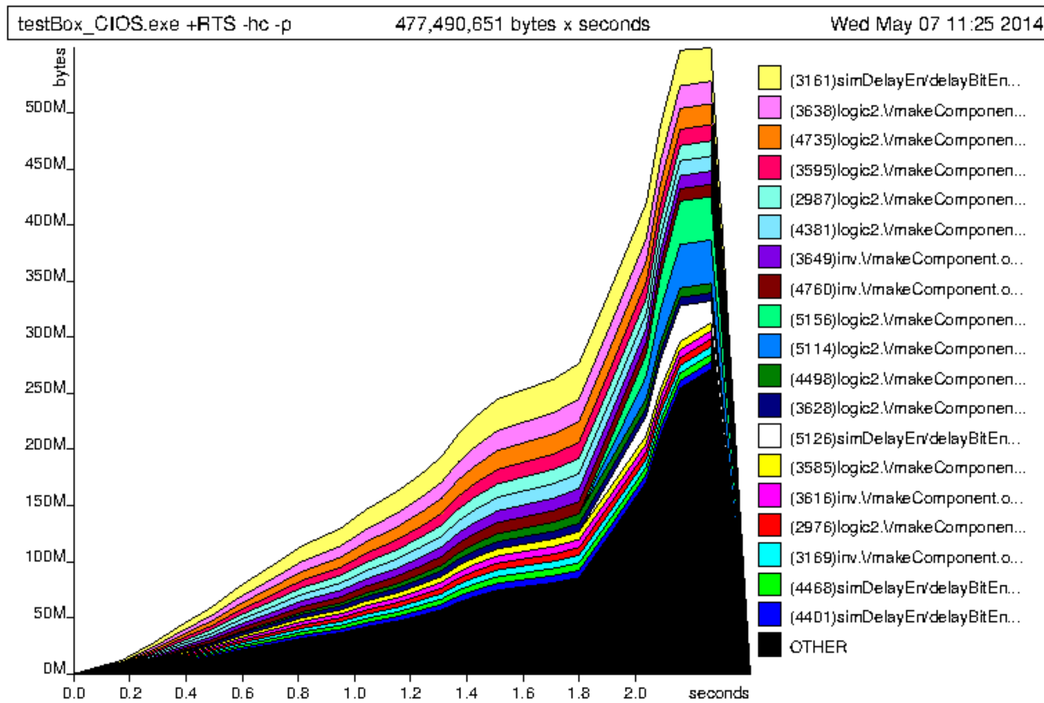
De variabelen in de formules zijn 's', 'n' en 'k'. De eerste variabele 's' is het aantal woorden, 'n' is het aantal enen in de exponent of vermenigvuldigingsfactor bij respectievelijk de modulaire machtsverheffing of puntvermenigvuldiging en 'k' is het aantal significante bits in de exponent of vermenigvuldigingsfactor.

Bij het testen van de CIOS-vermenigvuldiger met 256 bits kwamen er veel 'out-of-memory' errors voor.

Er is getest met getallen van 256 bits waarbij het aantal woorden en woordbreedte een aantal keer is aangepast. Er zijn drie gevallen:
-aantal woorden is 16 en woordbreedte is 16 geeft als aantal iteraties 599.
-aantal woorden is 8 en woordbreedte is 32 geeft als aantal iteraties 2215.
-aantal woorden is 4 en woordbreedte is 64 geeft als aantal iteraties 8519.

