

IMPLEMENTATIE VAN VERMOGENAANVALLEN EN TEGENMAATREGELLEN OP FPGA

Rapport over de masterproef van

Joren MOELANS en Steven SWINNEN

kandidaten voor de graad van Academische Master
Industriële Wetenschappen, Elektronica-ICT

Promotoren:
dr. L. Batina
ing. J. Vliegen
dr. ir. N. Mentens

Academiejaar 2009 - 2010
Referentie: E10/ELO/10

Woord vooraf

Deze scriptie is voorgedragen door ons, Joren Moelans en Steven Swinnen, om aan te tonen dat we de nodige vakkennis en vaardigheden bezitten die een Academische Master Industriële Wetenschappen, Elektronica-ICT moet hebben.

Al van bij de voostelling van de masterproefonderwerpen in mei 2009, boeide dit onderwerp ons het meest. We bedanken in de eerste plaats dan ook dr. Lejla Batina en dr. ir. Nele Mentens omdat zij ons de kans gaven om dit onderwerp aan te vatten.

De uitvoering van deze masterproef deden we in het Technologiecentrum te Diepenbeek. De medewerkers van ACRO, in het bijzonder ing. Jo Vliegen, zorgde voor het materiaal en de dagelijkse begeleiding. We zijn Jo zeer dankbaar voor de hints en tips die hij ons gaf. Deze zijn enorm waardevol geweest voor de voltooiing van onze masterproef.

Naast de rechtstreekse begeleiders willen we ook nog enkele andere mensen bedanken. Ten eerste bedanken we Dr. Jeroen Lievens. Hij was degene die communicatie rond de masterproef verzorgde en het spellingsadvies voor deze scriptie gaf. Ten tweede bedanken we de medestudenten die ook hun masterproef deden in het technologiecentrum. Hen bedanken we voor de fijne tijden en we wensen hen veel succes toe in hun carrière. Ten slotte willen we iedereen bedanken die iets met dit eindwerk te maken had.

J. Moelans en S. Swinnen

Kandidaten voor Academische Master Industriële wetenschappen, Elektronica-ICT
Katholieke Hogeschool Limburg

Inhoudsopgave

Lijst van tabellen	7
Lijst van figuren	8
Afkortingen	13
Verklarende woordenlijst	15
Abstract	17
Summary	18
Inleiding	19
1 Cryptologie	22
1.1 Geschiedenis van de cryptologie	22
1.1.1 Ontstaan van de cryptologie	23
1.1.2 Cryptologie in oorlogstijd	23
1.2 Hedendaagse cryptologie	25
1.2.1 Aspecten	25
1.2.2 Ontwikkeling	28
1.2.3 Indeling	29
2 Cryptografie	31
2.1 Symmetrische cryptografie	32
2.1.1 Blokcijfers	32
2.1.2 De Data Encryptie Standaard of DES	34
2.1.3 Stroomcijfers	36
2.2 Asymmetrische cryptografie	37

2.2.1	Het RSA encryptie schema	38
2.2.2	ECC encryptie	40
2.2.3	Digitale handtekening	44
2.3	Symmetrische- vs. Asymmetrische cryptografie	45
2.3.1	Voordelen van symmetrische cryptografie	45
2.3.2	Nadelen van symmetrische cryptografie	46
2.3.3	Voordelen van asymmetrische cryptografie	46
2.3.4	Nadelen van asymmetrische cryptografie	46
2.4	Hashfuncties	47
2.4.1	Eigenschappen	47
2.4.2	Hashfuncties in de praktijk	48
3	Cryptanalyse	49
3.1	Inleiding	49
3.2	Aanvalmodellen	50
3.3	Aanvalsmethoden	51
3.3.1	Brute kracht	51
3.3.2	Nevenkanalen (side channels)	51
3.4	Vermogenanalyse	51
3.4.1	Soorten	52
3.4.2	Praktisch voorbeeld	54
4	Vermogenanalyse meetopstelling - Ontwerpfase	56
4.1	Inleiding	56
4.2	SASEBO - Het FPGA-bord	57
4.2.1	Inleiding	57
4.2.2	Specificaties SASEBO-G	58
4.2.3	USB-interface	59
4.3	LabVIEW - De programmeertaal	61
4.3.1	Wat is LabVIEW?	61
4.3.2	Opbouw van een VI	62
4.3.3	Parallele poort in LabVIEW	63
4.4	MSO6052A - De oscilloscoop	65
4.4.1	De Agilent MSO6052A	65
4.4.2	GPIB - Interface tussen meettoestellen	65
4.4.3	SCPI - Commando's voor meettoestellen	69

4.4.4	Programmeerstructuur	70
4.4.5	Nuttige commando's	72
4.4.6	Interactie tussen LabVIEW en MSO6052A	73
4.4.7	De praktijk - eerste tests	76
4.5	Controle interface tussen pc en SASEBO	78
4.5.1	Simpele digitale I/O kaart	78
4.5.2	USB	81
5	Vermogenanalyse meetopstelling - Eindresultaat	86
5.1	pc-software	86
5.1.1	Gebruiksaanwijzingen	87
5.1.2	Gebruikte programmeerprincipes	92
5.2	VHDL-wrapper	94
5.2.1	RD-usb.vhd	95
5.2.2	WR_usb.vhd	97
5.2.3	Wrapper.vhd	99
5.2.4	Opmerkingen	102
5.3	Struikelblokken	103
5.3.1	Weergave van de kleine datavensters binnen een algoritme	103
5.3.2	Triggerbron van de Oscilloscoop	107
6	Testanalyses	111
6.1	Vermogenanalyse van een schuifregister	111
6.1.1	Inleiding	111
6.1.2	Opbouw van het algoritme	112
6.1.3	Realisatie in VHDL	112
6.1.4	Resultaten	113
6.1.5	Struikelblokken	118
6.2	SPA-vermogenanalyse van een simpel algoritme	120
6.2.1	Inleiding	120
6.2.2	Opbouw van het algoritme	120
6.2.3	Resultaten	122
6.2.4	Verhinderen van een SPA-vermogenaanval	123
7	SPA-aanval op een ECC-algoritme gebaseerd op Edwards-krommen	125
7.1	Inleiding	125

7.2	Werking van het algoritme	125
7.3	Communicatielogica en algoritme in controle-FPGA	127
7.3.1	Algemene opbouw	127
7.3.2	FSM-flowchart	128
7.3.3	Resultaat	129
7.4	Opsplitsing in controle- en cryptografische-FPGA	130
7.4.1	VHDL-wrapper in controle-FPGA	131
7.4.2	VHDL-wrapper in cryptografische-FPGA	133
7.4.3	Resultaat	136
7.5	Maatregelen	140
7.5.1	Resultaat	143
8	CPA-aanval op een ECC-algoritme gebaseerd op Edwards-krommen	146
8.1	Inleiding	146
8.2	Modulaire machtsverheffing VS. Modulaire optelling/verdubbeling	147
8.3	Matching van het vermogenverbruik van de twee metingen	149
8.4	Resultaat	149
8.4.1	Onthulling van de eerste sleutelbit	150
8.4.2	Onthullen van de andere sleutelbits	154
9	Besluit	157
	Bibliografie	159
	Appendices	
	Inhoud van de cd met bijlages	162

Lijst van tabellen

7.1	Aantal klokcycli van ECC-algoritme N. Cornelissen & C. Peeters	127
7.2	Programmaflow als de sleutelbit één is	142
7.3	Programmaflow als de sleutelbit nul is	142
7.4	Programmaflow als de sleutelbit één is na verbetering	143
7.5	Programmaflow als de sleutelbit nul is na verbetering	143

Lijst van figuren

1.1	Een 3-rotor Wehrmacht Enigma in het Imperial War Museum, London. . .	24
1.2	Voorstelling van user authentication door M.Vandenwauver	25
1.3	Voorstelling van data integrity door M.Vandenwauver	26
1.4	Voorstelling van data origin authentication door M.Vandenwauver	26
1.5	Voorstelling van non-repudiation door M.Vandenwauver	27
1.6	Voorstelling van data confidentiality door M.Vandenwauver	28
1.7	Indeling van de cryptologie	29
2.1	Essentie van sleutel gebaseerde cryptografie	31
2.2	Hashfunctiecryptografie	31
2.3	Principeschema van symmetrische cryptografie uit 'Overview of cryptografie' door A. Menezes	32
2.4	Principeschema van een blokcijfer	33
2.5	Encryptie bij CBC-mode blokcijfers uit 'Blockcipher modes of operation' ,Wikipedia	33
2.6	Decryptie bij CBC-mode blokcijfers uit 'Blockcipher modes of operation', Wikipedia	33
2.7	[20] Blokschema van DES uit 'Data Encryption Standard'	35
2.8	Principeschema van een stroomcijfer	36
2.9	Principeschema van asymmetrische cryptografie uit 'Overview of cryptogra- fie' door A. Menezes	38
2.10	Berekenen van de RSA sleutels	40
2.11	Plot van twee elliptische krommen uit de masterthesis, N. Cornelissen en C. Peeters [25]	41
2.12	Puntopstelling bij elliptische krommen uit de masterthesis, N. Cornelissen en C. Peeters [25]	42

2.13	Puntverdubbeling bij elliptische krommen uit de masterthesis, N. Cornelissen en C. Peeters [25]	42
2.14	Zender met berekening van digitale handtekening	44
2.15	Ontvanger met verificatie van digitale handtekening	45
2.16	schematische voorstelling van One-way	47
2.17	schematische voorstelling van target collision resistant	48
2.18	schematische voorstelling van Random collision resistant	48
3.1	Principeschema van een vermogenanalyse	52
3.2	Voorbeeld van simple power-analysis attack	53
3.3	Voorbeeld van differentiële power-analysis attack	53
3.4	Algoritme voor het bepalen van de publieke sleutel bij elliptische krommen cryptografie uit masterthesis, N. Cornelissen en C. Peeters [25]	54
3.5	Vermogenvorm tijdens een algoritme dat de publieke sleutel berekend bij elliptische kromme cryptografie	55
4.1	Meetopstelling	57
4.2	Afbeelding van het SASEBO-G evaluatie bord	58
4.3	Logo FTDI-chip	60
4.4	USB-functionaliteit op SASEBO-G	60
4.5	Het LabVIEW icoon	61
4.6	Controls palette	62
4.7	Controls palette	63
4.8	Icoon - connectorblok	63
4.9	Voorbeeld parallelle input, geschreven door National Instruments	64
4.10	Agilent MSO6052A oscilloscoop	65
4.11	Mogelijke GPIB-busconfiguraties. Links, de lineaire configuratie. Rechts, de sterconfiguratie	67
4.12	NI PCIe-GPIB+	69
4.13	SCPI commando structuur uit [15]	70
4.14	Basisstructuur van het oscilloscoop programma van [11]	71
4.15	Functie pallet voor Agilent MSO60xxA-type oscilloscoop	74
4.16	Configuratie subVI's voor de AG60XXA serie	75
4.17	Utility subVI's voor de AG60XXA serie	76
4.18	Data subVI's voor de AG60XXA serie	76
4.19	Testprogramma voor connectie te maken	77

4.20	Testprogramma om data uit te lezen	78
4.21	Beschrijving van de parallele poort	79
4.22	De gerealiseerde I/O interface	80
4.23	Input gedeelte van de I/O kaart	81
4.24	Output gedeelte van de I/O kaart	81
4.25	Communicatieprotocol tussen SASEBO en pc	83
4.26	Pin-layout van de FT245R uit [10]	83
4.27	Readcyclus van de FT245R uit [10]	84
4.28	Writecyclus van de FT245R uit [10]	85
5.1	Configuratie-instellingen van het meetprogramma	87
5.2	Gebruikersinstellingen van het meetprogramma	88
5.3	Settingsmenu van het hoofdmenu	89
5.4	Meetmenu van het hoofdmenu	90
5.5	Verwittiging: Zijn de scoop-settings correct?	91
5.6	Verwittiging: Scoop niet getriggerd.	91
5.7	Verwittiging: Vervang bestaande bestanden?	91
5.8	Verwittiging: Te weinig data-inputs in de tekst file.	92
5.9	Voorbeeld log bestand	92
5.10	Blockdiagram van main.vi	93
5.11	Voorbeeld van een Queue-lijst	94
5.12	Koppeling tussen de modules van de VHDL-wrapper	95
5.13	Port-map van RD_usb module	96
5.14	Flowchart van RD_usb FSM	97
5.15	Port-map van WR_usb module	98
5.16	Flowchart van WR_usb FSM	99
5.17	Port-map van de wrapper module	100
5.18	Flowchart de wrapper FSM	101
5.19	Testopstelling voor gedetailleerde opname van stukken uit een algoritme . .	105
5.20	Plot van de trigger- en datapuls in functie van de systeemklok	105
5.21	Opname van de datapuls na een delayed-trigger door de triggerpuls	106
5.22	LabVIEW testprogramma voor automatische instelling van de delayed-trigger	106
5.23	Eerste opname van het gewenste tijdsvenster met triggering door startpuls	108
5.24	Tweede opname van het gewenste tijdsvenster met triggering door startpuls	108
5.25	Blokschematische voorstelling van een state-machine die een triggerpuls ge- nereert juist voor het starten van het algoritme	109

5.26	Eerste opname van het gewenste tijdsvenster met triggering door FPGA	109
5.27	Tweede opname van het gewenste tijdsvenster met triggering door FPGA	110
6.1	Basisprincipe van het te maken schuifregister	111
6.2	Algoritme met schuifregister	112
6.3	Blokschema van de FSM van het schuifregister algoritme	113
6.4	Vermogen verloop over een één ohm weerstand bij een 256-bits schuifregister	114
6.5	Vermogen verloop over een één ohm weerstand bij een 4096-bits schuifregister	115
6.6	Vermogen verloop over een één ohm weerstand bij een 8192-bits schuifregister	115
6.7	Vermogenanalyse met behulp van de meetopstelling van het 256-bits schuif- register	116
6.8	Vermogenanalyse met behulp van de meetopstelling van het 4096-bits schuif- register	117
6.9	Vermogenanalyse met behulp van de meetopstelling van het 8192-bits schuif- register	117
6.10	Vermogen verloop over een één ohm weerstand bij een 8192-bits schuifregister	118
6.11	Vermogen meetinrichting van SASEBO-G	119
6.12	Opbouw van de FSM rondom het simpel SPA-algoritme	121
6.13	Resultaat van de SPA-vermogenanalyse met 8-bit sleutel	122
6.14	Resultaat van de SPA-vermogenanalyse met 16-bit sleutel	123
6.15	Resultaat van de SPA-vermogenanalyse met 8-bit sleutel	124
7.1	ECC-versleutelingsalgoritme	126
7.2	Algemeen blokschema van de VHDL code in de controle FPGA	128
7.3	Flowchart van de VHDL code in de controle FPGA	129
7.4	Vermogenverbruik van de controle FPGA tijdens de onregelmatigheid net na de vierde cyclus	130
7.5	Algemeen blokschema van het programma in de controle FPGA voor het ECC-algoritme	132
7.6	Flowchart van de FSM in de controle FPGA voor het ECC-algoritme	133
7.7	Algemeen blokschema van het programma in de cryptografische FPGA voor het ECC-algoritme	134
7.8	Flowchart van de FSM in de cryptografische FPGA voor het ECC-algoritme	135
7.9	Vermogenverbruik van de cryptografische FPGA tijdens de vierde cyclus	136
7.10	Vermogenverbruik van de cryptografische FPGA tijdens de achtste cyclus	137

7.11	Vermogenverbruik van de cryptografische FPGA tijdens de vermogendrop na de vierde cyclus	137
7.12	Vermogenverbruik van de cryptografische FPGA tijdens de vermogendrop na de achtste cyclus	138
7.13	Vermogenverbruik van de cryptografische FPGA tijdens de onregelmatigheid net na de vierde cyclus	139
7.14	Vermogenverbruik van de cryptografische FPGA tijdens de onregelmatigheid net na de achtste cyclus	139
7.15	Algemeen beeld van de simulatie van de eerste acht cycli van het ECC-algoritme	141
7.16	Uitvoeringstijd van de nodige instructies na de vierde cyclus	141
7.17	Uitvoeringstijd van de nodige instructies na de achtste cyclus	142
7.18	Uitvoeringstijd van de nodige instructies vóór de vijfde cyclus na verbetering	144
7.19	Uitvoeringstijd van de nodige instructies vóór de negende cyclus na verbetering	144
7.20	Vermogenverbruik van de cryptografische FPGA na de vierde cyclus	145
7.21	Vermogenverbruik van de cryptografische FPGA na de achtste cyclus	145
8.1	Principe van de CPA-aanval uit [31]: dezelfde bewerkingen	147
8.2	Principe van de CPA-aanval uit [31]: verschillende bewerkingen	147
8.3	Principe van de CPA-aanval op het ECC-algoritme van N. Cornelissen & C. Peeters [25]	148
8.4	Stappen voor waveform-matching uit [31]	149
8.5	Vermogenverloop van de puntverdubbeling tijdens de initialisatie	151
8.6	Vermogenverloop van de puntverdubbeling tijdens de evaluatie van de eerste sleutelbit als die nul is.	151
8.7	Vermogenverloop van de puntverdubbeling tijdens de evaluatie van de eerste sleutelbit als die één is.	152
8.8	Overgangsverschijnsel bij het starten van het algoritme	153
8.9	Vermogenverloop van de initialisatie als Z gelijk is aan één	154
8.10	Vermogenverloop van de initialisatie als Z verschillend is van één	154
8.11	Vergelijking tussen cyclus 1 (dubbel ingangspunt) en cyclus 2(enkel ingangspunt)	155
8.12	Vergelijking tussen cyclus 2 (dubbel ingangspunt) en cyclus 3(enkel ingangspunt)	155
1	Inhoudoverzicht van de cd	162

Afkortingen

AES	Advanced Encryption Standard
ASIC	Application-Specific Integrated Circuit
CBC	Cipher Block Chaining
CFB	Cipher FeedBack
DES	Data Encryption Standaard
DPA	Differential Power-Analysis Attack
DSA	Digital Signature Algorithm
ECB	Electronic CodeBook
ECC	Elliptic Curve Cryptography
EMC	ElektroMagnetische Compatibiliteit
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPIB	General Purpose Interface Bus
HP-IB	Hewlett-Packard Interface Bus
I/O	Input/Output
IBM	International Business Machines Corporation
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronic Engineers

IT	Information Technologie
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
NBS	National Bureau of Standards
NI	National Instruments .inc
NIST	National Institute of Standards and Technologie
NSA	National Security Agency
OFB	Output FeedBack
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
RCIS	Research Center for Information Security
RSA	Rivest/Shamir/Adleman
SASEBO	Side-channel Attack Standard Evaluation Board
SCPI	Standard Commands for Programmable Instruments
SMD	Surface Mounted Device
SPA	Simple Power-analysis Attack
TCP/IP	Transmission Control Protocol/Internet Protocol
TDEA	Triple Data Encryption Algorithm
USB	Universal Serial Bus
v.Chr.	voor Christus
VHDL	VHSIC Hardware Discription Language
VHSIC	Very High Speed Integrated Circuits
VI	Virtual Instrument

Verklarende woordenlijst

Aanvallen: een theoretische benadering of praktische implementatie van een cryptografisch systeem onderzoeken om de klaartekst te achterhalen.

Cijferalfabet: een groepering van symbolen, letters of cijfers waaruit de cijfertekst is opgebouwd.

Cijfertekst: gecijferde informatie die slechts verstaanbaar is na ontcijfering.

Cryptanalyse: een wetenschap die gecijferde boodschappen probeert te ontcijferen zonder de sleutel te kennen.

Decoderen of decrypteren: het ontcijferen van de gecijferde boodschap.

Encoderen of encrypteren: het gecijferen van de boodschap volgens een bepaald algoritme.

Hashfunctie: een functie uit de informatica, die een groot aantal van waarden omzet in een waarde met vaste lengte.

Klaaralfabet: een groepering van symbolen, letters of cijfers waaruit de klaartekst is opgebouwd.

Klaartekst: de informatie die nog niet gecijferd is of reeds ontcijferd is. Deze tekst is dus begrijpbaar voor iedereen.

Sleutel: een klaartekst wordt met behulp van van een sleutel omgezet naar een cijfertekst. Aan ontvangerzijde moet een (al dan niet andere) sleutel gekend zijn om de cijfertekst terug om te zetten naar klaartekst.

Substitutievercijfering: elk symbool uit klaartekst (die uit het klaaralfabet bestaat) wordt vervangen door een ander symbool uit het cijferalfabet.

Veilig kanaal: een communicatie kanaal (medium) dat fysisch beveiligd is. Denk maar aan de briefomslag bij de post, die het briefgeheim bewaard.

Onveilig kanaal: een communicatie kanaal (medium) dat toegankelijk is voor iedereen. Denk maar aan een boodschap in de krant.

Abstract

COSIC is een onderzoeksgroep van het departement Elektrotechniek van de KULeuven. Een deel van hun onderzoek richt zich op het ontwerp, de evaluatie en de implementatie van cryptografische algoritmes op elektronische platformen. Sinds de introductie van vermogenaanvallen op deze platformen zijn een groot aantal bestaande algoritmes onveilig geworden. Deze masterproef heeft als doel een implementatie van een algoritme gebaseerd op Elliptic Curve Cryptography te evalueren op veiligheid tegen vermogenaanvallen. Hiervoor is tijdens deze masterproef een geautomatiseerde vermogenmeetopstelling ontwikkeld.