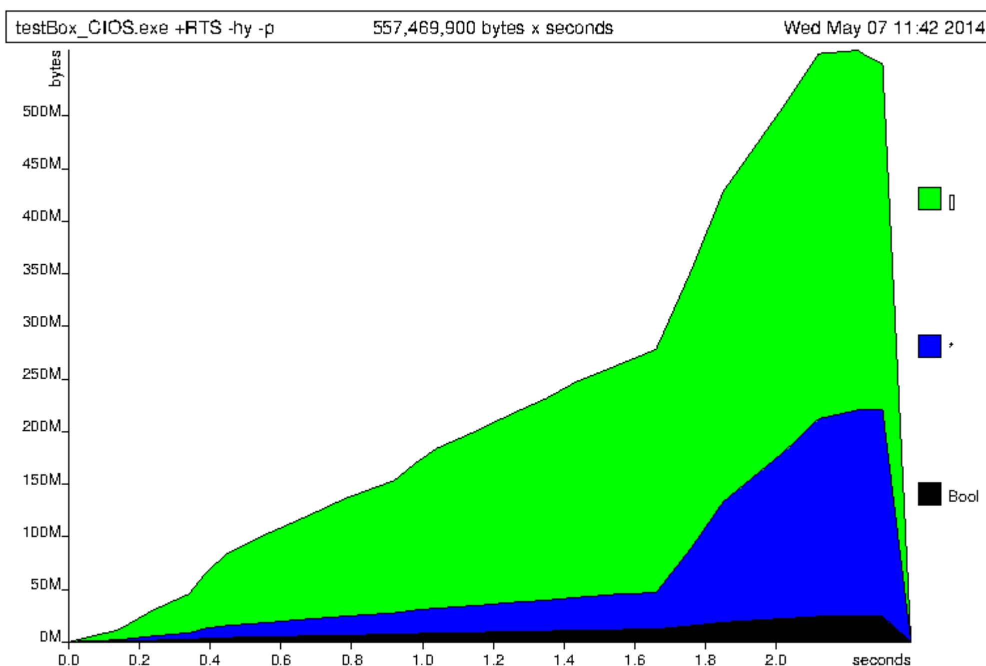
In de eerste twee gevallen worden correcte resultaten bekomen en geen errors, maar het laatste geval verkrijgt een 'out-of-memory' error voor de simulatie een resultaat kan bekomen.

Hierdoor werd er onderzoek gedaan naar de performantie van de functies [55] [56], zodat we kunnen bestuderen waar het meeste geheugen verbruikt wordt. Er werden hiervoor 3 grafieken gemaakt m.b.v. profiling commando's, die terug te vinden zijn via [55]. De processor van de gebruikte laptop, waarop deze commando's zijn uitgevoerd, is Intel® Core™ i5-3210M CPU @ 2.50GHz 2.50 GHz.



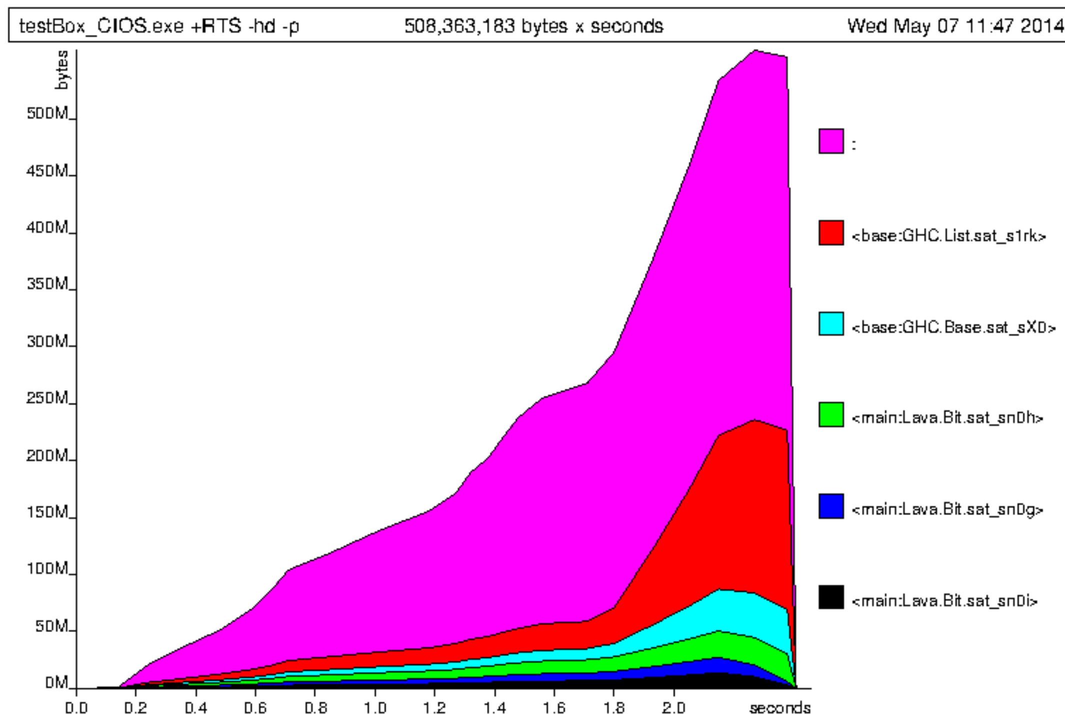
Figuur 63: CIOS Heap Profile by cost-centre stack

Op de eerste grafiek is te zien welke bewerkingen hoeveel geheugen verbruiken en op welk moment in de tijd. Op de horizontale as staat de tijd in seconden en op de verticale as het geheugen in bytes. Er is te zien dat er niet echt een bewerking is die heel veel geheugen verbruikt, op misschien twee na die iets meer verbruiken. Maar er is ook veel geheugen in de sectie OTHER waarvan niet direct geweten is wat dit inhoudt. In een vervolgproject kan hier dieper op ingegaan worden.



Figuur 64: CIOS Heap Profile by type

Op deze tweede grafiek is te zien wat voor type data wordt opgeslagen in het geheugen. Het teken '[]' staat voor lijsten. Het teken '*' staat voor onbekend. Het type Bit kan misschien behoren tot deze groep. Deze twee groepen nemen het meeste geheugen in beslag. En daarnaast is er nog een kleine groep Booleans. De optimisatiefase in een vervolgtraject kan dus best focussen op het verminderen van het aantal lijsten dat wordt gegenereerd.



Figuur 65: CIOS Heap Profile by closure description

De laatste grafiek verdeelt het opgeslagen geheugen naargelang 'closure'. De twee grootste groepen zijn 'violet' en 'rood'. De rode groep heeft te maken met de allocatie van lijsten. De violet groep met teken ':' heeft te maken met het aaneensluiten van lijsten, bv. bij (a:as) is a het eerste element van de lijst en as de overige elementen. Zoals bij Figuur 64 al geconstateerd was, is te zien dat het aantal lijsten voor het hoogste geheugenverbruik zorgt.

In het begin lukte het zelfs niet om voor 8 woordblokken met een woordbreedte van 32 de simulatie in Lava te vervolledigen op onze Windows configuratie. Via een beperkt aantal optimisaties om het geheugenverbruik naar beneden te krijgen, lukte dat uiteindelijk wel, maar voor de Montgomery-vermenigvuldiging met 4 woordblokken met een woordbreedte van 64 bleven we te veel geheugen verbruiken.

Dit wijst er op dat vooral de woordbreedte bepalend is voor het geheugenverbruik. Uiteindelijk is er wegens tijdsgebrek beslist om verder te werken aan de implementatie van de puntbewerkingen. Doelstelling van de tool is immers vooral VHDL te genereren en minder het simuleren in Lava zelf. Bovendien zou ghc op Linux machines meer geheugen kunnen aanspreken dan op Windows. Mogelijk stelt het probleem zich daar minder, en daarom kreeg het dus niet onze prioritaire aandacht.

6 VHDL Conversie

Eén van de originele opdrachten was ook om ervoor te zorgen dat de Lava code kon omgezet worden naar VHDL code. Hiervoor werd een test uitgeprobeerd om de CIOS vermenigvuldiging om te zetten naar VHDL maar die omzetting lukte niet en gaf een error. Deze is in de volgende figuur te zien.

```
*Main> makeCIOSMM
Creating directory 'MM_CIOS_RCA_VHDL_4_2/'
Er bestaat al een submap of bestand -p.
Er is een fout opgetreden tijdens het verwerken van: -p.
Er bestaat al een submap of bestand MM_CIOS_RCA_VHDL_4_2.
Er is een fout opgetreden tijdens het verwerken van: MM_CIOS_RCA_VHDL_4_2.
Writing to 'MM_CIOS_RCA_VHDL_4_2/MM_CIOS_RCA_VHDL_4_2.vhd'... *** Exception: JLi
st.zipWith: incompatible structures
Zero--
(Zero :+: One *** Exception: Can't simulate circuit containing a name ('z17')
*Main>
```