De software van de vermogenmeetopstelling ontwikkelden we met LabVIEW. Deze software stuurt een Agilent oscilloscoop en een gespecialiseerd FPGA-bord aan. Daarna testten we de vermogenmeetopstelling met behulp van testalgoritmes geïmplementeerd op het FPGA-bord. De verschillende vermogenaanvallen op de implementatie van het elliptische kromme algoritme deden we met behulp van onze meetopstelling en statistische berekeningen.

We deden een simpele vermogenaanval (SPA) op een bestaande elliptische kromme implementatie, hierdoor kwam een grootte implementatiefout aan het licht. Door extra code aan de implementatie toe te voegen, elimineerde we de fout en werd de implementatie bijgevolg veiliger. Daarnaast deden we een vergelijkende vermogenaanval (CPA) op dezelfde implementatie. Deze aanval onthulde een gedeelte van de geheime informatie.

Summary

COSIC is a research group of the electrical engineering department of the KULeuven. A part of their research is focused on the design, evaluation and implementation of cryptographic algorithms on electronic platforms. Since the introduction of power attacks on these platforms, many existing algorithms became unsafe. This master thesis aims at an implementation of an algorithm, based on Elliptic Curve Cryptography, to evaluate its security against power attacks. For this purpose an automated power measurement system is developed, during this master thesis.

The software of the power measurement system is developed with LabVIEW. This software controls an Agilent oscilloscope and a specialized FPGA board. Afterwards we tested the power measurement system, using test algorithms implemented on the FPGA board. We did various power attacks on the implementation of the elliptic curve algorithm with our power measurement system and statistical calculations.

While performing a simple power attack (SPA) on the existing elliptic curve implementation. This brought up a major implementation error. By adapting the implementation, we eliminated the error and the implementation became much safer. In addition, we did a comparative power analysis attack (CPA) on the same implementation. This attack revealed a portion of the secret information.

Inleiding

Situering

Dit eindwerk loopt in opdracht van COSIC. COSIC staat voor Computer Security and Industrial Cryptography en is opgericht in 1978. COSIC is een onderdeel van het departement Elektrotechniek van de KULeuven. Door de groeiende digitalisering van vertrouwelijke informatie wordt cryptografie steeds belangrijker. De onderzoeksgroep COSIC probeert de beveiliging van deze digitale informatie voortdurend te verbeteren om zo fraude tegen te gaan.

Het basisonderzoek bij COSIC dat relevant is voor deze masterproef, is opgedeeld in twee onderdelen. Eerst zoeken wiskundigen naar methodes en algoritmes om de veiligheid en echtheid van elektronische data te garanderen. Daarna gaan technici de uitgedachte algoritmes implementeren en evalueren in een elektronisch systeem. Als uit de evaluatie blijkt dat het algoritme niet volledig veilig is, wordt het verbeterd.

Dit eindwerk situeert zich binnen het tweede onderdeel van het basisonderzoek van COSIC. We gaan bepaalde cryptografische algoritmes evalueren en indien nodig verbeteren. Al deze algoritmes zijn geïmplementeerd op FPGA.

Het basisonderzoek van COSIC wordt toegepast op verschillende domeinen, bijvoorbeeld de elektronische identiteitskaart, elektronische verkiezingen, beveiliging van e-documents,... Een samenwerkingsverband tussen COSIC en verschillende overheden of bedrijven zorgt ervoor dat deze toepassingen tot stand komen.

Globale probleemstelling

Recent introduceerden cryptanalisten aanvallen die nevenkanaalinformatie zoals, vermogen, temperatuur en straling van een chip gebruiken om de sleutel van het cryptografisch systeem te achterhalen. Een veelgebruikt nevenkanaal is het vermogenverbruik van een chip.

Het is mogelijk om met behulp van een aantal vermogenmetingen de sleutel van het cryptografisch systeem te vinden. Om een vermogenaanval uit te voeren moet het cryptografisch algoritme en een dataset gekend zijn. Een vermogenmeetopstelling geeft de data aan het algoritme door en start daarna een of meerdere metingen. Wanneer de metingen gedaan zijn, analyseert de cryptanalist de vermogenpatronen en leidt daaruit de sleutel af.

Veel hedendaagse algoritmes zijn niet bestand tegen vermogenaanvallen en hierdoor komt hun veiligheid in gedrang. Onveilige algoritmes zijn een probleem voor toepassingen die gevoelige informatie beschermen, zoals banksystemen, digitale identiteitskaarten, pay-TV, enz. Buitenstaanders kunnen de toepassing ontleden en informatie bekijken.

Indien cryptografen niet snel een oplossing vinden om de bestaande algoritmes terug veilig te maken, kan een vermogenaanval een catastrofe betekenen voor bedrijven, de overheid en individuen. Stel bijvoorbeeld dat bepaalde bedrijfsgeheimen toegankelijk worden voor de concurrentie, dan kan dat zelfs leiden tot een faillissement van het getroffen bedrijf.

Doelstellingen

Het hoofddoel van dit eindwerk bestaat eruit om een bestaand algoritme te beveiligen tegen een aantal typische vermogenaanvallen. Bekende vermogenaanvallen zijn SPA, DPA, CBA, enz.

Voordat we een algoritme veiliger kunnen maken tegen vermogenaanvallen, moeten we eerst de initiële veiligheid onderzoeken. Hiervoor is een geautomatiseerde gebruiksvriendelijke meetopstelling nodig die zeer stabiel is en op een snelle nauwkeurige manier metingen uitvoert. Deze meetopstelling is nog niet beschikbaar binnen de KHLim.

Het algoritme dat we bestuderen, wordt gebruikt binnen de elliptic curve cryptography of ECC. Twee masterstudenten die afstudeerden in 2009 ontwikkelden het ECC-algoritme als eindwerk. Wij zullen dit algoritme evalueren op veiligheid en proberen te verbeteren.

Materiaal en methode

Meetopstelling

Voor de ontwikkeling van het meetprogramma gebruiken we LabVIEW[®]. LabVIEW[®] is een grafische programmeertaal van National Instruments en is een veel gebruikte taal voor het realiseren van meetprogramma's.

Voordat het ontwerp en de realisatie van de uiteindelijke meetopstelling plaatsvindt, voeren we eerst enkele experimenten uit. Deze experimenten zijn nodig om na te gaan wat de mogelijkheden zijn van LabVIEW©.

Na de realisatie van de meetopstelling volgen een aantal functionele tests. Hiervoor gebruiken we kleine algoritmes met een gekend vermogenverloop. Indien het opgemeten vermogenverloop niet overeenkomt met het verwachte vermogenverloop, passen we de meetopstelling aan.

Evaluatie en verbeteren van implementaties

Als de meetopstelling volledig werkt, beginnen we met het analyseren van het ECC-algoritme. We leggen eerst vast in welke mate het algoritme bestand is tegen bepaalde vermogenaanvallen. Daarna gaan we de implementatie bekijken en de VHDL-code aanpassen zodat de ontdekte informatie niet meer zichtbaar is in het vermogen.

Als de aanpassingen zijn uitgevoerd, moeten we onderzoeken welk effect ze hebben op de veiligheid. Hiervoor voeren we terug een aantal vermogenaanvallen uit. Indien na de aanpassing meer of andere metingen nodig zijn om de sleutel te vinden, is het algoritme veiliger geworden.

Deze stappen blijven we herhalen totdat het gewenste veiligheidsniveau behaald is. De veiligheid van een algoritme is subjectief en is voor elke toepassing anders.

Hoofdstuk 1

Cryptologie

Sinds het ontstaan van de menselijke beschaving - en de communicatie tussen mensen - is er altijd behoefte geweest aan het geheim houden van bepaalde boodschappen voor derden. Door afspraken te maken over de ontcijfering van boodschappen konden enkel mensen die op de hoogte waren van deze afspraken de boodschap ontcijferen. Voor eventuele vijandige stammen was het onmogelijk de geheimen van de stam te achterhalen.

De eigenlijke betekenis van het woord 'cryptologie' kunnen we terugvinden bij de oude Grieken. Het woord bestaat uit twee delen: 'crypto' stamt af van het woord 'kryptos' en betekent verborgen. Het tweede deel 'logy' is afkomstig van het woord 'logos' wat niet meer betekent als woord. De oorspronkelijke betekenis van cryptologie is dus het verbergen of geheim houden van woorden.

1.1 Geschiedenis van de cryptologie

'The codebook' van Simon Singh [6] is een boek dat de cryptografie doorheen de geschiedenis beschrijft aan de hand van historische voorbeelden. Het boek gaat over het gebruik maar ook over de gevolgen van cryptografie en de consequenties die het kraken van de code met zich meebrengen. Dit boek was een zeer interessante basis voor 1.1.1 en 1.1.2.

Naast dit boek gebruikte we ook enkele andere bronnen. Voor informatie over het ontstaan van de cryptologie raadpleegden we het artikel 'History of cryptografie' van Wikipedia [1]. Informatie over de enigma, beschreven in 1.1.2, vonden we terug in 'Tijdslijn van de codeermachines' van D.Rijmenants [2].

1.1.1 Ontstaan van de cryptologie

Archeologen ontdekten het gebruik van cryptologie voor het eerst in hiërogliefen die afgebeeld staan in monumenten van de oude Egyptenaren (ongeveer 4500 jaar oud). De Egyptenaren gebruikte deze hiërogliefen niet echt om geheime informatie door te geven, maar eerder om mysterie op te wekken bij de gewone bevolking. Toch beschouwen wetenschappers deze periode als het begin van de cryptologie, als het ware in een zeer primitieve vorm.

Enkele jaren later vonden archeologen voorwerpen uit Mesopotamië waaruit ze konden afleiden dat de mensen uit die tijd deze voorwerpen gebruikten om informatie geheim te houden. Zo vonden deze archeologen oude kleitabletten waarop cryptische omschrijvingen van bepaalde recepten gegraveerd stonden. Deze recepten moesten vermoedelijk geheim blijven omdat ze commercieel waardevol waren.

Verder weten we dat Hebreeuwse wetenschappers rond 500-600 v.Chr. gebruik maakten van een eenvoudig substitutieprincipe om bepaalde boodschappen geheim te houden. Door de letters van de boodschap te vervangen door de volgende letter van het toen bestaande alfabet, konden ze tot een cryptische boodschap komen.

Een minder pure vorm van cryptologie is terug te vinden bij de oude Grieken. Zij trachtten boodschappen verborgen te houden door middel van ze te graveren in een houten plaat en hierover dan een dikke laag wax te leggen. Een tweede manier die ze vaak gebruikte was het tatoeëren van de boodschap op het hoofd van een slaaf. De tatoeage werd bedekt door hun haargroei. Deze twee methodes zijn echter slechte voorbeelden van cryptologie omdat de boodschap - eens dat de beschermlaag weggehaald wordt - zo te lezen is.

1.1.2 Cryptologie in oorlogstijd

Cryptologie heeft door heel de geschiedenis een grote rol gespeeld, maar kende een echte booming tijdens en tussen de twee wereldoorlogen. In deze periode was het essentieel om zoveel mogelijk belangrijke informatie (onder andere strategische plannen) geheim te houden voor de vijand. Wetenschappers van die tijd hebben allerlei encodeer/decodeer apparaten uitgevonden om de geheimhouding zo goed mogelijk te kunnen garanderen.

Een van deze apparaten was de Enigma. Dit codeer/decodeer apparaat heeft een zeer belangrijke invloed gehad op de oorlogvoering tijdens de tweede wereldoorlog. Daarom is het de moeite waard deze even toe te lichten. Figuur 1.1 toont een enigma met drie rotoren.



Figuur 1.1: Een 3-rotor Wehrmacht Enigma in het Imperial War Museum, London.

De enigma is een elektromechanisch codeer toestel. De belangrijkste onderdelen van deze machine waren het toetsenbord, het lampenbord, de rotors en het schakelbord. Het toetsenbord zet de ingetypte letters om naar elektrische signalen die door het apparaat vloeien en uiteindelijk een letter doen oplichten op het lampenbord.

De kracht van deze machine is de wijze waarop de versleuteling gebeurt. De operator stelt de drie rotoren in op een bepaalde positie en kiest daarna enkele vaste verbindingen op het schakelbord. Telkens wanneer de operator een letter typt op het toetsenbord, draaien de rotoren één positie verder. De codering is dus voor elke letter anders. De ontvanger kan enkel de boodschap terug ontcijferen indien hij de beginpositie van de rotoren kent en de stand van het schakelbord.

In de loop van de oorlog heeft men nog andere types met meerdere rotoren gemaakt, maar het principe bleef altijd hetzelfde. Door de verhoging van het aantal rotoren wordt enkel het 'breken van de code' moeilijker.

1.2 Hedendaagse cryptologie

Hedendaagse cryptologie is meer dan alleen het geheimhouden van informatie.

M. Vandenwauver [3] geeft meer informatie over de nieuwe aspecten van de hedendaagse cryptografie.

1.2.1 Aspecten

Bij de opkomst van de computer en de steeds verdere ontwikkeling van datacommunicatiesystemen in de jaren '60, was er nood aan een techniek om digitale informatie te beveiligen tegen misbruik. Dit is dan ook meteen het begin van de hedendaagse cryptologie.

De hedendaagse cryptologie is uitgegroeid tot een wetenschap die zich bezighoudt met alles wat te maken heeft met het voorkomen van misbruik van digitale informatie. Het is dus niet enkel meer de wetenschap die zich bezighoudt met het verborgen houden van boodschappen voor derden, maar bevat ook een heleboel andere aspecten:

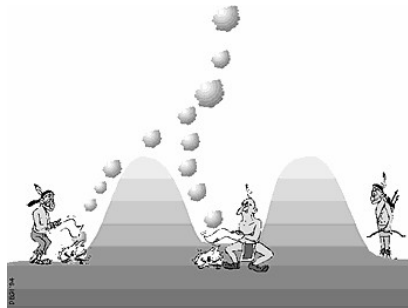
- **User authentication:** deze tak doet onderzoek naar technieken en algoritmes om de echtheid van de gebruikers te garanderen. Dit is de eerste stap in het communicatieproces. De zender/ontvanger wil weten of de ontvanger/zender degene is die hij beweert dat hij is. Deze identificatie kan op verschillende manieren: een paswoord, pincodes, magneetkaarten, biometrische gegevens, ... Figuur 1.2 stel user authentication op een ludieke manier voor.



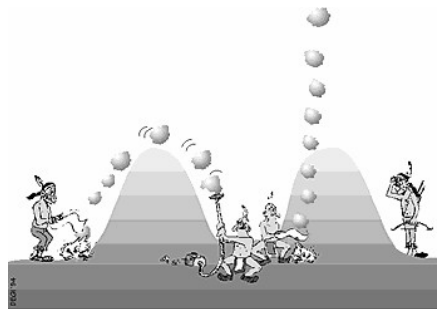
Figuur 1.2: Voorstelling van user authentication door M.Vandenwauver

Voorbeeld: Wanneer iemand wil inloggen op het computersysteem van een bank, moet hij eerst zichzelf identificeren. Dit gebeurt met een code die een toestel genereert aan de hand van de bankkaart, de pincode en een challenge. Deze methode is veel veiliger dan de methode die de bank vroeger gebruikte (gebruikersnaam-wachtwoord combinatie)

- **Data authentication:** Deze tak doet onderzoek naar technieken en algoritmes om de echtheid van de data te verzekeren. Hiervoor moet er voldaan worden aan twee voorwaarden: de ontvanger moet zeker zijn dat de data niet gewijzigd is na verzending (**data integrity**) en hij moet weten wie de zender is (**data origin authentication**).



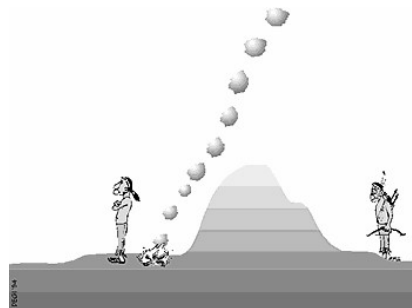
Figuur 1.3: Voorstelling van data integrity door M.Vandenwauver



Figuur 1.4: Voorstelling van data origin authentication door M.Vandenwauver

Voorbeeld: Wanneer je software (niet allemaal) installeert op een computer krijg je soms de melding dat de software van een onbekende uitgever is en stelt de computer de vraag of je deze software echt wil installeren. Dit is een voorbeeld van data authentication. De computer kent de schrijver van de software niet en vraagt zich af of de data wel te vertrouwen is.

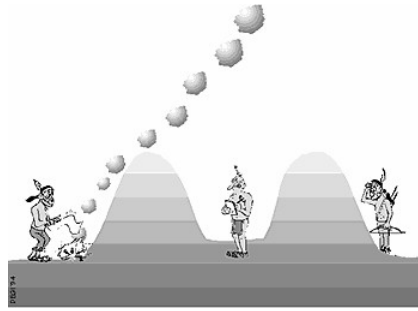
- **Non-repudiation:** Deze tak doet onderzoek naar technieken en algoritmes die 'meningsverschillen' tussen zender en ontvanger oplossen. Wanneer een van beide partijen beweert iets niet verzonden/ontvangen te hebben, moet een zeggende onafhankelijke derde partij optreden als 'bemiddelaar'.



Figuur 1.5: Voorstelling van non-repudiation door M.Vandenwauver

Voorbeeld: Veronderstel dat de eigenaar van een online-winkel beslist om bestellingen via mail te aanvaarden, dan is het belangrijk dat er een toezichter is die kan bevestigen dat de klant effectief een bestelling geplaatst heeft. Anders kan de klant beweren dat hij nooit iets besteld heeft. In de 'papier en pen' wereld wordt non-repudiation o.a. voorzien door middel van een handtekening.

- **Data confidentiality:** Deze tak doet onderzoek naar technieken en algoritmes om data geheim te houden voor onbevoegden. Met data confidentiality proberen we onszelf te beschermen tegen het ongewenst lezen van een bericht. Het bericht mag onderschept worden, zolang de onderschepper geen informatie uit het bericht kan halen.



Figuur 1.6: Voorstelling van data confidentiality door M.Vandenwauver

Voorbeeld: Wanneer we banktransacties doen over zowel internet als elk ander medium, moet de bank ten alle tijden zorgen voor de geheimhouding van de transactie. Dit moet gegarandeerd worden om privacy redenen. Encryptie zorgt voor deze geheimhouding.

Hedendaagse cryptologie is de studie (theoretisch) en de implementatie (praktisch) van technieken en algoritmes die bijdragen tot een betere data authentication, user authentication, non-repudiation en data confidentiality. Hedendaagse cryptologie is in het leven geroepen om misbruik van digitale data preventief tegen te gaan en te detecteren.

1.2.2 Ontwikkeling

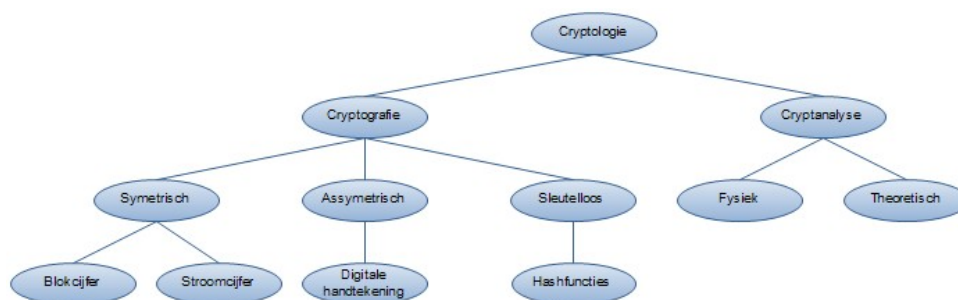
Wetenschappers hebben de laatste 50 jaar niet stil gezeten en zijn voortdurend bezig geweest met zoeken naar nieuwe en betere technieken om te voldoen aan de vier basisaspecten van de hedendaagse cryptologie. Hieronder sommen we de belangrijkste mijlpalen even op:

- In de beginjaren '70 beschrijft IBM een cryptografisch mechanisme in de DES [20] standaard. Een variant wordt gebruikt door verschillende financiële organisaties.
- In 1976 publiceerden Diffie & Hellman 'The new directions in cryptography' [21]. Dit werk introduceerde het revolutionaire concept van de 'public-key'en beschreef ook een ingenieuze methode om de sleutels uit te wisselen tussen de systemen over een onveilig kanaal. De auteurs hadden echter nog geen praktische uitvoering voor hun methode.

- In 1978 demonstreerden Rivest, Shamir & Adleman de eerste praktische realisatie van het eerder beschreven 'public-key' en 'signature' concept van Diffie & Hellman. Deze realisatie maakte gebruik van wiskundige bewerkingen met zeer grote getallen. Dit encryptieschema wordt het RSA schema genoemd.
- In 1985 ontwikkelde ElGamal nog andere krachtige en praktische 'public-key' algoritmes. Deze waren gebaseerd op een discreet logaritmisch probleem.
- In 1985 werd cryptografie over elliptische krommen beschreven door Neal Koblitz en Victor S. Miller. Elliptische krommen cryptografie is een bepaalde vorm van asymmetrische cryptografie.
- In 1991 werd de eerste internationale standaard voor digitale handtekeningen gepubliceerd. Deze was gebaseerd op het RSA schema.
- In 1994 werd de tweede internationale standaard voor digitale handtekeningen gepubliceerd. Deze was gebaseerd op het ElGamal schema.
- De zoektocht naar nieuwe algoritmes gaat nog steeds verder en men probeert permanent de veiligheid van de informatie te verhogen.