Figuur 66: CIOS VHDL conversie error

De fout met de zipWidth geeft aan dat er niet gebalanceerde structuren zijn die Lava niet kan omzetten in VHDL. Waarschijnlijk zit de oplossing hiervoor diep in de implementatie van de Lava versie zelf. De promotoren hebben aangegeven om hier geen tijd in te steken. Wanneer de tool geconverteerd wordt naar Kansas Lava, zal de VHDL generatie op een heel andere manier gebeuren en lossen we dan de problemen op als/wanneer ze zich voordoen.

Er is geopteerd om verder te gaan met de puntbewerkingen, zodat er meer operaties geïmplementeerd en gesimuleerd kunnen worden.

7 Conclusie

De Montgomery-vermenigvuldiging en puntverdubbeling zijn aangepast omwille van fouten gedetecteerd bij het testen. Puntoptelling en puntvermenigvuldiging zijn correct geïmplementeerd. Testen en simulaties geven het te verwachten resultaat. Getallen met grotere aantal bits (vb. 256 bits bestaande uit 4 woorden met woordbreedte 64 bits) geven nog out-of-memory errors. VHDL conversie werkt nog niet. Dit komt omdat Lava bepaalde ongebalanceerde structuren niet kan omzetten.

De tool biedt voldoende kansen om uitbreidingen toe te voegen, maar kent een steile leercurve. Grootste nadeel van de huidige aanpak is dat het debuggen moeilijk verloopt en dat het zwaar is om inzicht te krijgen hoe de VHDL gegenereerd wordt, waardoor dat probleem niet opgelost kon worden.

De simulatie toont wel aan dat de 2^{e} en 3^{e} laag correct geïmplementeerd zijn. Tegelijk is de kwaliteit van de documentatie verhoogd zodat men in vervolgprojecten gemakkelijker en sneller kan voortbouwen op de resultaten.

8 Uitbreidingsmogelijkheden

Er zijn verschillende uitbreidingsmogelijkheden die kunnen gerealiseerd worden in een vervolgproject:

- De VHDL conversie van de tool operationeel krijgen voor de nieuwe bewerkingen.
- De performantie van de tool verbeteren om het geheugenverbruik te verminderen.
- De verbeterde versie van de Montgomery-vermenigvuldiging implementeren.
- Het testen van de FIOS-vermenigvuldiging en eventueel aanpassen als deze niet werkt.
- Implementeren van de berekening van de factor p(=modulus-2) in de PtoA transformatie. Zien of dit nadelig is voor geheugenverbruik.
- De verwijdering van overbodige zerobits implementeren bij de vermenigvuldigingsfactor in de puntvermenigvuldiging (gelijkaardig als bij modulaire machtsverheffing).
- Nieuwe componenten van ECC toevoegen aan de tool.
- De tool omzetten naar Kansas Lava en documenteren van de verschillen tussen York Lava en Kansas Lava.

9 Appendix

9.1 Magma code

Montgomery-vermenigvuldiging:

```
p := 2^3 + 2^2 + 1;
F := GF(p);
a := F ! 5;
b := F ! 4;
w := 1;
R := F ! 2^(4);
R_square := R^2;
Rinv := R^-1;
p_inv := -InverseMod(p,2^w);
p_inv := p_inv + 2^w;
AM := a*R;
BM := b*R;
SM := AM * BM * Rinv;
S := SM * Rinv;
Stest := a*b;
a;
b;
AM;
BM;
p;
p_inv;
R;
R_square;
Rinv;
SM;
S;
Stest;
```