1.2.3 Indeling

Het is moeilijk om een volledige indeling van de cryptologie te maken omwille van de voortdurende uitbreidingen en de talloze technieken die gebruikt worden. Toch hebben we geprobeerd om een grove indeling te maken. Figuur 1.7 geeft schematisch deze indeling



Figuur 1.7: Indeling van de cryptologie

De eerste onderverdeling die we moeten maken is het verschil tussen cryptografie en cryptanalyse. In de cryptografie proberen wetenschappers nieuwe algoritmes en protocollen te bedenken voor de beveiliging van digitale informatie. In de cryptanalyse voeren ander wetenschappers aanvallen uit op deze algoritmes en protocollen om geheime informatie te achterhalen. In de cryptanalyse worden de uitgevonden algoritmes en protocols dus geëvalueerd op veiligheid om zo bepaalde problemen bloot te leggen. De wetenschappers in de cryptologie kunnen hun nieuwe algoritmes en protocollen daarop verbeteren.

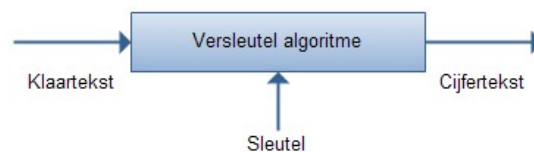
De cryptografie is onderverdeeld in drie takken. De sleutel gebaseerde cryptografie met de twee takken symmetrische cryptografie en asymmetrische cryptografie, en de sleutelloze cryptografie, als derde tak, die onder andere hashfuncties bevat. Deze begrippen staan verder uitgewerkt in 2.

De cryptanalyse kan opgedeeld worden in twee grote groepen. De eerste groep is de theoretische cryptanalyse. Hier zoeken de wetenschappers naar theoretische zwakheden in de cryptografische algoritmen en protocollen. De tweede groep is de fysieke cryptanalyse. Hier ligt de focus op het onderzoeken van zwakheden in implementaties door bijvoorbeeld de studie van de nevenkanaal informatie.

Hoofdstuk 2

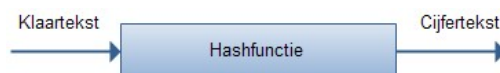
Cryptografie

Bij sleutelgebaseerde cryptografie worden sleutels in combinatie met een complex algoritme gebruikt om de klaartekst om te zetten naar de cijfertekst. Het spreekt voor zich dat enkel de zender en de ontvanger(s) bepaalde sleutels mogen kennen en dat de sleutel zeer moeilijk of niet te achterhalen is door een 'meeluisterende' buitenstaander. Figuur 2.1 geeft een schematische voorstelling van sleutelgebaseerde cryptografie.



Figuur 2.1: Essentie van sleutel gebaseerde cryptografie

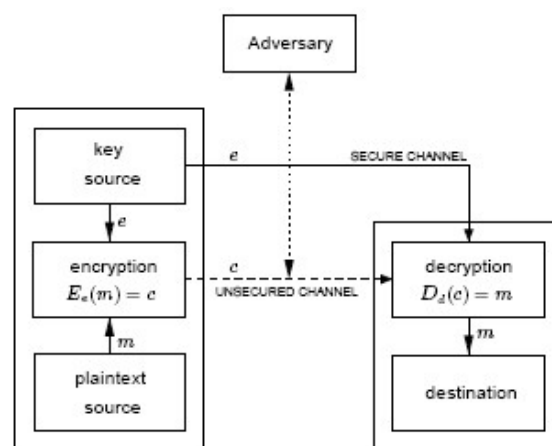
Een tweede tak in de cryptografie houdt zich bezig met het vercijferen van boodschappen zonder gebruik te maken van een sleutel. Een voorbeeld hiervan is hashfunctiecryptografie. Een kenmerk van een hashfunctie is dat de functie de boodschap altijd naar een even grootte cijfercombinatie omzet, onafhankelijk van de lengte van de klaartekst. Figuur 2.2 geeft een schematische voorstelling van hashfunctiecryptografie.



Figuur 2.2: Hashfunctiecryptografie

2.1 Symmetrische cryptografie

Symmetrische cryptografie is de eerste toepassing van sleutelgebaseerde cryptografie. Tot 1970 was alle encryptie en decryptie gebaseerd op symmetrische sleutelcryptografie. Zowel bij zender en ontvanger wordt dezelfde sleutel gebruikt om de informatie te encoderen/decoderen. Bij symmetrische cryptografie moet de sleutel van zender naar ontvanger gestuurd worden via een beveiligd kanaal. Zolang niemand de sleutel onderscheept, is de informatie veilig. Figuur 2.3 geeft het principeschema van symmetrische cryptografie.



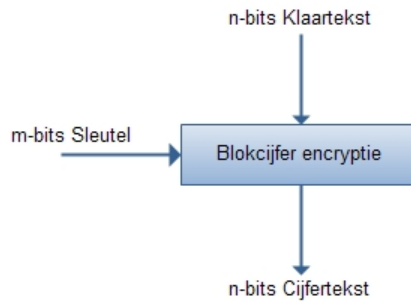
Figuur 2.3: Principeschema van symmetrische cryptografie uit 'Overview of cryptografie' door A. Menezes

De uitwisseling van de sleutels kan je zien als een brief met informatie over de ontcijfering die de zender via de post - in een gesloten briefomslag - naar de ontvanger stuurt.

Daarna gaat de zender een gecijferde boodschap in de krant plaatsen. Enkel de personen die de brief via de post ontvangen, kunnen de boodschap ontcijferen.

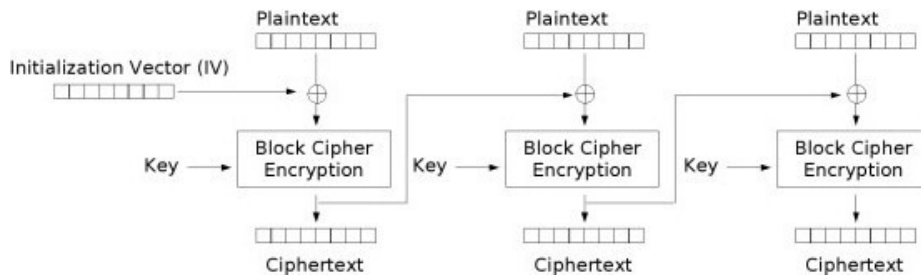
2.1.1 Blok cijfers

Een veel gebruikt encryptiemechanisme binnen de symmetrische cryptografie is het blok cijfer. Men deelt de klartekst op in blokken met een vaste grootte, die men dan met behulp van een sleutel en een encryptie-algoritme gaat versleutelen. Figuur 2.4 geeft het principeschema van een blok cijfer.

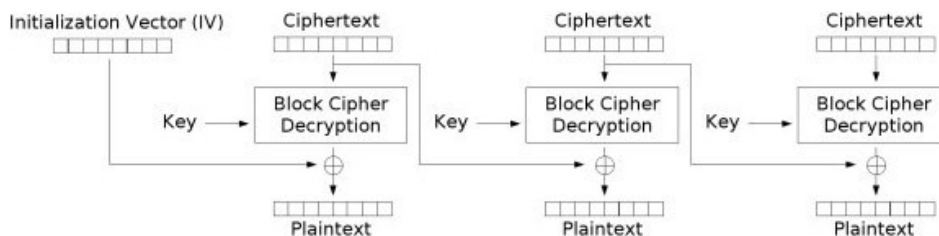


Figuur 2.4: Principeschema van een blokcijfer

Wanneer de zender meerdere blokken met dezelfde sleutel gaat encrypteren wordt het data-pad niet verborgen gehouden. Daarom heeft men allerhande methodes bedacht. Deze methodes zorgen ervoor dat elk blok telkens anders geëncrypteerd wordt. Hierdoor kan het datapatroon moeilijk achterhaald worden. De verschillende methodes worden 'modes of operation' genoemd. Een veel toegepaste methode is het in rekening brengen van de vorige geëncrypteerde blok om de nieuwe blok te encrypteren. Bij de eerste encryptie is er echter wel een initialisatievector nodig, die zowel bij zender als ontvanger bekend moet zijn. Deze methode wordt CBC-mode genoemd. Figuur 2.5 en figuur 2.6 geven het encryptie- en decryptieschema voor CBC-mode.



Figuur 2.5: Encryptie bij CBC-mode blokcijfers uit 'Blockcipher modes of operation', Wikipedia



Figuur 2.6: Decryptie bij CBC-mode blokcijfers uit 'Blockcipher modes of operation', Wikipedia

Andere veel gebruikte methodes bij blokcijfers zijn ECB, CFB, OFB, ...

Een aantal veelgebruikte blokcijfers zijn:

- **DES:** [20] Zoals vermeld in 1.2.2 werd deze standaard gepubliceerd in 1977, maar omwille van zijn 56-bit sleutel werd deze standaard onveilig verklaard voor praktische applicaties. Voor de praktische veiligheid te garanderen werd de 'triple DES' of TDEA beschreven. Dit is niet meer dan de DES standaard drie keer in cascade geschakeld.

Omdat de DES standaard zo'n belangrijk rol speelde in de voorbije jaren, lichten we deze standaard verder toe in 2.1.2.

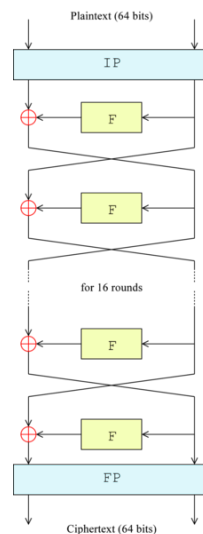
- **AES:** [22] Voor bepaalde applicaties liep TDEA tegen zijn grenzen aan. Daarom dat het NIST de AES standaard lanceerde. Deze standaard beschrijft een blokcijfer encryptie algoritme voor sleutels van 128 tot 256 bits lang.

2.1.2 De Data Encryptie Standaard of DES

De Data Encryptie Standaard is een blokcijfer dat door het NBS geselecteerd werd als standaard voor geheimhouding van digitale informatie in de Verenigde Staten. Deze standaard werd in 1975 gepubliceerd en gestandaardiseerd door het NIST in 1977. De standaard is verder uitgegroeid tot een veel verspreide standaard die moest zorgen voor een geheimhouding van allerhande digitale informatie.

Het algoritme

De DES is gebaseerd op een cryptografisch algoritme dat een sleutel met een lengte van 56-bits gebruikt. Figuur 2.7 geeft kort de werking van deze standaard weer.



Figuur 2.7: [20] Blokschema van DES uit 'Data Encryption Standard'

In figuur 2.7 is zichtbaar dat er 16 identieke trappen, één initiële permutatie en één finale permutatie is. De twee permutaties hebben bijna geen cryptografisch belang maar waren blijkbaar nodig om blokken in en uit de hardware van de jaren 70 te laden.

Na de permutatie wordt het 64-bit grote blok onderverdeeld in twee blokken van 32-bits die dan kris-kras door de 16 identieke blokken worden gestuurd, tussendoor worden de twee blokken ook nog eens met elkaar ge-exor-t.

De 16 identieke blokken worden Feistel-blokken genoemd. Deze blokken bevatten een algoritme dat het hart is van de DES.

De Initiële veiligheid van DES

Wetenschappers stelde zich al van in het begin vragen over de veiligheid van DES. Men had vooral twijfels over de lengte van de sleutel, het aantal herhalingen en het 'black-box' algoritme.

Al deze twijfels werden weerlegd door IBM, de ontwikkelaar van de standaard. IBM beweerde dat men 17 jaren gedaan had over de ontwikkeling van deze standaard en dat er gedurende deze jaren ook intensief aan cryptanalyse gedaan werd.

Sommige wetenschappers wezen er ook op dat het NSA, een Amerikaanse overheidsinstelling die meewerkte aan de standaard, wel eens een algoritme kon hebben waarmee DES berichten gemakkelijk ontcijferd kunnen worden.

De hedendaagse veiligheid van DES

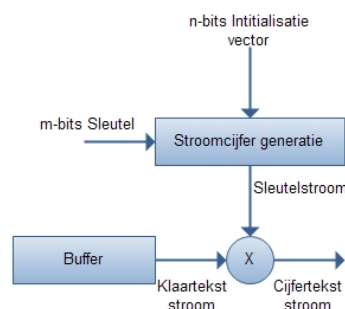
Het antwoord op de vraag 'Hoe veilig is DES vandaag?' is vrij gemakkelijk te beantwoorden. Door de korte sleutel en de rekenkracht van de hedendaagse computers is de sleutel van een DES algoritme met een brute kracht aanval zeer snel te vinden.

Er zijn gegevens over een apparaat dat in 1993 in 3,5 uur tijd, de sleutel van een DES algoritme gevonden had. Dit apparaat kostte toen 1.000.000 dollar en deze kost zal dalen met een factor 5 elke 10 jaar. Dat wil dus zeggen dat dit apparaat vandaag ongeveer 100.000 dollar kost.

Om veiligheidsredenen is men dus gaan zoeken naar opvolgers voor DES. Voorbeelden hiervan zijn Multiple DES, DESX, CRYPT(3), RDES, ...

2.1.3 Stroomcijfers

Een tweede veelgebruikt encryptiemechanisme is een stroomcijfer. Een stroomcijfer is vooral interessant wanneer men te maken heeft met een stroom van data die tegen een hoge snelheid geëncrypteerd moeten worden. Bij stroomcijfers worden geen blokken data geëncrypteerd zoals bij blokcijfers, maar worden de bits van de klaartekst één voor één geëncrypteerd. Net zoals bij de 'modes of operation' van een blokcijfer is ook bij een stroomcijfer een initialisatievector nodig die zowel bekend is bij zender als ontvanger. Figuur 2.8 geeft het principeschema van een stroomcijfer weer.



Figuur 2.8: Principeschema van een stroomcijfer

Een aantal veelgebruikte stroomcijfers zijn:

- **RC4:** [23] Dit stroomcijfer werd ontworpen door Ron Rivest in 1987. Het stroomcijfer maakt gebruik van een 8-bits initialisatievector en een sleutel die kan variëren tussen 8 en 2048 bits. Deze standaard is zeer snel en vrij veilig.
- **A5/1:** [24] Deze standaard wordt veel gebruikt voor het beveiligen van draadloze netwerken, mobiele-communicatie en bluetooth. Het is een stroomcijfer met een 54-bits grote sleutel en een 114-bits grote initialisatievector.

Omdat veel van de gebruikte stroomcijfers al onveilig zijn bevonden, schreven wetenschappers enkele jaren geleden een wedstrijd uit, dit om nieuwe veilige stroomcijfers te vinden. Een van deze kandidaat stroomcijfers, waar hoge verwachtingen voor gekoesterd worden, is TRIVIUM. Dit nieuw algoritme heeft al verscheidene aanvallen overleefd en blijkt tamelijk veilig te zijn.

2.2 Asymmetrische cryptografie

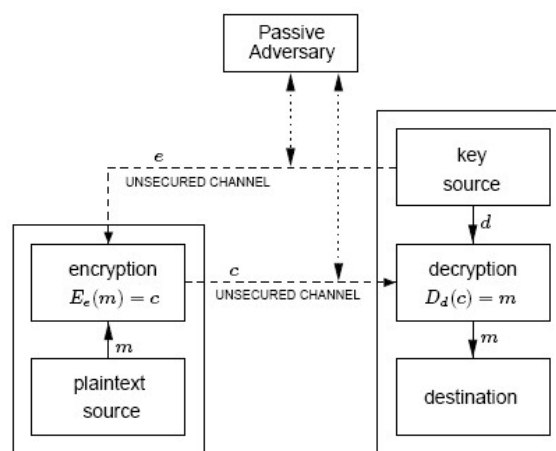
Asymmetrische cryptografie is gebaseerd op een wiskundige functie die men de 'trapdoor one-way function' noemt. Bij dergelijke functie is het zeer moeilijk om het omgekeerde van de functie te vinden. Wanneer er extra informatie over de functie vrijgegeven wordt, is het vrij eenvoudig om het omgekeerde van deze functie te vinden.

In 1976 gebeurde er een grote doorbraak in de cryptografie toen Diffie & Hellman [21] een methode introduceerde om sleutel informatie over een publiek kanaal te verzenden. Dit was het begin van asymmetrische cryptografie, dat als volgt werkt:

- Zowel zender en ontvanger bezitten twee sleutels. Een private en een publieke sleutel. De private sleutel is enkel gekend door de eigenaar. De publieke sleutel kan aan derden bekend gemaakt worden.
- Wanneer de zender een geheim bericht wil verzenden naar de ontvanger, gaat de zender de boodschap coderen met de publieke sleutel van de ontvanger. Deze sleutel heeft hij verkregen over het publieke kanaal. Iedereen kan dus in bezit zijn van deze sleutel.

- Wanneer de ontvanger het versleutelde bericht ontvangen heeft, gaat hij dit decoderen met behulp van zijn private sleutel. Derden die aan het 'meeluisteren' waren kunnen het bericht niet decoderen omdat ze enkel in bezit zijn van de publieke sleutel en niet van de private sleutel.

Zoals we in 1.2.2 vermeldde, hadden Diffie & Hellman nog geen praktische implementatie van de techniek. De eerste praktische realisatie van de 'public-key' cryptografie was een feit in 1978. Deze realisatie werd het 'RSA encryptie schema' genoemd. Een prinseschema hiervan wordt getoond in figuur 2.9.



Figuur 2.9: Prinseschema van asymmetrische cryptografie uit 'Overview of cryptografie' door A. Menezes

2.2.1 Het RSA encryptie schema

Het RSA encryptieschema is wiskundig zeer veilig. De sleutel wordt gegenereerd op basis twee priemgetallen en werkt als volgt:

- Kies twee priemgetallen die geheim gehouden worden. Deze priemgetallen mogen niet te klein zijn. Ze moeten minstens 150 cijfers groot zijn.
- Vermenigvuldig deze twee priemgetallen met elkaar, zo krijg je een zeer groot geheel getal. Dit getal maak je aan iedereen bekend en is een deel van de publieke sleutel. Het is praktisch onmogelijk om uit dit getal de twee gebruikte priemgetallen te halen omwille van de grootte. Een netwerk van computers doet er jaren over om uit dit getal de gebruikte priemgetallen te halen.

- Het tweede deel van de publieke sleutel bestaat uit een vercijferexponent dat gerelateerd is aan de twee priemgetallen. Ook dit getal wordt aan iedereen meegedeeld.
- Iedereen die een bericht wil versturen, kan nu via een bekende formule zijn klaartekst omzetten naar een cijfertekst.

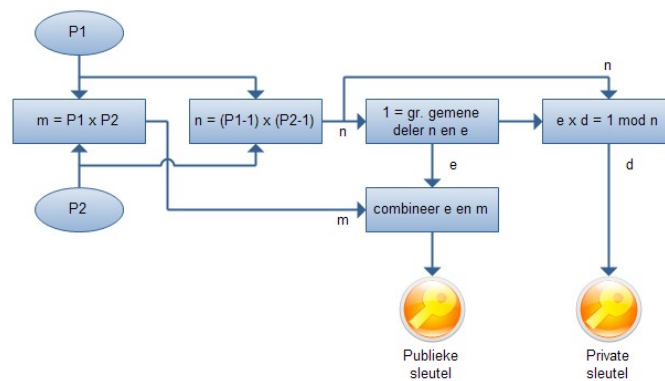
$$x^e \bmod m$$

Hierbij is **m** de modulus in de publieke sleutel, **e** de vercijferexponent en **x** de klaartekst. Een aanvaller die het versleutelde bericht onderschept kan deze formule, samen met de publieke sleutel, niet gebruiken om het originele bericht te ontcijferen. Hij loopt dan tegen het 'RSA-probleem', een puzzel die bij zulke grote getallen zo moeilijk is dat de huidige wiskunde hem niet kan oplossen.

- Bereken de ontcijferexponent aan de hand van de twee priemgetallen en de vercijferexponent. De ontcijferexponent is dan de private sleutel.
- Aan de hand van een bekende formule kan het versleutelde bericht ontcijferd worden.

$$y^d \bmod m$$

Hierbij is **m** de modulus in de publieke sleutel, **d** de ontcijferexponent of private sleutel en **y** de cijfertekst. Omdat de ontcijferexponent enkel bekend is bij de ontvanger, zal een aanvaller de boodschap niet kunnen ontcijferen. Figuur 2.10 geeft schematisch weer hoe we een RSA-sleutel kunnen berekenen.

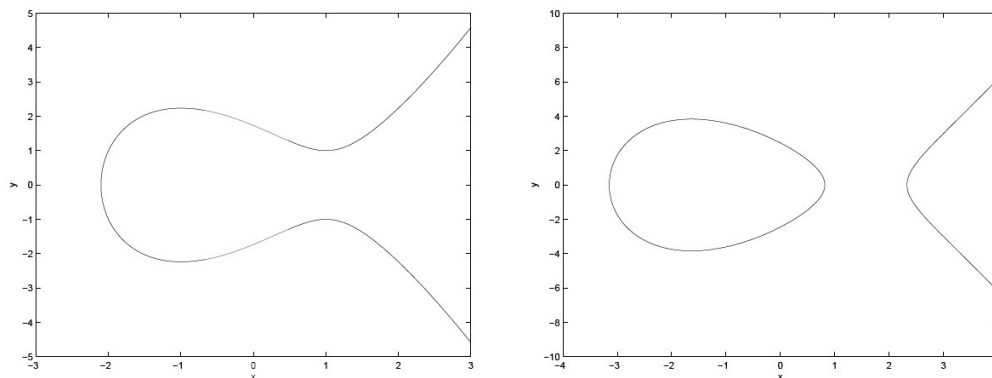


Figuur 2.10: Berekenen van de RSA sleutels

2.2.2 ECC encryptie

Elliptische kromme cryptografie is een benadering van de publieke sleutelcryptografie die gebaseerd is op elliptische krommen in eindige velden. Door allerlei bewerkingen te doen op deze elliptische krommen kan men een publieke en private sleutel bekomen. De sleuteluitwisseling gebeurt asymmetrisch, de data uitwisseling gebeurt symmetrisch!

Elliptische krommen worden tegenwoordig veel in de cryptografie gebruikt, waaronder bij het versleutelen van creditcardgegevens, bij mobiele telefonie en op het internet. De cryptografische systemen die gebruik maken van elliptische krommen zijn gebaseerd op een wiskundig probleem. Het is vrij gemakkelijk om bij een gegeven punt P op een elliptische kromme en een getal n het punt $n \cdot P = Q$ te vinden, maar als je P en Q weet is het moeilijk de n te vinden waarvoor $n \cdot P = Q$ geldt. Dit probleem staat bekend als het discrete logaritme probleem bij elliptische krommen. De voorgaande bewerkingen worden punt vermenigvuldiging genoemd en is niet hetzelfde als de algebraïsche vermenigvuldiging die wij kennen. Figuur 2.11 is een voorbeeld van mogelijke elliptische krommen.



Figuur 2.11: Plot van twee elliptische krommen uit de masterthesis, N. Cornelissen en C. Peeters [25]

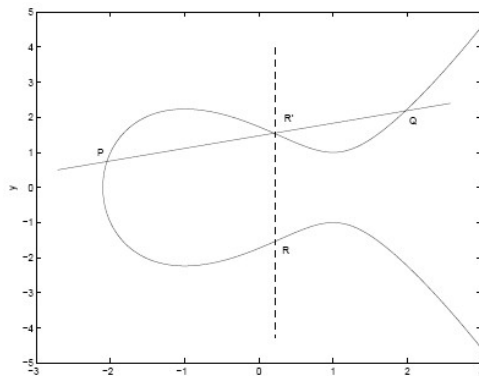
Bewerkingen op elliptische krommen

Er zijn enkele bewerkingen op elliptische krommen die we vaak gebruiken bij cryptografische toepassingen. Deze bewerkingen leggen we verder uit omwille van hun verdere belang in deze thesis.

- **Puntoptelling:** Wanneer we een elliptische kromme E beschouwen en twee verschillende niet-samenvallende punten P en Q op E nemen, kunnen we een rechte door P en Q tekenen. Deze rechte noemen we L . Uit de eigenschappen van elliptische krommen kunnen we afleiden dat de rechte nog eens snijdt met de kromme in één punt. Dit punt noemen we R' . Wanneer we dit punt spiegelen ten opzichte van de x -as krijgen we een punt dat ook op de elliptische kromme gelegen is. Dit punt is het resultaat van de puntoptelling van P en Q .

$$P + Q = R$$

Figuur 2.12 geeft aan hoe je een puntoptelling doet van een punt op de elliptische kromme.

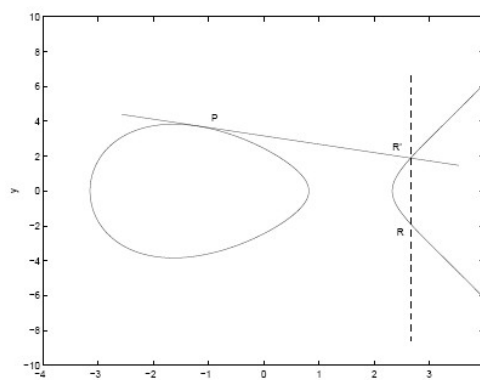


Figuur 2.12: Puntoptelling bij elliptische krommen uit de masterthesis, N. Cornelissen en C. Peeters [25]

- **Puntverdubbeling:** Een punt P kan ook opgeteld worden bij zichzelf. Dit is de bewerking puntverdubbeling. Deze verdubbeling verloopt ongeveer hetzelfde als de puntoptelling. Neem de raaklijn aan de kromme in het punt P . Deze raaklijn snijdt de elliptische kromme exact in een ander punt R' . Wanneer dit punt gespiegeld wordt ten opzichte van de x -as verkrijgen we het punt R . Dit punt R noemen we het dubbele van P .

$$P + P = R$$

Figuur 2.13 geeft aan hoe je een puntverdubbeling doet van een punt op de elliptische kromme.



Figuur 2.13: Puntverdubbeling bij elliptische krommen uit de masterthesis, N. Cornelissen en C. Peeters [25]

- **Puntvermenigvuldiging:** De belangrijkste bewerking in de elliptische kromme cryptografie is de elliptische kromme puntvermenigvuldiging. Wanneer we een punt P op de kromme en een geheel getal 't' nemen, kunnen we de puntvermenigvuldiging berekenen door een reeks punt optellingen en puntverdubbelingen achter elkaar te doen.

Voorbeeld

Cryptografie met elliptische krommen is gemakkelijker te begrijpen aan de hand van een voorbeeld. Stel dat een zender en ontvanger met elkaar willen communiceren en nog geen sleutel hebben afgesproken.

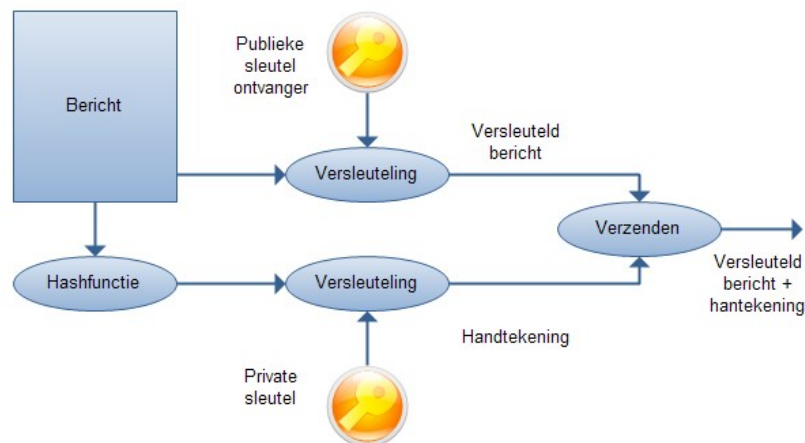
- Zender en ontvanger spreken af welke elliptische kromme ze gaan gebruiken en kiezen een punt P op deze elliptische kromme. Deze informatie wordt publiek gemaakt en mag dus geweten zijn door iedereen.
- De zender kiest hierna een geheim geheel getal. Dit kunnen we beschouwen als de private sleutel.
- De zender berekend hierna de puntvermenigvuldiging van het afgesproken punt P met het geheime geheel getal n , door hem zelf gekozen. Het resultaat wordt Q_{zender} genoemd. Dit is dan de publieke sleutel en wordt bekend gemaakt aan de ontvanger.
- Ook de ontvanger kiest een geheim geheel getal m en gaat ook de puntvermenigvuldiging van het afgesproken punt P met het geheime geheel getal m doen. Het resultaat wordt $Q_{ontvanger}$ genoemd. Deze publieke sleutel wordt ook bekend gemaakt aan de zender.
- Zowel aan zender- als ontvangerzijde gaat de publieke sleutel van de tegenpartij vermenigvuldigd worden met de eigen private sleutel (het gekozen getal). Het resultaat van deze vermenigvuldiging wordt dan de sleutel.
$$sleutel = Q_{zender} \cdot m = Q_{ontvanger} \cdot n$$
- Nu is zowel aan zenderzijde als ontvangerzijde de symmetrische sleutel bekend en kan de communicatie beginnen.

2.2.3 Digitale handtekening

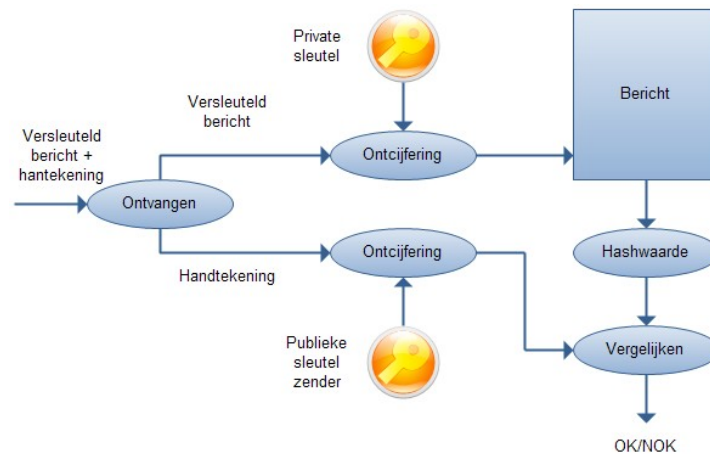
Bij asymmetrische cryptografie is het mogelijk dat de zender zich kan identificeren bij de ontvanger. Zo weet de ontvanger dat de informatie werkelijk komt van de zender die hij beweert te zijn. Eén van de principes die gebruikt wordt is vrij gemakkelijk te begrijpen:

- Stel dat een zender een bericht naar een ontvanger wil zenden en dat het nodig is om zijn identiteit bekend te maken aan de ontvanger.
- De zender versleutelt het bericht en gaat op het einde de hashwaarde van het verzonden bericht berekenen, deze hashwaarde versleutelt hij met zijn private sleutel. De zender zendt de versleutelde hashwaarde naar de ontvanger.
- Wanneer de ontvanger het versleutelde bericht ontvangt, gaat hij de ontvangen hashwaarde decoderen met de publieke sleutel van de zender en kijken of deze waarde overeenkomt met de hashwaarde die hij zelf berekend heeft uit het ontcijferde bericht.
- Indien deze twee waarden hetzelfde zijn, kan de ontvanger er zeker van zijn dat dit bericht daadwerkelijk door de zender verzonden is naar hem en niet door derden.

Figuur 2.14 en figuur 2.15 geven aan hoe een digitaal handtekenproces in zijn werk gaat.



Figuur 2.14: Zender met berekening van digitale handtekening



Figuur 2.15: Ontvanger met verificatie van digitale handtekening

Er zijn echter een hele reeks van algoritmes bedacht om digitale handtekeningen te zetten. Een van de belangrijkste en veel gebruikt algoritme het DSA [26].

2.3 Symmetrische- vs. Asymmetrische cryptografie

Symmetrische en asymmetrische cryptografie zijn twee aparte deeltakken binnen cryptografie die hand in hand leven. Voor sommige toepassingen is het beter symmetrische cryptografie te gebruiken terwijl voor andere toepassingen beter asymmetrische cryptografie gebruikt wordt. Elk type heeft zijn voor- en nadelen. Onderstaande voor- en nadelen zijn beschreven in 'Overview of cryptography' door A. Menezes [4]

2.3.1 Voordelen van symmetrische cryptografie

- Algoritmes binnen de symmetrische cryptografie worden zo ontworpen dat ze zeer hoge data-stromen aankunnen. Sommige hardware implementaties halen snelheden tot honderden megabyte per seconden, terwijl softwarematige implementaties snelheden tot enkele megabyte per seconden halen.
- Sleutels binnen de symmetrische cryptografie zijn meestal relatief klein
- Het gebruik van symmetrische cryptografie is zeer oud. Er bestaan zeer veel toepassingen van dit soort cryptografie, wat leidt tot een zeer goede kennis van de mogelijkheden.

2.3.2 Nadelen van symmetrische cryptografie

- In een tweewegscommunicatie moeten zowel zender als ontvanger de sleutel geheimgehouden.
- In grote netwerken zijn er veel sleutelparen. Daarom is een efficiënt sleutelmanagement noodzakelijk.
- Bij symmetrische systemen moet de sleutel zeer dikwijls veranderen. Het kan zelfs zijn dat de sleutel na elke communicatiesessie verandert.
- Het gebruik van digitale handtekeningen is mogelijk maar niet aan te raden omdat hiervoor zeer grote sleutels nodig zijn.

2.3.3 Voordelen van asymmetrische cryptografie

- Zender en ontvanger hebben een private sleutel. Het is niet nodig dat de zender de private sleutel van de ontvanger weet.
- Afhankelijk van de 'modes of operation' blijft de sleutel hetzelfde gedurende een aantal sessies tot een aantal jaar.
- Het is mogelijk om met de meeste asymmetrische cryptografische schema's op een efficiënte manier digitale handtekeningen te zetten.
- In netwerken met asymmetrische cryptografie is de opslag van sleutels beperkt tot de private sleutel.

2.3.4 Nadelen van asymmetrische cryptografie

- De datasnelheid van asymmetrische systemen is doorgaans veel trager dan de datasnelheid van de symmetrische systemen.
- Sleutels van asymmetrische systemen zijn meestal veel groter dan sleutels van symmetrische systemen.
- Asymmetrische cryptografie heeft niet zo'n grote voorgeschiedenis als symmetrische cryptografie.

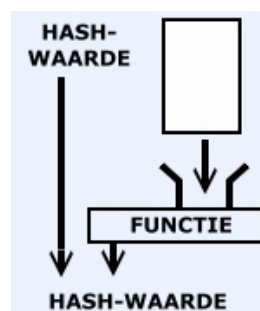
2.4 Hashfuncties

Het woord 'hash' komt uit de informatica en betekent 'hakken'. Een hashfunctie zet een klaartekst van willekeurige grootte om naar een opeenvolging van cijfers met een vaste lengte. Er wordt als het ware een vingerafdruk gemaakt van de klaartekst. De eigenschappen waaraan een hashfunctie moet voldoen zijn terug te vinden op de website van B. Heek [27].

2.4.1 Eigenschappen

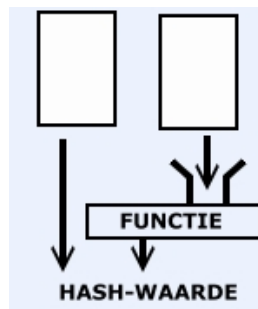
Enkel sterke hashfuncties komen in aanmerking voor cryptografische doeleinden. Een sterke hashfunctie voldoet aan drie eigenschappen:

- **one-way:** Een sterke hashfunctie is onomkeerbaar. Dit wil zeggen dat het onmogelijk is om uit een gegeven hashwaarde de klaartekst af te leiden. Figuur 2.16 is een schematische voorstelling hiervan.



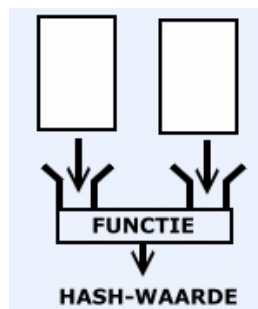
Figuur 2.16: schematische voorstelling van One-way

- **Target collision resistant:** Een sterke hashfunctie laat niet toe dat er een tweede invoer kan verzonden worden die dezelfde hashwaarde geeft. Figuur 2.17 is een schematische voorstelling hiervan.



Figuur 2.17: schematische voorstelling van target collision resistant

- **Random collision resistant:** Bij een sterke hashfunctie mogen twee verschillende invoeren niet dezelfde hashwaarde opleveren. Figuur 2.18 is een schematische voorstelling hiervan.



Figuur 2.18: schematische voorstelling van Random collision resistant

Verder zijn er ook nog een aantal eigenschappen die algemeen gelden voor hashfuncties. Zo moeten ze er willekeurig uitzien en een kleine verandering moet tot een volledig nieuwe hashwaarde leiden.

2.4.2 Hashfuncties in de praktijk

Voor cryptografische toepassingen zijn er verschillende algoritmes en standaarden bedacht. Hieronder enkele opgesomt:

- **MD5:** [28] De ingevoerde klaartekst wordt omgezet in 32 cijfers en letters. Ook wanneer er niets ingevoerd wordt, zal er een hashwaarde uit de functie komen.
- **SHA-1:** [29] De ingevoerde klaartekst wordt omgezet in 40 cijfers en letters.

Hoofdstuk 3

Cryptanalyse

3.1 Inleiding

Zoals reeds vermeld in 1.2.3 is cryptanalyse de tak van de cryptologie die probeert om de boodschap te ontcijferen op allerlei verschillende manieren. Dankzij deze wetenschap kunnen cryptanalysten de cryptografische systemen evalueren op veiligheid. Daarna kunnen cryptografen met de informatie van de cryptanalysten de systemen terug veilig maken.

Cryptanalyse is onderverdeeld in twee grote groepen. We hebben de theoretische cryptanalyse die zich bezighoudt met het zoeken naar 'gaten' in de wiskundige algoritmes en protocollen. Terwijl de fysieke cryptanalyse met behulp van allerhande methodes en nevenkanaalinformatie de boodschap probeert te ontcijferen of te manipuleren.

Verder is de fysieke cryptanalyse nog eens verdeeld in twee deeltakken. Dit heeft vooral betrekking op de aanvallen die men gaat uitvoeren. Er zijn passieve aanvallen en actieve aanvallen. Bij een passieve aanval gaat men enkel de cijfertekst monitoren en zo trachten de klaartekst te achterhalen. Bij actieve aanvallen gaat men het cryptografische toestel proberen te analyseren. Dit kan zelfs destructief zijn zoals het afschrappen van FPGA's.

In dit eindwerk gaan we niet zoeken naar theoretische fouten in de cryptografische systemen. We gaan vooral bezig zijn met de passieve tak van de fysieke cryptanalyse. Het is nodig om even te verduidelijken welke aanvallen er allemaal mogelijk zijn, wat nevenkanalen zijn en hoe we deze nevenkanalen gebruiken om het cryptografisch systeem te kraken.

Dit hoofdstuk is geschreven aan de hand van een aantal eindwerken die betrekking hebben op nevenkanaalaanvallen. Het eerste belangrijke eindwerk is de doctoraatsthesis van dr. Nele Mentens, getiteld 'Secure and Efficient Coprocessor Design for Cryptographic Applications on FPGAs' [16]. Naast de doctoraatsthesis hebben we ook de masterthesis [17] van dr. Nele Mentens gebruikt. Een derde en laatste bron die we raadpleegden was het eindwerk van Pieter Buyschaert en Elke De Mulder [18].

3.2 Aanvalmodellen

We kunnen een verdeling maken van de mogelijke passieve aanvallen. Dit doen we op basis van de informatie die voor de aanvaller beschikbaar is en welke mogelijkheden er zijn om het systeem te evalueren.

- **'Ciphertext-only' aanval:** Hier tracht de aanvaller de sleutel of de klartekst te achterhalen door middel van de cijfertekst te observeren. De aanvaller moet dus geen fysieke toegang hebben tot het systeem. Cryptografische systemen die vatbaar zijn voor dit type van aanval worden als compleet onveilig beschouwd.
- **'Known-plaintext' aanval:** Bij dit type aanval bezit de aanvaller klartekst met de bijhorende cijfertekst. Door middel van studie van deze twee, kan de aanvaller eventueel een verband vinden waardoor hij de sleutel kan bepalen. Dit type aanval is praktisch gezien moeilijk omdat de aanvaller een goede kopie van de klartekst moet zien te bemachtigen. Dit is op zich vrij moeilijk. Deze aanval wordt meestal gebruikt tijdens onderzoek naar veiligheid van het algoritme.
- **'Chosen-plaintext' aanval:** Deze aanval kan enkel gedaan worden indien de aanvaller fysiek toegang heeft tot het encryptiesysteem en hier zijn eigen gekozen klartekst kan aanleggen. Door deze klartekst en de bijhorende cijfertekst dan te analyseren, kan hij/zij de sleutel achterhalen.
- **'Chosen-ciphertext' aanval:** Deze aanval is het tegengestelde van de chosen-plaintext aanval. Hier gaat de aanvaller een gekozen cijfertekst aanleggen aan een decodeer systeem. Door de aangelegde cijfertekst en de bijhorende klartekst te analyseren, kan de aanvaller de sleutel achterhalen. Ook hier heeft de aanvaller fysieke toegang tot het decodeer systeem.

3.3 Aanvalsmethoden

3.3.1 Brute kracht

Het is mogelijk om bepaalde cryptografische algoritmes te kraken met behulp van brute kracht. Met brute kracht bedoelen we dat we alle mogelijke combinaties van de sleutel uitproberen totdat er een zinnige klaartekst tevoorschijn komt. Vooral cryptografische systemen die een kleine sleutel gebruiken of niet regelmatig hun sleutel vernieuwen zijn kwetsbaar voor dit soort aanval.

3.3.2 Nevenkanalen (side channels)

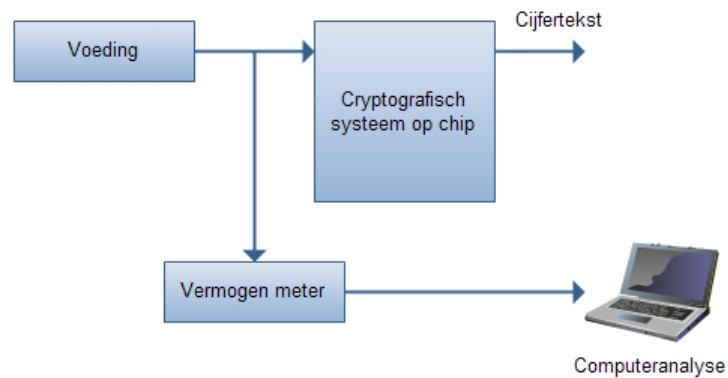
Theoretisch kan een cryptografisch systeem 100% veilig zijn, maar daarom is de implementatie ervan nog niet veilig. Omdat moderne cryptografie gebeurt met elektronica, zijn er ook nog heel wat nevenkanalen zoals temperatuur, geluid, elektromagnetische straling, vermogenverbruik, enz.

Het is mogelijk geworden om een volledig veilig systeem te kraken dankzij deze nevenkanalen. Door de nevenkanalen goed te analyseren, kan de aanvaller sleutels achterhalen. Het spreekt voor zich dat het cryptografisch systeem niet in handen van de aanvaller mag komen als dit niet beveiligd is tegen nevenkanaalaanvallen.