Figuur 67: Magma berekeningen Montgomery vermenigvuldiging

Puntverdubbeling:

```
p := 2^3 + 2^2 + 1;
F := GF(p);
x := F ! 5;
y := F ! 4;
z := F ! 1;
a := F ! 2;
az := F ! (z^4*a);
b := F ! 11;
x2M := F ! 7;
y2M := F ! 1;
z2M := F ! 11;
az2M := F ! 6;
E := EllipticCurve([a,b]);
G := E![x,y];
w := 1;
R := F ! 2^(4);
R_square := R^2;
Rinv := R^-1;
p_inv := -InverseMod(p,2^w);
p_inv := p_inv + 2^w;
XM := x*R;
YM := y*R;
ZM := z*R;
AZM := az*R;
X2 := x2M*Rinv;
Y2 := y2M*Rinv;
Z2 := z2M*Rinv;
AZ2 := az2M*Rinv;
Z2dub := F ! Z2*Z2;
Z2tri := F ! Z2dub*Z2;
X2noZ := F ! X2/Z2dub;
Y2noZ := F ! Y2/Z2tri;
X2noZM := F ! X2noZ*R;
Y2noZM := F ! Y2noZ*R;
```

```
x;
y;
z;
az;
p;
p_inv;
XM;
YM;
ZM;
AZM;
x2M;
y2M;
z2M;
az2M;
X2;
Y2;
Z2;
AZ2;
X2noZ;
Y2noZ;
X2noZM;
Y2noZM;
G;
H := E ! [X2noZ,Y2noZ];
H;
J := E ! G+G;
J;
```

Figuur 68: Magma berekeningen puntverdubbeling

Puntoptelling:

```
p := 2^3 + 2^2 + 1;
F := GF(p);
x := F ! 5;
y := F ! 4;
z := F ! 1;
a := F ! 2;
az := F ! (z^4*a);
x2 := F ! 2;
y2 := F ! 7;
z2 := F ! 1;
az2 := F ! 2;
b := F ! 11;
x3M := F ! 7;
y3M := F ! 6;
z3M := F ! 4;
az3M := F ! 5;
E := EllipticCurve([a,b]);
G := E![x,y];
K := E![x2,y2];
w := 1;
R := F ! 2^(4);
R_square := R^2;
Rinv := R^-1;
p_inv := -InverseMod(p,2^w);
p_inv := p_inv + 2^w;
XM := x*R;
YM := y*R;
ZM := z*R;
AZM := az*R;
XM2 := x2*R;
YM2 := y2*R;
ZM2 := z2*R;
AZM2 := az2*R;
X3 := x3M*Rinv;
Y3 := y3M*Rinv;
Z3 := z3M*Rinv;
AZ3 := az3M*Rinv;
Z3dub := F ! Z3*Z3;
Z3tri := F ! Z3dub*Z3;
X3noZ := F ! X3/Z3dub;
Y3noZ := F ! Y3/Z3tri;
X3noZM := F ! X3noZ*R;
Y3noZM := F ! Y3noZ*R;
```

Figuur 69: Magma berekeningen puntoptelling, deel 1

```
x;
y;
z;
az;
x2;
y2;
z2;
az2;
p;
p_inv;
XM;
YM;
ZM;
AZM;
XM2;
YM2;
ZM2;
AZM2;
x3M;
y3M;
z3M;
az3M;
X3;
Y3;
Z3;
AZ3;
X3noZ;
Y3noZ;
X3noZM;
Y3noZM;
G;
K;
H := E ! [X3noZ,Y3noZ];
H;
J := E ! G+K;
J;
```