In dit eindwerk gaan we gebruik maken van een vermogenanalyse op een FPGA waarin een cryptografisch algoritme geprogrammeerd is. Met deze vermogenanalyse trachten we de sleutel van het systeem te achterhalen.

3.4 Vermogenanalyse

Cryptografische systemen van deze tijd zijn opgebouwd met behulp van elektronische schakelingen. Deze schakelingen worden gemaakt met transistoren. In rusttoestand ('1' of '0') verbruiken transistoren geen vermogen, maar wanneer deze schakelen is er wel een miniem vermogenverbruik. Uit dit vermogenverbruik kunnen we dus afleiden of er transistoren schakelen of niet. Figuur 3.1 laat zien hoe cryptanalysten vermogenaanvallen meestal uitvoeren.



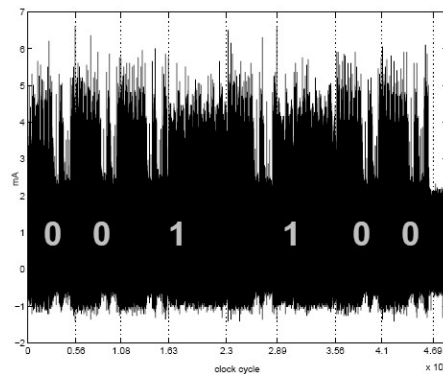
Figuur 3.1: Principeschema van een vermogenanalyse

3.4.1 Soorten

Er bestaan twee soorten vermogenanalyse aanvallen. We hebben de eenvoudige vermogenanalyse aanval en de differentiële vermogenanalyse aanval. Beide aanvallen werden voor het eerst beschreven en gepubliceerd in 1998 door Paul Kocher [19].

SPA

De 'simple power-analysis attack' (SPA) legt een rechtstreeks verband tussen het gemeten vermogen en de uitgevoerde bewerkingen. Bij dit type aanval nemen we aan dat elke instructie een unieke vorm in het vermogenverbruik heeft. Een aanvaller kan dus simpelweg het vermogenverbruik van het systeem bekijken en zo de volgorde van uitgevoerde bewerkingen bepalen. Indien deze bewerkingen gerelateerd zijn aan de geheime sleutel, kan de aanvaller rechtstreeks de sleutel uit het vermogenverbruik aflezen. Figuur 3.2 is een voorbeeld van een SPA-aanval.

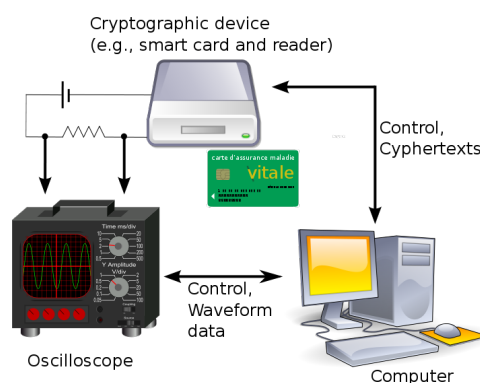


Figuur 3.2: Voorbeeld van simple power-analysis attack

In figuur 3.2 kunnen we duidelijk 2 patronen onderscheiden. Het patroon van '1' is langer en anders dan het patroon voor een '0'.

DPA

De 'Differential Power-Analysis Attack' (DPA) is moeilijker dan een SPA aanval. In tegenstelling tot SPA heb je bij DPA meerdere vermogenmetingen nodig. Toch wordt deze techniek in de praktijk zeer veel toegepast omdat er meestal weinig geweten is over het toestel dat aangevallen wordt of omdat maatregelen tegen SPA makkelijk te nemen zijn. Deze kennis was bij SPA wel nodig zelfs als er veel ruis op de vermogenmeting zit, kan een aanvaller de sleutel van het cryptografisch systeem met DPA vinden. Het basis idee achter DPA is dat de aanvaller een aantal sleutelbits gokt en dan met behulp van de cijfertekst en de vermogenmeting nagaat of de gok juist was. Figuur 3.3 laat zien met welke meetopstelling we een DPA-aanval kunnen uitvoeren.



Figuur 3.3: Voorbeeld van differentiële power-analysis attack

3.4.2 Praktisch voorbeeld

Om het principe van poweranalyse nog iets duidelijker te maken, even een praktisch voorbeeld. Bij elliptische kromme cryptografie wordt de publieke sleutel bepaald door een puntvermenigvuldiging van een gekozen punt P op de elliptische kromme met de private sleutel. We kunnen een puntvermenigvuldiging als een reeks van puntoptellingen en puntverdubbelingen schrijven. Afhankelijk van de waarde van de geheime sleutel zal een van deze bewerkingen uitgevoerd worden. Figuur 3.4 toont een algoritme dat gebruikt wordt om een publieke sleutel te berekenen bij ECC-cryptografie.

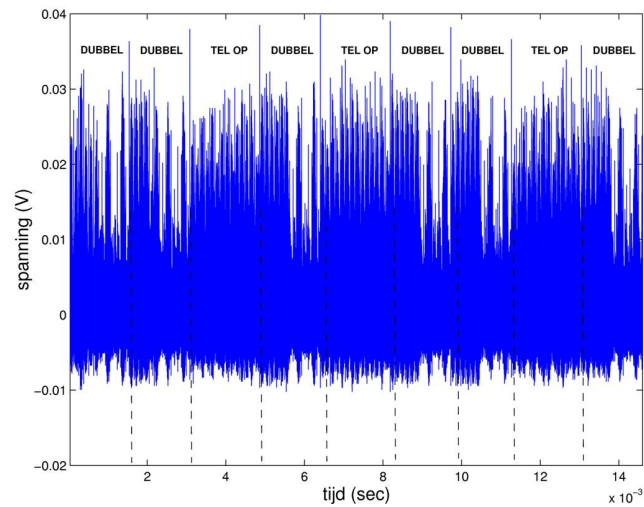
Algoritme voor het bepalen van public key bij elliptische kromme cryptografie

Geheime sleutel: k
Gekozen punt op kromme: P
Lengte van de sleutel: l

```
1: for i from l - 2 downto 0 do
2:   If  $k_i = 1$  then
3:      $P_{tussen} = 2.P$ 
4:      $P = P + P_{tussen}$ 
5:   Else
6:      $P = 2.P$ 
7:   end for
8:  $k_{public} = P$ 
```

Figuur 3.4: Algoritme voor het bepalen van de publieke sleutel bij elliptische krommen cryptografie uit masterthesis, N. Cornelissen en C. Peeters [25]

Wanneer we het vermogen tijdens het berekenen van de publieke sleutel gaan bekijken, zien we telkens twee patronen herhaaldelijk terugkomen. Er wordt een puntverdubbeling uitgevoerd wanneer de sleutelbit 0 is. En een puntverdubbeling gevolgd door een puntoptelling wanneer de sleutelbit 1 is. Figuur 3.5 laat het vermogenverbruik van de FPGA zien tijdens de uitvoer van het algoritme uit figuur 3.4.



Figuur 3.5: Vermogenvorm tijdens een algoritme dat de publieke sleutel berekend bij elliptische kromme cryptografie

Zoals zichtbaar is in bovenstaande figuur kan je de sleutel gewoon afleiden uit de golfvorm. In dit geval is de waarde van de sleutel 01101. Wat hierboven beschreven staat is dus een voorbeeld van SPA.

Hoofdstuk 4

Vermogenanalyse meetopstelling - Ontwerpfase

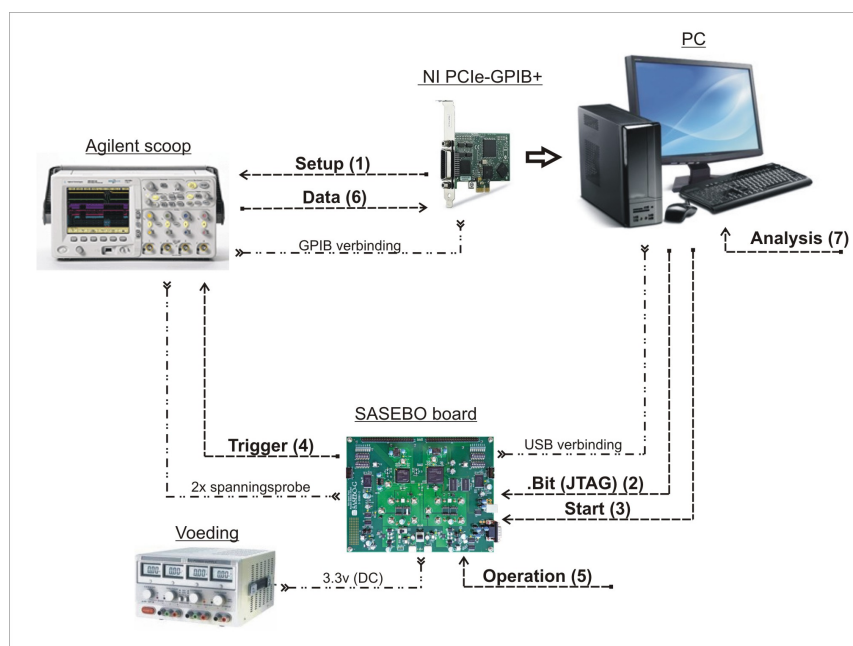
4.1 Inleiding

Om een geslaagde vermogenaanval uit te voeren, moet er een gespecialiseerde meetopstelling beschikbaar zijn. De meetopstelling moet het vermogenverbruik van de FPGA, die de cryptografische bewerkingen uitvoert, nauwkeurig kunnen opnemen. Figuur 4.1 toont het blokschema van zo'n meetopstelling.

De meetopstelling start meerdere keren de cryptografische bewerkingen met een gekende klaartekst die kan veranderen. Tijdens elke cyclus meet de opstelling het vermogenverbruik van de FPGA. Hoe minder cycli er nodig zijn om de sleutel te achterhalen, hoe minder veilig de bewerking is.

Het is belangrijk dat de meetopstelling volledig automatisch de metingen uitvoert. De meetgegevens dienen opgeslagen te worden zodat een computerprogramma ze nadien kan verwerken. Figuur 4.1 geeft de schematische werking van de meetopstelling weer.

Nadat een cryptografisch algoritme ingeladen is in de FPGA en alle parameters op de pc ingevoerd zijn, kan de meting starten. Eerst wordt een setup van de scoop gedaan (Stap(1)). Daarna gaat de pc een signaal geven aan de FPGA. Dit signaal start een controle algoritme dat een triggersignaal aan de scoop geeft (2) en vervolgens de cryptografische bewerkingen aanvat (3). Tijdens deze bewerkingen (4) wordt het vermogenverbruik van de FPGA opgenomen (5). Wanneer de cryptografische bewerkingen afgelopen zijn, draagt het controle algoritme de data over naar de pc. Deze gaat op zijn beurt de data opslaan in een document. Als alle metingen gedaan zijn (6), is het mogelijk de opgenomen data te analyseren met speciale software (7).



Figuur 4.1: Meetopstelling

In sectie 4.2, 4.3 en 4.4 bespreken we de gebruikte hard- en software in detail.

4.2 SASEBO - Het FPGA-bord

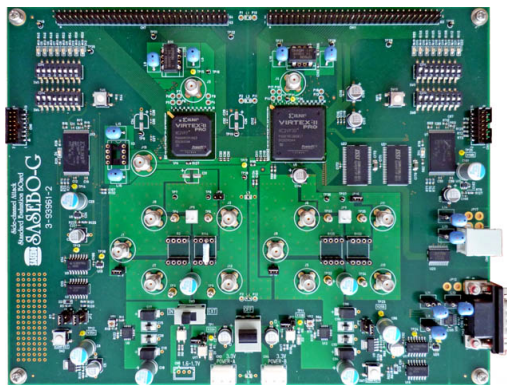
4.2.1 Inleiding

Het SASEBO [7] bord, ontworpen door het RCIS [7], is het hart van de meetopstelling. Het RCIS is een Japanse onderzoeksinstituting die fysische ondersteuning en technische adviezen geeft aan overheden en bedrijven inzake IT security.

Het RISC ontwerp het SASEBO-bord werd ontworpen in 2007. Het bord bevat twee Xilinx Virtex-II Pro FPGA's. Eén van deze FPGA's wordt gebruikt als implementatie FPGA voor cryptografische algoritmes, de andere is een communicatie FPGA. Verder zijn er een aantal voorzieningen om een poweranalyse van zowel de cryptografische als controle FPGA mogelijk te maken.

Er zijn vier verschillende versies van het SASEBO-bord. Zo hebben we SASEBO, SASEBO-G, SASEBO-B en SASEBO-R. SASEBO en SASEBO-G zijn bordes die Xilinx FPGA's bevatten terwijl SASEBO-B uitgerust is met Altera FPGA's. Deze drie bordes bevatten FPGA's met microprocessor. De vierde versie, SASEBO-R, bevat een ASIC die een hele reeks blokcijfers ondersteunt alsook het RSA protocol.

Wij hebben gekozen voor de SASEBO-G omdat we vertrouwd zijn met Xilinx FPGA's. SASEBO-G heeft naast een RS-232 verbinding ook nog een USB verbinding terwijl SASEBO enkel een RS-232 verbinding heeft. Figuur 4.2 is een foto van het bord, afkomstig van de website van het RCIS [5]



Figuur 4.2: Afbeelding van het SASEBO-G evaluatie bord

4.2.2 Specificaties SASEBO-G

De specificaties van SASEBO-G zijn terug te vinden in de gebruikshandleiding [7]:

- Het bord is gemaakt van FR-4 materiaal en de geleiders zijn verdeeld over acht lagen. Het is een vrij compact bord (23cm x 18cm), ideaal voor experimentele doeleinden;

- SASEBO-G bevat twee Xilinx Virtex-II FPGA's. De cryptografische FPGA is van het type XC2VP7 en de controle FPGA is van het type XC2VP30. De FPGA's zijn met elkaar verbonden via een 16-bits bi-directionele databus, een 16-bits adresbus en vier controlesignalen, namelijk RD, WT, RST en CLOCK;
- On-board zijn er twee oscillators aanwezig die de FPGA's voorzien van een klokfrequentie van 24MHz;
- Een externe 3.3V voeding voedt het bord. On-board zijn er verschillende spanningsregelaars die de rest van de schakeling voorzien van de juiste voedingsspanningen. Het is ook mogelijk om de cryptografische FPGA aan te sluiten op een 1.5V externe voeding;
- Er zijn shunt weerstanden en SMA-connectoren voorzien om het vermogenverbruik van de FPGA's differentieel te meten;
- De host pc bestuurt en communiceert met het bord via een RS-232 of een USB interface;
- Elke FPGA heeft een aparte JTAG-interface voor debugging en programming.

4.2.3 USB-interface

In de definitieve meetopstelling gebruiken we de USB-interface als communicatie-interface tussen de pc en het SASEBO-G bord. De USB-interface kan enkel gebruikt worden samen met de controle FPGA.

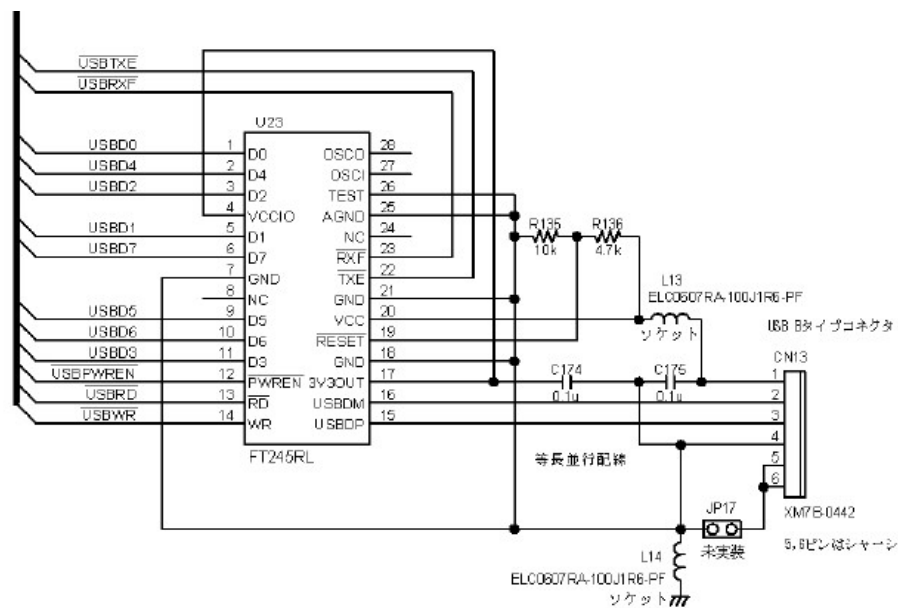
Een speciaal IC verwezelijkt de USB-functionaliteit op het SASEBO-G evaluatiebord. De FT245R is een USB naar parallele FIFO IC ontworpen en geproduceerd door FTDI-chip (logo Figuur 4.3). Dit IC implementeert het volledige USB-protocol. FTDI-chip levert freeware drivers die het mogelijk maken om de USB-verbinding aan te spreken als een virtuele COM-poort. Hierdoor is het niet nodig om speciale USB-drivers te ontwikkelen. Voor meer informatie over de drivers en de chip verwijzen we naar de website van FTDI [10].



Figuur 4.3: Logo FTDI-chip

De FT245 bevat een read- en writebuffer. De controle FPGA moet de readbuffer op regelmatige basis uitlezen om te vermijden dat er data- en/of controlesignalen verloren gaan. Indien de controle-FPGA controle- of datasignalen terug wil sturen naar de pc, worden deze in de writebuffer gezet. De FT245 neemt het zenden volgens het USB protocol voor zijn rekening. De pc software ontvangt de data, afkomstig van de FPGA, in hetzelfde formaat als bij een COM-poort.

Figuur 4.4 toont hoe het IC op het SASEBO-bord verbonden is. Deze figuur is afkomstig uit de handleiding van SASEBO-G [7].



Figuur 4.4: USB-functionaliteit op SASEBO-G

4.3 LabVIEW - De programmeertaal

Het automatiseren van de meetopstelling gebeurt met LabVIEW van National Instruments (NI). We geven in deze sectie een korte inleiding tot LabVIEW en bekijken de mogelijkheden van dit softwarepakket. Hiervoor hebben we een aantal bronnen geraadpleegd. Het boek van 'LabVIEW Graphical Programming' van Gary Johnson & Richard Jennings [8] is een introductie tot LabVIEW.

4.3.1 Wat is LabVIEW?

Het doel van National Instruments was een grafische programmeer omgeving te creëren waarmee automatisatie en instrumentatie van productieprocessen op een eenvoudige manier mogelijk werd. Hiervoor ontwikkelde National Instruments LabVIEW in 1986. Het motto luidde 'Je moet geen programmeur zijn om aan automatisatie te doen'.

National Instruments schreef de eerste LabVIEW versies voor de Macintosh computer. Macintosh was destijds het enige computersysteem dat een grafische gebruikersinterface had. Grafische programmacode sluit goed aan op de manier waarop ingenieurs denken, namelijk in (blok)schema's. LabVIEW heeft een uitgebreide bibliotheek met programmeerelementen. Deze manier van programmeren verschilt grondig met andere programmeertalen zoals C of Basic. Deze talen vormen een programma door gebruik te maken van tekst. Het logo van LabVIEW staat in Figuur 4.5 ContInterTssPCenSASEBO



Figuur 4.5: Het LabVIEW icoon

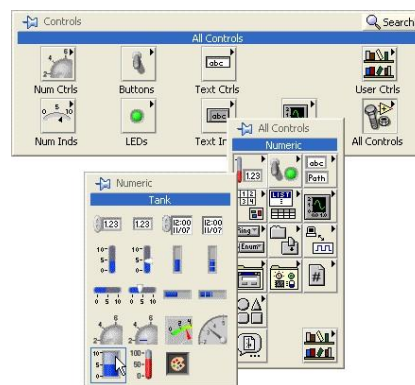
Het toepassingsdomein van LabVIEW is vrijwel onbeperkt. De eerste toepassing waarvoor gebruikers LabVIEW gebruikten was het automatiseren van metingen. LabVIEW moest een eenvoudige en goede communicatie tussen meetinstrumenten en een computer tot stand brengen. Hedendaagse toepassingen in industriële omgevingen die gegevens verzamelen, analyseren en visualiseren, gebruiken bijna altijd LabVIEW als programmeertaal.

4.3.2 Opbouw van een VI

LabVIEW programma's worden virtuele instrumenten (VI's) genoemd. Een VI bestaat uit drie onderdelen: een frontpaneel, een blokdiagram en een icoon.

Het frontpaneel

De eerste fase in de ontwikkeling van een VI is het ontwerpen van een frontpaneel of bedieningspaneel. De gebruiker vindt hier de nodige informatie om een bepaalde toepassing in werking te stellen. Dit grafische paneel bevat o.a. virtuele drukknoppen, uitlezingen, grafieken, enz.... LabVIEW noemt deze elementen 'controls en indicators'. Het 'controls en indicators' palet staat weergegeven in figuur 4.6.

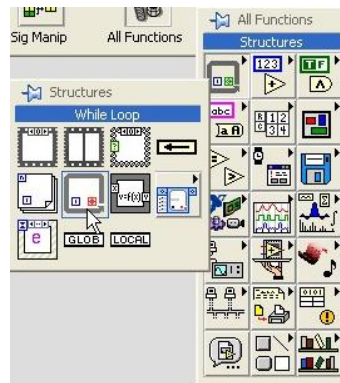


Figuur 4.6: Controls palette

Het frontpaneel is vergelijkbaar met het bedieningspaneel van een fysisch toestel. Afhankelijk van de ingegeven parameters zal het achterliggende programma een bepaalde actie uitvoeren. Het frontpaneelontwerp met LabVIEW is een vrij gemakkelijke taak. De gebruiker haalt de bedieningselementen en indicatoren rechtstreeks uit de bibliotheek. De bibliotheek die deze componenten bevat, noemt het control- en indicatorpalet.

Het blokdiagram

Het blokdiagram is de eigenlijke programmacode. Het blokdiagram bevat verwijzingen naar de bedieningselementen en indicatoren op het frontpaneel. Deze elementen vormen samen met functieblokken het programma. De bibliotheek die de functieblokken bevat, noemt het functiepalet. Figuur 4.7 toont dit functiepalet.



Figuur 4.7: Controls palette

Deze functieblokken bevatten wiskundige bewerkingen, functies voor data-acquisitie en analyse, dataopslag, netwerkcontrole, ...

De connector en het icoon

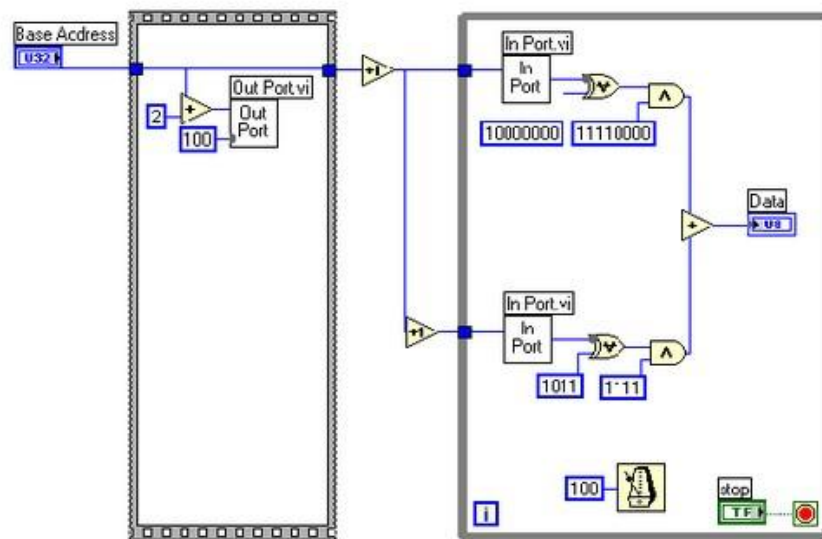
Elk functieblok heeft een connector en een icoon zoals figuur 4.8 toont. Het icoon geeft bondig de werking van het functieblok weer. De connector bevat één of meerdere terminals. Deze terminals vormen de aansluitpunten voor het functieblok. Draden verbinden terminals van verschillende functieblokken met elkaar om zo een werkend geheel te vormen.



Figuur 4.8: Icoon - connectorblok

4.3.3 Parallele poort in LabVIEW

Om de programmeerwijze in LabVIEW te verduidelijken gaan we het voorbeeld uit Figuur 4.9 uitleggen. Dit voorbeeld leest gegevens in van de parallele poort en is nuttig omdat we deze functionaliteiten tijdens de ontwikkeling van de meetopstelling gebruikte. Dit voorbeeld is geen zelfgeschreven voorbeeld, maar een van de National Instruments website [9].



Figuur 4.9: Voorbeeld parallele input, geschreven door National Instruments

Dit voorbeeld maakt gebruik van de controle- en statusregisters van de parallele poort om aan input te doen. Eerst is er een initialisatie fase, gevolgd door een lus. Het programma blijft deze lus herhalen totdat de gebruiker het programma afsluit.

Tijdens de initialisatie fase worden de lijnen, die verbonden zijn met het statusregister, hoog gemaakt. De reden hiervoor is verklaard in 4.5

In de lus gaat het programma voortdurend het controle- en statusregister uitlezen en weergeven op het frontpaneel. Het valt op dat er allerlei bewerkingen nodig zijn om de bit's op een juiste manier weer te geven. Verder zijn er een aantal lijnen die geïnverteerd worden om een juiste weergave van de ingangssignalen te verkrijgen.

De ontwikkelaar koos ervoor om de processorbelasting te beperken door slechts om de 100 ms de ingangen te controleren. Dit wordt aangegeven met het klokje binnen de lus en is zeer belangrijk wanneer er meerdere programma's/processen gelijktijdig draaien op de computer.

De 'in port'- en 'out port'-VI's zijn ontworpen door National Instruments of door de LabVIEW community. Deze VI's kunnen rechtstreeks lezen/schrijven van/naar bepaalde registers. Door gebruik te maken van deze VI's omzeilt de ontwikkelaar het communicatieprotocol van de parallele poort en zijn er geen specifieke drivers nodig.

4.4 MSO6052A - De oscilloscoop

4.4.1 De Agilent MSO6052A

Als nauwkeurig meetinstrument gebruiken we een Agilent MSO6052A oscilloscoop, afgebeeld in Figuur 4.10. Deze oscilloscoop heeft een samplerate van 500 MHz en beschikt over twee analoge kanalen of 15 digitale kanalen. De pc kan deze oscilloscoop besturen en uitlezen. De MSO6052A heeft een USB-, GPIB- en RS232-verbinding. Verder is bij de MSO6052A software meegeleverd die tools bevat om de oscilloscoop via LabVIEW te besturen. De startersguide en manual van deze oscilloscoop zijn terug te vinden op de website van Agilent [11].



Figuur 4.10: Agilent MSO6052A oscilloscoop

Het GPIB-protocol verzorgt de communicatie tussen de pc en de oscilloscoop. Veel geautomatiseerde meetopstellingen gebruiken dit protocol voor de interactie tussen pc en meetapparatuur. 4.4.2, 4.4.3 en 4.4.4 bespreken daarom het GPIB-protocol met bijhorende SCPI-programmeertaal en de programeerstructuur.

4.4.2 GPIB - Interface tussen meettoestellen

In deze sectie hebben we gebruik gemaakt van een aantal elektronische bronnen. Algemene informatie over het GPIB protocol is terug te vinden bij [12], [13] en [14]. Voor gedetailleerde informatie verwijzen we naar de IEEE 488.1 standaard.

Geschiedenis

De eerste bus tussen meettoestellen werd ontwikkeld in 1965 door Hewlett-Packard (HP). HP gebruikte deze bus om hun instrumenten op een gemakkelijke manier met elkaar te verbinden en te controleren met een pc. Hewlett-Packard noemde deze bus HP-IB.

Met de introductie van de digitale controllers en programmeerbare instrumenten steeg de vraag naar een standaard om te communiceren tussen instrumenten en controllers. Deze keer niet alleen tussen HP toestellen maar ook tussen toestellen van andere fabrikanten. Er werd een bus ontwikkeld die een groot succes kende, vooral omwille van zijn hoge overdrachtsnelheid en betrouwbaarheid. Deze bus is in 1975 door het IEEE in een standaard specificatie gegoten en tot de IEEE488.1 standaard gedoopt. Deze bus is ondertussen wereldwijd verspreid geraakt en is bekend onder volgende namen:

- General Purpose Interface Bus (GPIB);
- Hewlett-Packard Interface Bus (HP-IB);
- IEEE 488 Bus;
- IEEE 488.1.

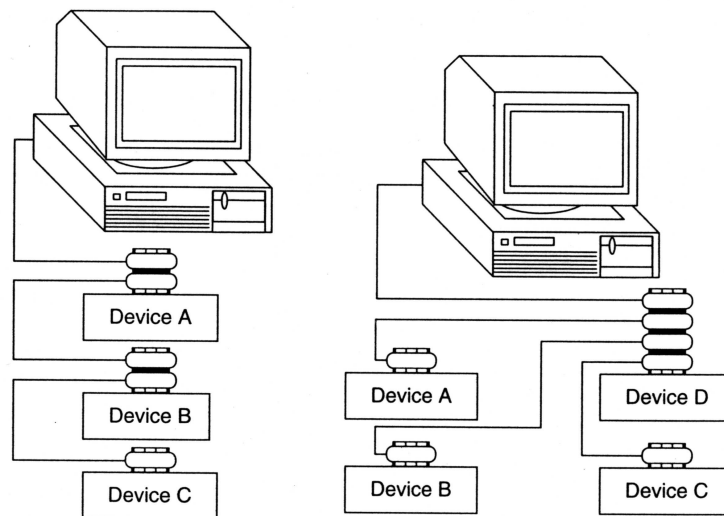
Eigenschappen

Fysieke opbouw van de bus

De GPIB-bus gebruikt negatieve logica, met standaard TTL-niveaus. Wanneer een bepaalde bit waar is, komt dit overeen met een laag TTL-niveau ($\leq 0.8V$) en wanneer een bit niet waar is, met een hoog TTL-niveau ($\geq 2.0V$).

De onderlinge verbinding tussen de toestellen gebeurt met een afgeschermd kabel van 24 aders. Aan de uiteinden van de kabel bevindt zich een 24-polige Amphenol of AMP connector. Deze connector is te vergelijken met een connector van een printerkabel aan de zijde van de printer, maar dan 24-polige uitgevoerd.

We kunnen de GPIB-compatibele toestellen op twee mogelijke manieren met elkaar linken. Een eerste methode is een lineaire configuratie. Bij een lineaire configuratie zijn de apparaten achter elkaar geplaatst waarbij de controller het eerste apparaat is. De tweede mogelijkheid is een sterconfiguratie. Bij een sterconfiguratie staat de controller in het midden van de opstelling. Alle andere apparaten hebben een individuele verbinding met deze controller. Figuur 4.11 toont de mogelijke configuraties van de GPIB-bus.



Figuur 4.11: Mogelijke GPIB-busconfiguraties. Links, de lineaire configuratie. Rechts, de sterconfiguratie

Communicatie tussen de deelnemers

De GPIB-interface bestaat uit zestien communicatielijnen en acht massalijnen (samen dus 24 lijnen). Van deze zestien lijnen, zijn er acht bi-directionele datalijnen, drie handshakelijnen en vijf interfacemanagementlijnen.

Een GPIB-toestel stuurt zijn data parallel naar de ontvanger via acht datalijnen. Naast het dataverkeer vloeien ook de commando's over deze acht datalijnen. De bus heeft een bandbreedte van 1Mbyte/s. Deze bandbreedte werd later uitgebreid tot 8Mbyte/s.

De drie asynchrone handshakelijnen (NRFD, NDAC en DAV) geven informatie over de data op de bus. Deze lijnen vervullen twee belangrijke functies. Ten eerste maakt een GPIB-toestel met behulp van de handshakelijnen een onderscheid tussen data en commando's. Ten tweede bevatten de handshakelijnen informatie over de juistheid van de data.

De vijf interfacemanagementlijnen (ATN, EOI, IFC, REN, SRQ) leiden de vloed aan controle- en databytes over de interface in goede banen.

Vereiste configuratie

De GPIB-interface bereikt zijn hoge transferrate enkel indien de gebruiker zich aan de standaard houdt. Hieronder staan de beperkingen die de standaard oplegt:

- Een maximum afstand van 4m tussen 2 toestellen;
- Een maximale kabellengte van 20m;
- Maximum 15 toestellen aangesloten op de bus, minimum 2/3 van de toestellen moeten aan staan.

Begrippen binnen GPIB

Boodschappen - berichten

GPIB toestellen communiceren met andere toestellen door het versturen van boodschappen. Er zijn twee type datastromen terug te vinden op de bus. Ten eerste is er de gewone data, zoals meetresultaten en instellingen van een meettoestel. Ten tweede is er de interface- of commandodata. Dit zijn boodschappen die het toestel besturen, zoals bijvoorbeeld adresseringsdata.

Talkers, listeners en controllers

Een GPIB-apparaat kan een talker, listener, of controller zijn. Een talker stuurt data naar een of meerdere listeners. Controllers regelen het data- en instructieverkeer op de bus. Er kunnen meerdere controllers op de bus aanwezig zijn. Indien dit het geval is, is er slechts één actief. In onze meetopstelling is de pc een controller/listener en de oscilloscoop een listener/talker.

'PCIe-GPIB+' kaart

De pc waarop het meetprogramma draait heeft een GPIB-interfacekaart nodig. Om compatibiliteitsredenen met LabVIEW kozen we voor een GPIB-interfacekaart van NI.

NI heeft PCI-GPIB-kaarten en USB-GPIB-convertors. National Instruments raadt aan dat een toepassing waar een hoge datatroughput nodig is, een PCI-GPIB-kaarten gebruikt. NI stelt hierin twee types voor; de 'NI-PCIe GPIB+' kaart en de 'NI-PCIe GPIB' kaart.

Onze voorkeur ging naar de 'NI-PCIe GPIB+' kaart, afgebeeld in Figuur 4.12. Deze kaart is net iets duurder dan de 'NI-PCIe GPIB' kaart, maar omdat de debugmogelijkheden van deze kaart veel groter zijn dan die van de andere, is dit de investering zeker waard.



Figuur 4.12: NI PCIe-GPIB+

4.4.3 SCPI - Commando's voor meettoestellen

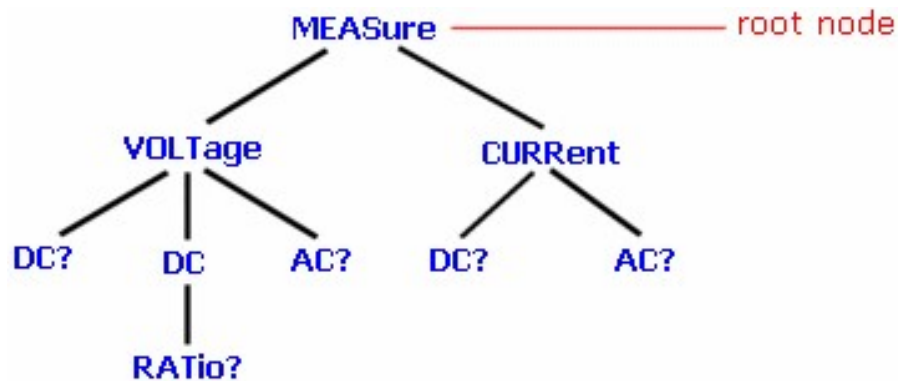
Geschiedenis

Op 23 april 1990 maakte een groep van toestelfabrikanten de SCPI-specificaties [30] bekend. Deze specificaties definieerde een algemene bevelenreeks voor programmeerbare toestellen. Vóór de SCPI-bevelenreeks, ook wel uitgesproken als 'skippy', ontwikkelde elke fabrikant zijn eigen reeks van bevelen voor zijn programmeerbare toestellen. Dit gebrek aan standaardisering dwong de ontwikkelaars van meetsystemen om een hele reeks bevelen en specifieke toestelparameters te leren.

Dit alles leidde tot problemen bij het programmeren en resulteerde in onvoorspelbare ontwikkelingskosten en vertragingen in het werkschema. Door het definiëren van een gestandaardiseerde bevelen-reeks voor het programmeren van de toestellen, vermindert de ontwikkelingstijd en vergroot de leesbaarheid van testprogramma's. Het werd ook mogelijk om toestellen van verschillende fabrikanten onderling te wisselen. SCPI is een standaard die de softwarematige programmeerbevelen voor toestellen samenbundelt. De eerste versie van de standaard werd uitgebracht in het midden van 1990.

Voorbeeld van de SCPI taal

Uit een tutorial over SCPI [15] haalde we dat SCPI commando's een boomstructuur volgen zoals weergegeven in Figuur 4.13. Gerelateerde commando's zijn gegroepeerd in de takken van een boom. Elk sleutelwoord in het commando is een knooppunt van deze boom. De ontwikkelaars van SCPI noemen het eerste knooppunt van de boom de 'root node'.



Figuur 4.13: SCPI commando structuur uit [15]

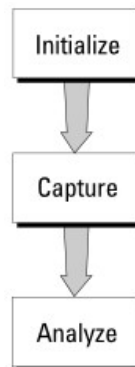
Figuur 4.13 geeft een stuk van de SCPI-boom weer. Deze structuur bevat commando's om een digitale multimeter te besturen. De controller vormt een commando door neer te dalen in de boom en tussen elk knooppunt een dubbelpunt te plaatsen. Daarna verzendt de controller dit commando naar de digitale multimeter. De digitale multimeter bouwt de boom terug op. Achter het diepst liggende knooppunt zit de actie die uitgevoerd moet worden. Voorbeelden uit deze boom zijn:

- MEASure:VOLTage:DC?,
- MEASure:VOLTage:DC:RATio?,
- MEASure:VOLTage:AC?.

Opmerking: de hoofdletters geven de korte vorm van de sleutelwoorden. De kleine letters in combinatie met de hoofdletters geven de lange vorm van de sleutelwoorden. Het is mogelijk om beide notaties te gebruiken. Het laatste voorbeeld wordt dan bevoorbeeld: MEAS:VOLT:AC?.

4.4.4 Programmeerstructuur

Figuur 4.14 laat de basisstructuur voor een oscilloscoop programma zien. Elk programma bestaat uit drie stappen. Eerst gebeurt een initialisatie van de oscilloscoop, daarna zal de meting plaatsvinden. Na de meting gebeurt een analyse van de opgemeten data.



Figuur 4.14: Basisstructuur van het oscilloscoop programma van [11]

De initialisatiefase

Om een goed werkend programma met een hoge prestatie te verkrijgen, moet het programma, de controller en de oscilloscoop starten in een gekende staat. Zonder een correcte initialisatie kan het programma soms correct uitgevoerd worden, maar soms ook niet. Zonder initialisatie behoudt het programma de instellingen van de vorige keer.

Een initialisatie bestaat uit drie delen. Een programma initialisatie, een controllerinitialisatie en een oscilloscoopinitialisatie.

- **Programma initialisatie:** De programma-initialisatie gebeurt als eerste. Men definieert en initialiseert variabelen, maakt geheugen vrij en test de systeemconfiguratie van het computersysteem.
- **Controller initialisatie:** De controllerinitialisatie gebeurt als tweede. Deze initialisatie zorgt ervoor dat de interface die communiceert met de oscilloscoop goed is ingesteld en klaar is voor de data-overdracht.
- **Oscilloscoop initialisatie:** De oscilloscoopinitialisatie is de laatste stap in het initialisatieproces. Deze stap stelt de oscilloscoop correct in.

De meetfase

Eens dat de oscilloscoop geïntialiseerd is, start de meting. Tijdens deze meting slaat de oscilloscoop de meetdata op in zijn geheugen.

Om meting te starten stuurt de pc het `:DIGitize` commando naar de oscilloscoop. Dit commando maakt de databuffers leeg en start een meting. De meting gaat door tot het 'acquisitie' geheugen van de oscilloscoop vol is. Daarna wacht de oscilloscoop op een commando dat aangeeft dat de controller klaar is om de data over te brengen naar de pc. Commando's die tijdens het uitvoeren van een meting binnenkomen worden opgeslagen in een buffer. De oscilloscoop voert deze commando's pas uit na het voltooien van de meting.

De analysefase

Nadat de oscilloscoop de meting heeft afgerond, kunnen we de data analyseren. Dit kan op twee manieren: ofwel op de oscilloscoop zelf ofwel door de data over te brengen naar de controller om dan via een het programma een grondigere analyse uit te voeren. In de oscilloscoop zitten verschillende analysetools ingebouwd. Zo hebben we een frequentieanalyse, een dutycyclometing, een periodometing, een pulsbreedtemeting

Als de oscilloscoop een van de `:WAVEform` commando's ontvangt, brengt deze de data over via de controller naar de pc. Hierdoor is het mogelijk om de data weer te geven op de pc en te vergelijken met een gekende meting of logische patronen te herkennen.

4.4.5 Nuttige commando's

Om de oscilloscoop in te stellen en data uit te lezen maken we gebruik van SCPI-commando's. Er bestaan een hele boel SCPI-commando's. Het is mogelijk dat verschillende commando's tot hetzelfde resultaat leiden. In dit deel gaan we een aantal commando's beschrijven die volgens ons het best geschikt zijn voor onze meetopstelling.

Het `:DIGitize` commando

Het `:DIGitize` commando is een van de rootcommando's. Dit commando start slechts één acquisitie volgens de instellingen gedaan door de `:ACQUIRE`-commando's. Wanneer de acquisitie gedaan is, stopt de oscilloscoop de meting.

Als er geen argumenten volgen na het `:DIGitize`-commando, doet de oscilloscoop een acquisitie op alle zichtbare kanalen. Als het `:DIGitize`-commando een kanaal als argument heeft, doet de oscilloscoop enkel een acquisitie op dit kanaal.

De :ACQuire commandoset

De pc stelt met :ACQuire-commando's in hoe de acquisities moeten gebeuren. Er zijn verschillende opties en instellingen mogelijk met deze commando's.

Een goede instelling met de :ACQuire-commando's is noodzakelijk voor een goede werking van het :DIGitize-commando.

De :WAVeform commandoset

:WAVeform-commando's zijn een reeks van commando's die dienen om data over te brengen van de oscilloscoop naar de controller.

De opgemeten data bestaat uit twee delen: de 'preamble' en de 'waveform' data. Er bestaan twee aparte commando's om de 'preamble' en de 'waveform' uit te lezen. We hebben het :WAVeform:DATA commando en het :WAVeform:PREAmble commando. De 'waveform' data is de eigenlijke waarde van elk opgenomen punt. De 'preamble' geeft informatie over hoe de ontvanger 'waveform' data moet interpreteren. De 'preamble' bevat onder andere het aantal opgenomen punten, het formaat van de opgenomen data en het type van de opgenomen data. Verder bevat hij ook informatie over het X en Y interval en de referentie. De ontvanger vertaalt de datapunten met behulp van de 'preamble' naar tijd- en spanningsinformatie.