Figuur 70: Magma berkeningen puntoptelling, deel 2

Puntvermenigvuldiging:

```
p := 2^3 + 2^2 + 1;
F := GF(p);
x := F ! 5;
y := F ! 4;
z := F ! 1;
a := F ! 2;
az := F ! (z^4*a);
b := F ! 11;
x2M := F ! 6;
y2M := F ! 9;
z2M := F ! 7;
az2M := F ! 5;
E := EllipticCurve([a,b]);
G := E![x,y];
w := 1;
R := F ! 2^(4);
R_square := R^2;
Rinv := R^-1;
p_inv := -InverseMod(p,2^w);
p_inv := p_inv + 2^w;
XM := x*R;
YM := y*R;
ZM := z*R;
AZM := az*R;
X2 := x2M*Rinv;
Y2 := y2M*Rinv;
Z2 := z2M*Rinv;
AZ2 := az2M*Rinv;
Z2dub := F ! Z2*Z2;
Z2tri := F ! Z2dub*Z2;
X2noZ := F ! X2/Z2dub;
Y2noZ := F ! Y2/Z2tri;
X2noZM := F ! X2noZ*R;
Y2noZM := F ! Y2noZ*R;
x;
y;
z;
az;
p;
p_inv;
XM;
YM;
ZM;
AZM;
x2M;
y2M;
z2M;
az2M;
X2;
Y2;
Z2;
AZ2;
X2noZ;
Y2noZ;
X2noZM;
Y2noZM;
G;
H := E ! [X2noZ,Y2noZ];
H;
J := E ! G*3;
J;
```

Figuur 71: Magma berekeningen puntvermenigvuldiging

Bibliografie

[1] „ES&S," ACRO, [Online]. Available: http://www.acro.be/NL/ess.php?id=102. [Geopend 27 11 2013].

[2] D. Wolfs, K. Aerts en N. Mentens, „A newcomers guide to Crea".

[3] D. Wolfs, K. Aerts en N. Mentens, „Design space exploration for automatically generated cryptographic hardware using functional languages," 29 8 2012. [Online]. Available: https://lirias.kuleuven.be/bitstream/123456789/359120/2/FPL_CREA_published.pdf. [Geopend 6 10 2013].

[4] S. B. Ors, L. Batina en B. Preneel, „Hardware Implementation of Elliptic Curve processor over GF(p)," 2003.

[5] Bithin, „Simple explanation for Elliptic Curve Cryptographic algorithm (ECC)," 22 02 2012. [Online]. Available: http://bithin.wordpress.com/2012/02/22/simple-explanation-for-elliptic-curve-cryptography-ecc/. [Geopend 15 02 2014].

[6] J. Dams, „An introduction to elliptic curve cryptography," 12 10 2012. [Online]. Available: http://www.embedded.com/design/safety-and-security/4396040/An-Introduction-to-Elliptic-Curve-Cryptography. [Geopend 15 02 2014].

[7] K. Baens, KHLim, Diepenbeek, 2013.

[8] KaKaRoTo, „How the ECDSA algorithm works," 31 01 2012. [Online]. Available: http://kakaroto.homelinux.net/2012/01/how-the-ecdsa-algorithm-works/. [Geopend 15 02 2014].

[9] „Haskell (programmeertaal)," Wikipedia, 6 Juni 2013. [Online]. Available: http://nl.wikipedia.org/wiki/Haskell_%28programmeertaal%29. [Geopend 14 november 2013].

[10] S. P. Jones, „Introduction," Haskell, 8 oktober 2013. [Online]. Available: http://www.haskell.org/haskellwiki/Introduction. [Geopend 14 november 2013].

[11] „Functioneel programmeren," 19 april 2013. [Online]. Available: http://nl.wikipedia.org/wiki/Functioneel_programmeren. [Geopend 23 10 2013].

[12] „Comparison of functional programming languages," 07 02 2013. [Online]. Available: http://www.haskell.org/haskellwiki/Comparison_of_functional_programming_languages. [Geopend 15 02 2014].

[13] „Family Tree Of Functional Programming Languages," 24 08 2010. [Online]. Available: http://lambda.jimpryor.net/family_tree_of_functional_programming_languages/. [Geopend 29 05 2014].

[14] „Monad (Functioneel programmeren)," 15 02 2014. [Online]. Available: http://en.wikipedia.org/wiki/Monad_(functional_programming). [Geopend 16 02 2014].

[15] H. Van Thiel, „Ontwerpen met Bluespec," [Online]. Available: http://www.muitovar.com/pub/pdf/bluespec.pdf. [Geopend 14 11 2013].