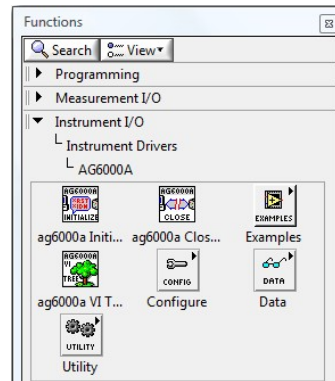
4.4.6 Interactie tussen LabVIEW en MSO6052A

De handleiding van de oscilloscoop op de website van Agilent [11] geeft aan dat LabVIEW de MSO6052A ondersteunt. Dit wil zeggen dat ergens een bibliotheek te vinden is met een aantal subVI's die de communicatie met de scoop gemakkelijker maken. LabVIEW noemt deze subVI's 'instrument drivers'.

Bij het opstarten van LabVIEW verschijnt er een 'Getting Started' venster. Het installeren van de 'instrument drivers' gebeurt door in het menu 'tools' te klikken op 'Instrumentation' en daarna op 'Find instrument driver'. LabVIEW toont daarna een scherm waar alle instrumenten staan die hij ondersteunt.

Na het selecteren van de fabrikant en het invullen van het typenummer wordt er een bibliotheek gevonden. De 'AG6000A' bibliotheek ondersteunt een ruim gamma aan oscilloscopen van Agilent, waaronder ook de MSO6052A. Door op 'install' te klikken,

installeert LabVIEW de desbetreffende bibliotheek en komen de subVI's op het functiepalet te staan. Het functiepalet voor de MS060xxA is weergegeven in Figuur 4.15.



Figuur 4.15: Functie pallet voor Agilent MSO60xxA-type oscilloscoop

De AG6000A initialize VI

De AG6000A initialize VI opent de connectie met het instrument. Om connectie met een instrument te maken moet de gebruiker de naam van het instrument te kennen. Deze naam wordt altijd in SCPI-taal geformuleerd en bevat onder andere het adres van het instrument.

Voorbeeld: GPIB::22::INSTR

Op deze manier weet LabVIEW dat hij een verbinding maakt met het instrument op adres 22 via de GPIB-interface.

Verder zijn er ook nog twee opties voorzien:

- De gebruiker kan opgeven of LabVIEW na het openen van de connectie moet nagegaan of het instrumenttype juist is. Indien de gebruiker deze optie selecteert, zal LabVIEW nagaan of het instrument met het opgegeven adres daadwerkelijk een instrument is uit de Agilent 60xxA reeks. Indien dit niet het geval is, zal de gebruiker een error krijgen en kan hij/zij een gepaste actie ondernemen.
- Als tweede optie laat toe het instrument te resetten na het openen van de connectie. Na de reset bevindt de oscilloscoop zich in een gekende 'state' die beschreven staat in de handleiding.

Indien er een fout optreedt, verbreekt LabVIEW de verbinding met het instrument.

De AG6000A close VI

Elke verbinding met de een GPIB apparaat moet beëindigd zijn voordat het programma afsluit. De 'AG6000A close' VI neemt dit voor zijn rekening. Voordat deze VI de verbinding afsluit, zal hij eerst controleren of er ergens in de communicatie een fout opgetreden is.

De AG6000A config subbibliotheek

In de AG6000A bibliotheek zit ook een subbibliotheek met subVI's om de oscilloscoop te configureren. Deze subVI's zijn afgebeeld in Figuur 4.16 en doen de instellingen die de gebruiker ook op het frontpaneel kan doen. Enkele voorbeelden zijn: acquisitie type, kanaalkeuze, kanaallabels, ... Een gedetailleerde uitleg van deze commando's is terug te vinden in de gebruikershandleiding van de Agilent MSO6052A-oscilloscoop of in de context-hulp van LabVIEW.



Figuur 4.16: Configuratie subVI's voor de AG60XXA serie

De AG6000A utility's subbibliotheek

De AG6000A-utility's subbibliotheek bevat subVI's die informatie geven over de oscilloscoop. Deze subVI's zijn afgebeeld in Figuur 4.17. Met deze bibliotheek is het mogelijk om een self-test uit te voeren en de resultaten te bekijken. De belangrijkste functie in deze bibliotheek is de functie die de oscilloscoop volledig reset.



Figuur 4.17: Utility subVI's voor de AG60XXA serie

De AG6000A data subbibliotheek

Figuur 4.18 toont de AG6000A data subbibliotheek. Deze subbibliotheek bevat allerlei subVI's om data van de oscilloscoop in te lezen. Deze subVI's zijn het belangrijkste voor onze toepassing. Als de gebruiker de oscilloscoop goed heeft ingesteld, kan LabVIEW de data met deze subVI's op een eenvoudige manier binnenhalen.



Figuur 4.18: Data subVI's voor de AG60XXA serie

4.4.7 De praktijk - eerste tests

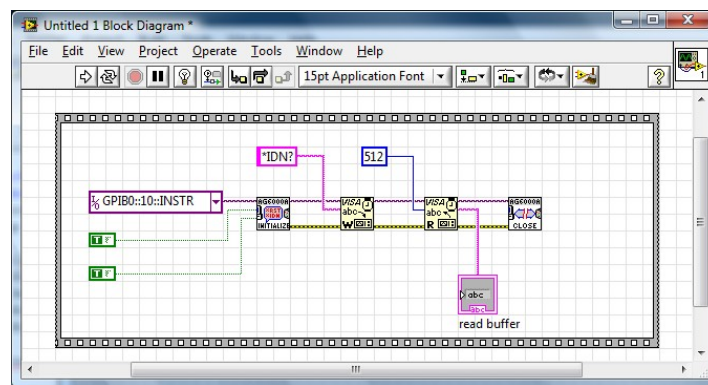
Adresseren van de oscilloscoop

Om de MSO6052A te adresseren sluiten we deze eerst aan op de pc. Daarna kennen we een adres toe aan deze oscilloscoop. Deze toekenning gebeurt op onderstaande manier via zijn bedieningspaneel.

- Sluit de oscilloscoop aan op de pc via de GPIB-kabel.
- Druk op de 'utility' knop die zich bevindt op het frontpaneel van de oscilloscoop

- Kies voor de optie I/O en zorg dat GPIB bij 'controller' en 'configure' geselecteerd is.
- Kies nu het GPIB-adres met de 'entry' knop op de oscilloscoop. Dit adres mag liggen tussen 1 en 30. Wij kiezen voor adres 10.
- Start LabVIEW en ontwikkel het programma

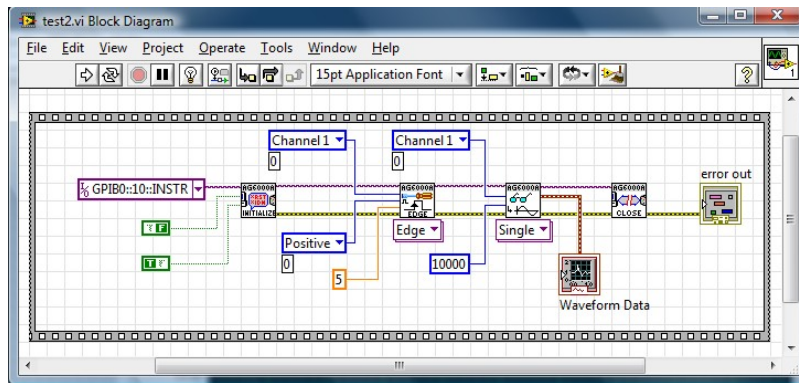
Figuur 4.19 toont een voorbeeldprogramma dat een connectie met de oscilloscoop maakt, deze reset en zijn ID opvraagt. Daarna toont LabVIEW dit ID op het frontpaneel. Als de gebruiker het programma afsluit, verbreekt LabVIEW eerst de connectie met de oscilloscoop en sluit daarna het programma af.



Figuur 4.19: Testprogramma voor connectie te maken

Uitlezen van oscilloscoop data

Als de gebruiker de adresseringsinstellingen van hierboven goed deed, kan LabVIEW data uitlezen van de oscilloscoop. Om dit te testen hebben we een functiegenerator aangesloten op de oscilloscoop en een klein testprogramma geschreven. Figuur 4.20 toont het testprogramma. Dit programma opent een connectie met de oscilloscoop, doet een aantal kanaalinstellingen en start één meting op kanaal 1. Als de meting gedaan is, haalt het programma de oscilloscoopdata binnen en geeft de datapunten weer in een grafiek op het frontpaneel.



Figuur 4.20: Testprogramma om data uit te lezen

4.5 Controle interface tussen pc en SASEBO

4.5.1 Simpele digitale I/O kaart

Zoals beschreven in 4.1, moet de pc controlesignalen geven aan de FPGA. Hiervoor hebben we een soort van communicatie interface nodig. Deze interface kan bestaan uit een simpele I/O-kaart in de pc.

National Instruments produceert zulke I/O-kaarten. Voor onze toepassing zijn deze kaarten vrij duur. Als alternatief kozen we voor een parallele I/O-interface. In een later stadium bleek echter dat deze niet meer voldeed. Daarom hebben we deze vervangen door een USB interface.

We staken enkele weken tijd in het ontwerp van deze I/O-kaart en veel van de testanalyses zijn ermee uitgevoerd. Vandaar is het zeker de moeite om de werking ervan kort toe te lichten.

D-sub	Signal	Function	Source	Register	Register Bit #	Inverted at Connector?	Pin: Centronics
1	nStrobe	Strobe D0-D7	PC ¹	Control	0	Yes	1
2	D0	Data Bit 0	PC ²	Data	0	No	2
3	D1	Data Bit 1	PC ²	Data	1	No	3
4	D2	Data Bit 2	PC ²	Data	2	No	4
5	D3	Data Bit 3	PC ²	Data	3	No	5
6	D4	Data Bit 4	PC ²	Data	4	No	6
7	D5	Data Bit 5	PC ²	Data	5	No	7
8	D6	Data Bit 6	PC ²	Data	6	No	8
9	D7	Data Bit 7	PC ²	Data	7	No	9
10	nAck	Acknowledge	Printer	Status	6	No	10
11	Busy	Printer busy	Printer	Status	7	Yes	11
12	PaperEnd	Paper end, Empty	Printer	Status	5	No	12
13	Select	Printer selected (online)	Printer	Status	4	No	13
14	nAutoLF	Generate automatic line feeds	PC ¹	Control	1	Yes	14
15	nError (nFault)	Error	Printer	Status	3	No	32
16	nInit	Initialize Printer	PC ¹	Control	2	No	31
17	nSelectIN	Select printer (Place online)	PC ¹	Control	3	Yes	35
18	Gnd	Ground Return for nStrobe, D0					19,20
19	Gnd	Ground Return for D1, D2					21,22
20	Gnd	Ground Return for D3, D4					23,24
21	Gnd	Ground Return for D5, D6					25,26
22	Gnd	Ground Return for D7, nAck					27,28
23	Gnd	Ground Return for nSelectIn					33
24	Gnd	Ground Return for Busy					29
25	Gnd	Ground Return for Init					30
	Chassis	Chassis Ground					17
	NC	No connection					15,18,34
	NC	Signal Ground					16
	NC	+V5	Printer				35

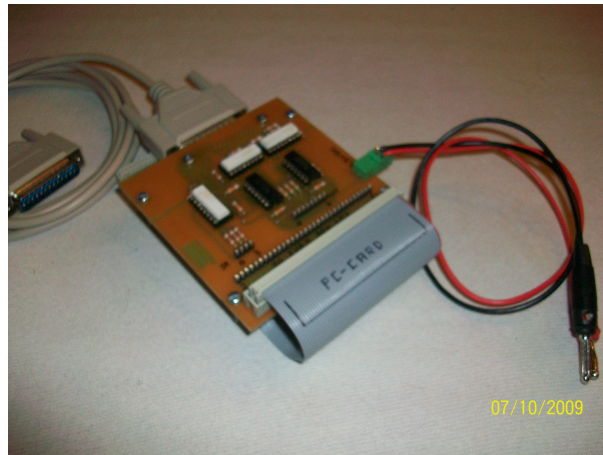
¹Setting this bit high allows it to be used as an input ²Some Data ports are bidirectional

Table 1. Pin Functionality Diagram.

Figuur 4.21: Beschrijving van de parallele poort

Zoals Figuur 4.21 weergeeft, heeft de parallele-poort lijnen die verbonden zijn met een statusregister, controleregister en dataregister. Het dataregister is enkel bedoeld voor output en het statusregister enkel voor input. Het controleregister is oorspronkelijk bedoeld voor output, maar kan met enkele simpele hardware aanpassingen ook gebruik worden als input.

Ontwerp van de I/O interface

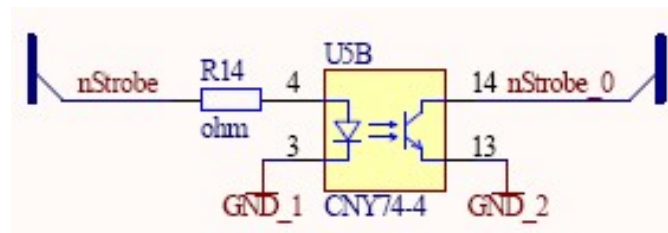


Figuur 4.22: De gerealiseerde I/O interface

Voor het ontwerp van de kaart hebben we gekozen voor acht uitgangen en vier ingangen. Dit was ruim voldoende voor de toepassing die wij voor ogen hadden. Voor de uitgangen gebruikten we de acht lijnen die verbonden zijn met het dataregister. Voor de inputlijnen dachten we de lijnen van het controleregister te gebruiken, maar dit veroorzaakte problemen.

Omwille van de veiligheid kozen we voor een volledig gescheiden circuit tussen de pc en de FPGA. We gebruikten optocouplers voor een volledig galvanische scheiding te bekomen. Voor de uitgangen was dit geen probleem, we kunnen met de 5V output van de parallele-poort de optocouplers aansturen. De ingangen van de parallelepoort zien 5V als een hoog niveau. Dit zorgde voor een probleem omdat de voedingsspanning niet aanwezig is op de connector van de parallele-poort. De enige oplossing was een externe voeding van 5V plaatsen. Een tweede voeding was geen optie, omdat we al een 3,3V voeding gebruikten om het FPGA-bord te voeden.

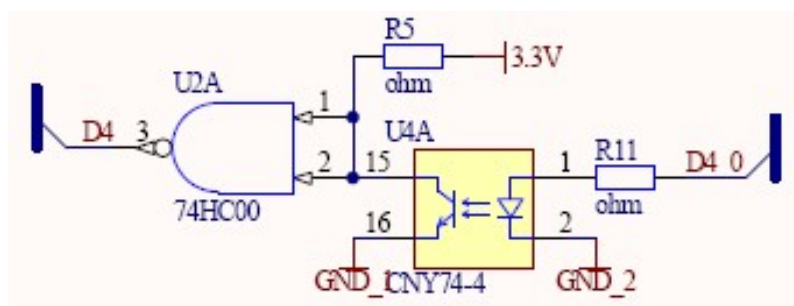
Om het gebruik van een tweede externe voeding te vermijden, hebben we de lijnen van het statusregister bekeken. Hieruit bleek dat de uitgangen als ingang gebruikt kunnen worden door ze allemaal hoog te zetten met de pc en ze extern laag te trekken. De transistor in de optocoupler kan deze taak op zich nemen. Op deze manier is er geen externe voeding nodig om de ingangen aan te sturen.



Figuur 4.23: Input gedeelte van de I/O kaart

Omdat de initlijn van het statusregister in de connector niet geïnverteerd is, is de optocouplerschakeling anders opgebouwd dan bij de andere lijnen. Bij de uitgangen is nog een NAND-poort geplaatst die werkt als invertor. Dit is gedaan omwille van het inverterend gedrag van de optocouplers. De schakeling van het ingangsgedeelte is weergegeven in Figuur 4.23. Figuur 4.24 toont de schakeling van het uitgangsgedeelte

Voor de precieze schakeling verwijzen we naar de PDF-documenten op de cd.



Figuur 4.24: Output gedeelte van de I/O kaart

4.5.2 USB

De parallele I/O-interface werkte goed voor simpele toepassingen, maar beperkt de meetopstelling enigszins. De FPGA moet tijdens elke meetcyclus het cryptografisch algoritme met een andere klaartekst uitvoeren om een geslaagde vermogenaanval te doen.

Omdat met de parallele interface geen data-overdracht mogelijk is, moet de klaartekst tijdens het downloaden van het algoritme in de FPGA bekend zijn. De implementatiesoftware moet deze klaartekst in een RAM-geheugen van de FPGA zetten. Tijdens het uitvoeren van de metingen moet de FPGA voor elke meetcyclus een nieuwe klaartekst uit dit RAM-geheugen halen.

De klaartekst in een RAM-geheugen plaatsen heeft twee grote nadelen. Ten eerste moet de FPGA extra logica bevatten om het RAM-geheugen uit te lezen en om bij te houden wat de volgende klaartekst is. Omdat dit voor extra vermogenverbruik zorgt, zal er meer ruis op het powersignaal komen. Ten tweede is het RAM-geheugen in grootte beperkt. Dit zorgt er rechtstreeks voor dat ook het aantal metingen beperkt is.

Omwille van deze nadelen kozen we ervoor om de parallele I/O interface te vervangen door een USB-interface. Deze interface verzorgt de synchronisatie tussen pc en SASEBO. We versturen zowel data als controlesignalen over deze interface. Voordat elke meetcyclus start, zendt de pc nieuwe klaartekst naar de FPGA. Daarna geeft hij een startsignaal dat de cryptografische bewerkingen start met de nieuwe klaartekst. Als de cryptografische bewerkingen gedaan zijn, meldt de FPGA dit aan de pc.

Communicatieprotocol

De computer ziet de USB-verbinding met het SASEBO-bord als een virtuele COM-poort. Het gevolg hiervan is dat we enkel 8-bit ASCII-karakters kunnen doorsturen. Omdat de data uit groepen van 8-bits bestaat, kunnen we geen tekens uit de ASCII-karakterset reserveren om de commando's weer te geven. We moeten dus een soort van communicatieprotocol gebruiken.

Voor onze toepassing zijn de meeste bestaande communicatieprotocols te ingewikkeld. Daarom hebben we zelf een klein communicatieprotocol bedacht. Figuur 4.25 stelt schematisch dit communicatieprotocol voor

Een nieuwe meetcyclus start als de FPGA een 's' ontvangt. Daarna volgen de karakters die overeen komen met de klaartekst. Wanneer de FPGA de volledige klaartekst ontvangen heeft, zal deze de cryptografische bewerkingen starten. Wanneer deze bewerkingen gedaan zijn, zendt de FPGA een 'd' naar de pc. De pc detecteert dit en start indien nodig een volgende cyclus.

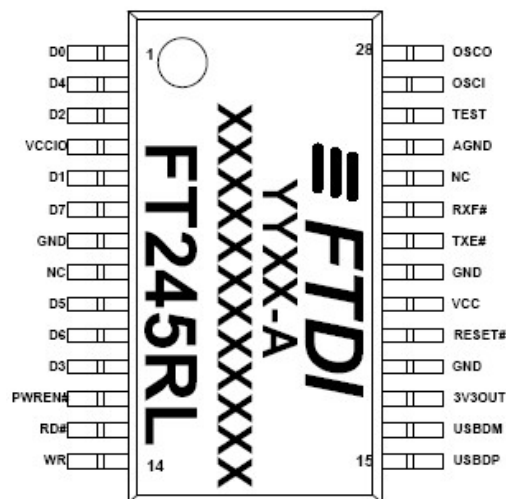


Figuur 4.25: Communicatieprotocol tussen SASEBO en pc

Communicatie met FT245R

Zoals in 4.2.3 vermeld, gebruikt SASEBO-G een FT245R om USB-communicatie mogelijk te maken. De FT245R bezit een read- en writebuffer. Als de pc data (of commando's) verzendt, zullen deze in de readbuffer van de FT245R komen te staan. Indien de FPGA iets wil verzenden naar de pc, zal deze dit in de writebuffer zetten.

De FPGA leest/schrijft de readbuffer/writebuffer via een parallele bus. Hiervoor zijn extern acht bidirectionele pinnen voorzien op de FT245R. De FPGA maakt een keuze tussen lezen en schrijven door bepaalde controle lijnen hoog of laag te maken. In Figuur 4.26 staat de pin-layout van de FT245R.

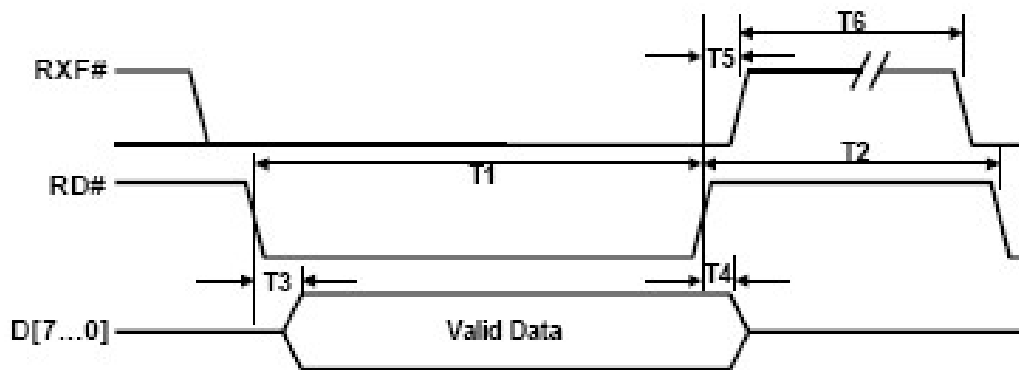


Figuur 4.26: Pin-layout van de FT245R uit [10]

Figuur 4.27 geeft de timingspecificaties weer voor het uitlezen van de readbuffer. Om de readbuffer uit te lezen, maken we gebruik van twee controlelijnen. Het IC maakt de RXF-lijn laag wanneer er geldige data in de readbuffer zit. Indien de readbuffer leeg is of de data op de bi-directionele bus niet geldig is, zal deze lijn hoog zijn.