[16] P. Wadler, „Bluespec," [Online]. Available: http://homepages.inf.ed.ac.uk/wadler/realworld/bluespec.html. [Geopend 14 11 2013].

[17] V. Bonato, „Bluespec Overview," [Online]. Available: http://lcr.icmc.usp.br/cp/en/bluespec.pdf. [Geopend 14 11 2013].

[18] S. Singh, „Bluespec Lectures 3 & 4," [Online]. Available: http://cas.ee.ic.ac.uk/people/ssingh/bluespec_l3l4.pdf. [Geopend 14 11 2013].

[19] D. Chisnall, „What Language I Use for... Hardware Design: BlueSpec," 08 10 2013. [Online]. Available: http://www.informit.com/articles/article.aspx?p=2144810. [Geopend 29 05 2014].

[20] P. Gammie, „Synchronous Digital Circuits as Functional Programs," 2013. [Online]. Available: http://peteg.org/papers/cs_2013_fp_circuits_survey.pdf. [Geopend 14 11 2013].

[21] B. Peemöller, „The Hawk package," 01 06 2010. [Online]. Available: http://hackage.haskell.org/package/Hawk. [Geopend 14 11 2013].

[22] W. Swierstra, K. Claessen, C. Sege, E. Shrive en M. Sheeran, „Functional programming and hardware design: where to now??," 23 09 2009. [Online]. Available: http://www.cs.ox.ac.uk/dcc2010/slides/sheeran.pdf.

[23] J. Matthews, B. Cook en J. Launchbury, „Microprocessor Specification in Hawk," Washington, DC, 1998.

[24] J. Launchbury, „Theorem based Circuit Derivation in Cryptol," 24 10 2011. [Online]. Available: http://www.program-transformation.org/pub/GPCE11/ConferenceProgram/slides-gpce11-launchbury.pdf. [Geopend 14 11 2013].

[25] L. Erkök en J. Matthews, „High assurance programming in Cryptol," 2009. [Online]. Available: http://www.csiir.ornl.gov/csiirw/09/CSIIRW09-Proceedings/Abstracts/Erkok-abstract.pdf. [Geopend 14 11 2013].

[26] J. Hurd en S. Browning, „Cryptol: The language of cryptanalysis," 03 2012. [Online]. Available: http://www.gilith.com/research/talks/sharcs2012.pdf. [Geopend 14 11 2013].

[27] U. Costa, „The Cryptol Epilogue: Swift and Bulletproof VHDL," 23 06 2009. [Online]. Available: http://www.slideshare.net/UlissesCosta/the-cryptol-epilogue-swift-and-bulletproof-vhdl. [Geopend 14 11 203].

[28] L. Pike, „Cryptol: Specification, Implementation and Verification of High-Grade Cryptographic Applications," 2008. [Online]. Available: http://www3.di.uminho.pt/~mbb/cacewebsite/cryptol-slides-pike.pdf. [Geopend 14 11 2013].

[29] D. Moreira en M. Eiras, „Specifying and Implementing ECC primitives in Cryptol," 15 07 2010. [Online]. Available: http://wiki.di.uminho.pt/twiki/pub/Education/MFES0910/ProjectoIntegrado/G1-presentation.pdf. [Geopend 14 11 2013].

[30] L. Erkök, M. Carlsson en A. Wick, „Hardware/Software Co-verification of Cryptographic Algotithms using Cryptol," 18 11 2009. [Online]. Available: http://fmv.jku.at/fmcad09/slides/erkok.pdf. [Geopend 14 11 2013].

[31] U. Costa, „Cryptol the language of cryptography," 01 04 2009. [Online]. Available: http://ulissesaraujo.wordpress.com/2009/04/01/cryptol-the-language-of-cryptography/. [Geopend 29 05 2014].

[32] M. Sheeran en T. Hallgren, 2012. [Online]. Available: http://www.cse.chalmers.se/edu/year/2012/course/TDA956/Slides/Lava112.pdf. [Geopend 29 10 2013].

[33] J. P. Pizani Flor, „Comparing functional Embedded Domain-Specific Languages for hardware description," Utrecht University, 2013.

[34] A. Gill, „Type-Safe Observable Sharing in Haskell," The University of Kansas, 2010.

[35] K. Claessen en D. Sands, „Observable Sharing for functional circuit description," 1999.

[36] „Sharing," 09 11 2007. [Online]. Available: http://www.haskell.org/haskellwiki/Sharing. [Geopend 29 05 2014].

[37] „The york-lava package," Hackage; Cabal, 15 09 2009. [Online]. Available: http://hackage.haskell.org/package/york-lava. [Geopend 7 11 2013].

[38] M. Naylor, C. Runciman en R. Jason, „The Reduceron," 2010. [Online]. Available: http://www.cs.york.ac.uk/fp/reduceron/. [Geopend 14 11 2013].

[39] M. Naylor en C. Runciman, „The Reduceron reconfigured and re-evaluated," The University of York, 2010.

[40] „The kansas-lava package," Hackage; Cabal, 07 11 2011. [Online]. Available: http://hackage.haskell.org/package/kansas-lava. [Geopend 7 11 2013].

[41] A. Gill en B. Neuenschwander, „Handshaking in Kansas Lava using Patch Logic," The University of Kansas, 2012.

[42] J. Maistros, B. Siegel en P. E. III, „Elliptic Curve Cryptography in HDL," 05 04 2007. [Online]. Available: http://www.bsiegel.net/FinalTechnicalDesignReport.pdf. [Geopend 21 11 2013].

[43] Ç. K. Koc en T. Acar, „Analyzing and Comparing Montgomery Multiplication Algorithms," IEEE, 1996.

[44] J. Launchbury, „Compositional Verification of Ellipctic Curve Cryptography," 2012. [Online]. Available: http://www.acsac.org/2012/workshops/law/pdf/Launchbury.pdf. [Geopend 14 11 2013].

[45] N. Mentens, „Secure and Efficient Coprcocessor Design for Cryptographic Applications on FPGAs," ESAT, Katholieke Universiteit Leuven, 2007.

[46] P. Montgomery, „Modular Multiplication Without Trial Division," 1985.

[47] C. D. Walter, „An Overview of Montgomery Multiplication Technique: How to make it Smaller and Faster," 1999.

[48] C. D. Walter, „Montgomery Exponentiation Needs No Final Subtractions," 1999.

[49] Z. Cheng en M. Nistazakis, „Implementing Pairing-Based Cryptosystems," 2005.

[50] H. Cohen, A. Miyaji en T. Ono, „Efficient Elliptic Curve Exponentiation Using Mixed Coordinates," 1998.

[51] T. Marley, „Fermat's theorem and inverse modulo m," 10 11 2008. [Online]. Available: http://www.math.unl.edu/~tmarley1/math189/notes/Nov4-6notes.pdf. [Geopend 29 05 2014].

[52] M. Hausner, „Fermat's Little Theorem," 2007. [Online]. Available: http://www.math.nyu.edu/faculty/hausner/fermat.pdf. [Geopend 29 05 2014].

[53] „Big and Little Endian," [Online]. Available: http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/endian.html. [Geopend 29 05 2014].

[54] „Big O notation," [Online]. Available: http://www.cs.waikato.ac.nz/Teaching/COMP317B/Week_1/Big_o_notation.html. [Geopend 29 05 2014].

[55] B. O'Sullivan, D. Stewart en J. Goerzen, „Haskell: Profiling and Optimization," [Online]. Available: http://book.realworldhaskell.org/read/profiling-and-optimization.html. [Geopend 05 05 2014].

[56] „Haskell: Profiling Memory Usage," [Online]. Available: http://www.haskell.org/ghc/docs/7.0.4/html/users_guide/prof-heap.html. [Geopend 14 05 2014].

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Een bibliotheek van cryptografische operatie omzetten met Lava

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt
behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -,
vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten
verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Hannosset, Nicky

Datum: **6/06/2014**