Indien de FPGA de readbuffer van het IC wil uitlezen, moet deze de RD-lijn laag maken. Deze lijn moet minstens 50 ns laag blijven. Na 20 tot 50 ns verschijnt er geldige data op de datalijnen van de FT245R.

Nadat de FPGA de RD-lijn hoog maakt, zal de RXF-lijn gedurende 80 ns hoog blijven. Dit wil zeggen dat nieuwe data pas beschikbaar is na 80 ns.

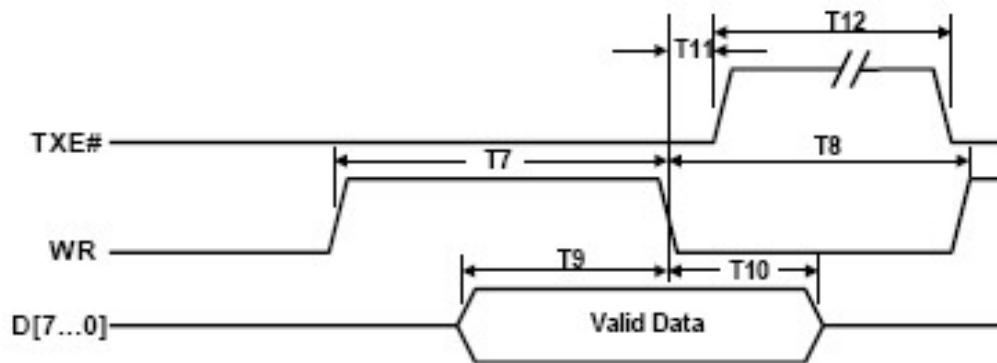


Figuur 4.27: Readcyclus van de FT245R uit [10]

Figuur 4.28 geeft de timingspecificaties weer voor het schrijven van de writebuffer. De FT245R bestuurt de TXE-lijn. Indien deze lijn hoog is, mag de FPGA niet schrijven naar de writebuffer. Dit kan het geval zijn wanneer de writebuffer vol geraakt. Als de lijn laag is, mag de FPGA een writecyclus starten.

Wanneer de FPGA wil schrijven, maakt deze de WR-lijn hoog en zet de FPGA data klaar op de datalijnen. De WR-lijn moet minstens 50 ns hoog blijven. De FT245R leest de data die zich op de datalijnen bevindt in op de dalende flank van de WR-lijn. Deze data schrijft hij daarna naar de writebuffer.

Na een writecyclus zal de TXE-lijn gedurende 80 ns hoog blijven. Dit wil zeggen dat de FPGA 80 ns moet wachten om nieuwe data te schrijven naar de FT245R.



Figuur 4.28: Writecyclus van de FT245R uit [10]

Hoofdstuk 5

Vermogenanalyse meetopstelling - Eindresultaat

5.1 pc-software

Om een vermogenaanval te doen, is een vermogenmeetopstelling nodig. Zoals in 4.2.3 aangehaald, moet de meetopstelling stabiel, gebruiksvriendelijk, snel en nauwkeurig zijn. Het meetprogramma dat de meetopstelling bestuurt, moet op een zelfstandige manier een aantal metingen achter elkaar uitvoeren. Na elke meting slaat het meetprogramma de meetdata op in afzonderlijke bestanden. Later gebruikt de gebruiker deze bestanden om de sleutel van het geanalyseerde systeem te achterhalen. Indien dit lukt, gaat hij/zij het systeem verbeteren en de metingen opnieuw uitvoeren.

Hoofdstuk 4 gaf informatie over hoe de meetopstelling hardwarematig opgebouwd is. Dit hoofdstuk gaat iets dieper in op het meetprogramma. Het meetprogramma is het brein van de opstelling en zal de afzonderlijke hardwarecomponenten aansturen.

Dit hoofdstuk bestaat uit twee delen. Het eerste deel 5.1.1 geeft aan hoe de gebruiker de meetopstelling moet gebruiken en welke instellingen hij/zij moet doen om een goede meting te krijgen. Het tweede deel 5.1.2 geeft beknopt weer welke principes er in het programma toegepast zijn.

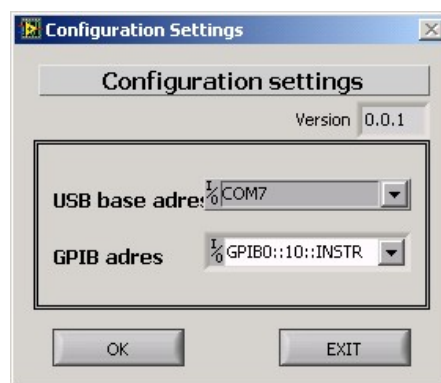
5.1.1 Gebruiksaanwijzingen

Elke keer dat de gebruiker het meetprogramma opstart, zal hij/zij de configuratiesettings moeten doen. Deze instellingen zijn de apparaatinstellingen zichtbaar in Figuur 5.1. Het meetprogramma moet immers weten op welke poort de apparaten aangesloten zijn.

De eerste instelling in dit scherm is de communicatiepoort waarmee de pc communiceert met het SASEBO-bord. Via deze weg geeft de pc instructies aan de FPGA.

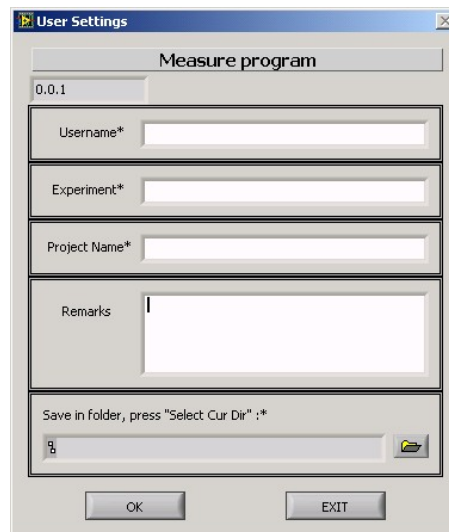
Hardwarematig is dit een USB-verbinding, maar de pc ziet deze verbinding als een seriëlepoort. Er is een pc-driver voorzien die de omzetting doet van seriëel naar USB en omgekeerd. De volgende instelling geeft aan welk adres op de GPIB-bus de oscilloscoop heeft. Dit adres is ingesteld door de gebruiker via het bedieningspaneel van de oscilloscoop.

Deze essentiële apparaatinstellingen blijven gedurende de uitvoering van het programma altijd hetzelfde. De gebruiker kan deze instellingen enkel bij het opstarten van het programma veranderen.



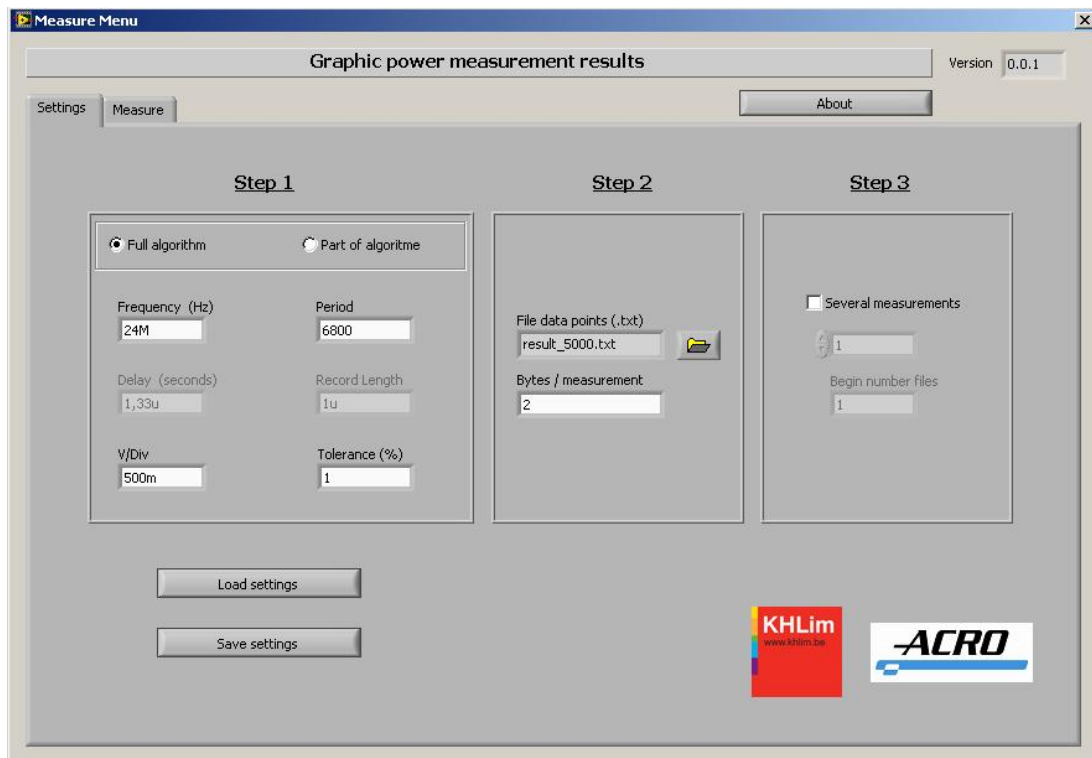
Figuur 5.1: Configuratiesettings van het meetprogramma

Nadat de gebruiker de apparaatinstellingen gedaan heeft, zal het meetprogramma vragen om enkele gebruikersinstellingen te doen. Deze instellingen zijn algemene gegevens over de gebruiker en het project. Met deze gegevens maakt het meetprogramma op de hardeschijf een unieke mappenstructuur aan voor elke gebruiker en project. De pc slaat na elke meting de meetdata op in deze mappen. De mappenstructuur zorgt ervoor dat de gebruiker zijn/haar meetdata later gemakkelijk kan terugvinden. Figuur 5.2 toont deze settings.



Figuur 5.2: Gebruikersinstellingen van het meetprogramma

Als de gebruiker alle systeem- en gebruikerafhankelijke instellingen gedaan heeft, start het programma verder op. Figuur 5.3 toont het hoofdmenu van het programma dat bestaat uit twee tabbladen namelijk 'Settings' en 'measure'. Het settingsmenu bevat de instellingen die metingsafhankelijk zijn. De gebruiker moet deze instellingen doen om de juiste data op te meten.



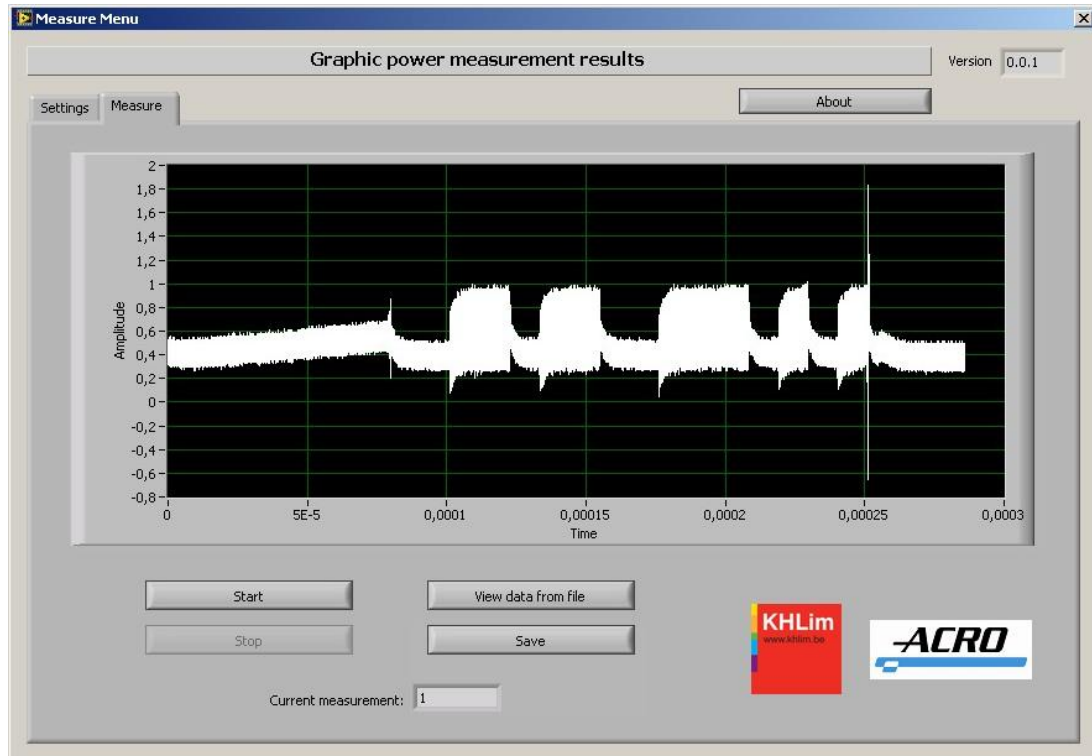
Figuur 5.3: Settingsmenu van het hoofdmenu

In het settingsmenu doorloopt de gebruiker drie verschillende stappen. De eerste stap bevat de instellingen die aangeven op welk tijdstip de gewenste data zit. De meetopstelling kan kleine vermogenintervallen binnen het volledige algoritme meten, maar ook het vermogenverbruik van het volledige algoritme.

Elke meetcyclus gebeurt met een andere klaartekst. Daarom zal de gebruiker in de tweede stap een bestand opgeven dat de klaarteksten bevat. Dit bestand bestaat uit hexadecimale getallen gescheiden door tabs. Elk hexadecimaal getal stelt een 8-bits getal voor. De grootte van de klaartekst is voor elk algoritme anders. De gebruiker moet in de tweede stap opgeven hoeveel hexadecimale getalen de pc moet doorsturen naar de FPGA per cyclus.

Voor sommige vermogenanalyses is er maar één meting nodig, voor andere meerdere. De gebruiker geeft deze informatie op in de derde stap. Indien voor het algoritme meerdere metingen nodig zijn, selecteert de gebruiker 'several measurements'. Het aantal geeft hij/zij op in het veld dat dan verschijnt.

De gebruiker heeft de mogelijkheid om de instellingen op te slaan. Deze instellingen kan hij/zij later ophalen om opnieuw te gebruiken.



Figuur 5.4: Meetmenu van het hoofdmenu

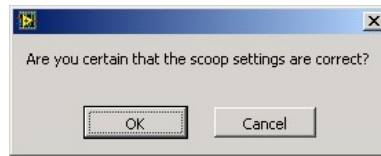
In het meetmenu uit Figuur 5.4 kan de gebruiker de metingen starten. Het meetprogramma voert de metingen uit volgens de instellingen in het settingsmenu. Als het meetprogramma alle metingen heeft afgerond, kan de gebruiker een aantal metingen bekijken.

Indien de gebruiker koos voor meerdere metingen, slaat de meetopstelling deze automatisch op in de map die aangemaakt is met de behulp van de gebruikersinstellingen. Als de gebruiker koos voor één meting, moet hij/zij deze zelf opslaan. Dit is mogelijk in het meetmenu van het programma.

Verwittigen

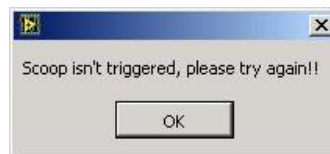
Als het meetprogramma vermoedt dat de gebruiker foute informatie opgaf, zal de meetopstelling de gebruiker hiervan op de hoogte brengen. Hieronder staan de verwittigen, die de meetopstelling kan geven, kort opgesomd.

- Een verwittiging zoals Figuur 5.5 verschijnt nadat de gebruiker op de startknop heeft gedrukt. De meetopstelling vraagt of de gebruiker er zeker van is dat hij/zij alle instellingen in het settingsmenu correct gedaan heeft.



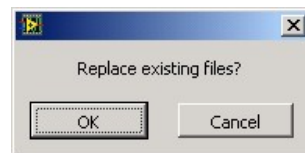
Figuur 5.5: Verwittiging: Zijn de scope-settings correct?

- Het triggersignaal van de oscilloscoop is een belangrijk signaal. Als de oscilloscoop niet triggert, kan het meetprogramma geen meting uitvoeren. Het meetprogramma start de meting. Als het programma na enkele seconden geen data ontvangt van de oscilloscoop, stopt de meetcyclus. De gebruiker wordt hiervan op de hoogte gesteld door de verwittiging in figuur 5.6.



Figuur 5.6: Verwittiging: Scope niet getriggerd.

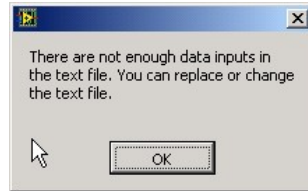
- Soms start de gebruiker een nieuwe meting met dezelfde gebruikersinstellingen. Dit geeft problemen want deze nieuwe meting zal bestaande data overschrijven. Het meetprogramma vraagt aan de gebruiker of de nieuwe meting deze bestanden mag overschrijven. Figuur 5.7 toont deze verwittiging.



Figuur 5.7: Verwittiging: Vervang bestaande bestanden?

- Als de gebruiker een databestand met te weinig klaartekst heeft opgegeven, zal de meetopstelling dit aangeven met een pop-up als Figuur 5.8. De gebruiker kan dan

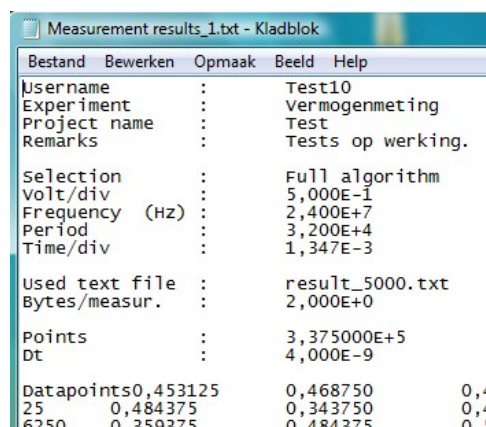
kiezen uit twee opties. Ten eerste kan de gebruiker het aantal metingen verkleinen. Ten tweede kan de gebruiker zoeken naar een groter databestand.



Figuur 5.8: Verwittiging: Te weinig data-inputs in de tekst file.

Inhoud van het databestand

Figuur 5.9 toont een databestand dat de meetopstelling aanmaakt na elke meting. De eerste vier regels bevatten gebruikersinstellingen. De volgende zeven regels bevatten de instellingen die de gebruiker deed in het settingsmenu. Daarna volgen nog twee regels die informatie geven over de datapunten. Na al deze instellingen komen de data-punten van de meting. Elk punt is gescheiden met een tab.



Measurement results_1.txt - Kladblok			
Bestand Bewerken Opmaak Beeld Help			
Username	:	Test10	
Experiment	:	Vermogenmeting	
Project name	:	Test	
Remarks	:	Tests op werking.	
Selection	:	Full algorithm	
volt/div	:	5,000E-1	
Frequency (Hz)	:	2,400E+7	
Period	:	3,200E+4	
Time/div	:	1,347E-3	
used text file	:	result_5000.txt	
Bytes/measur.	:	2,000E+0	
Points	:	3,375000E+5	
Dt	:	4,000E-9	
Datapoints	:	0,453125	0,4
25	:	0,484375	0,4
6250	:	0,350375	0,5

Figuur 5.9: Voorbeeld log bestand

5.1.2 Gebruikte programmeerprincipes

De gebruikte programmeermethodes komen hier aan bod en we leggen uit waarom we deze gebruiken.

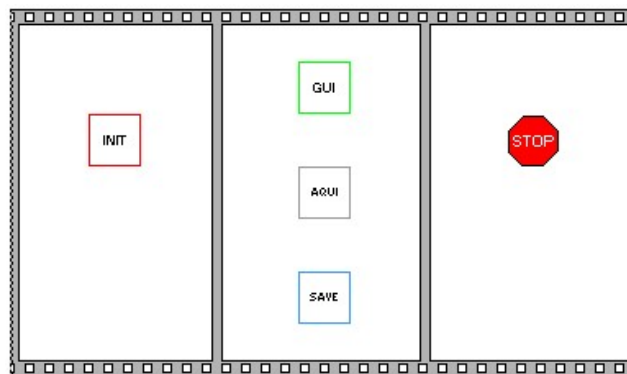
Sequentieel/multithreaded

Het meetprogramma start altijd vanaf de 'main' VI. Deze VI bestaat uit een groep zelfgeschreven bouwblokken. Het programma doorloopt een aantal van deze bouwblokken sequentiëel. Naast deze sequentiële uitvoer bevat het programma ook een aantal bouwblokken die multithreaded uitgevoerd worden.

Een programmeur gebruikt sequentieel programmeren als bepaalde stappen achter elkaar moeten gebeuren. Bepaalde systemen laten enkel een sequentiële structuur toe. Denk maar aan microcontrollers.

Een multithreaded applicatie is gebruikelijk bij programma's waar meerdere dingen gelijktijdig mogen gebeuren. De programmeur deelt het programma op in kleine deelprogramma's met elk hun eigen verantwoordelijkheden. Deze deelprogramma's kunnen met elkaar communiceren. Deze manier van werken leidt meestal tot een robuust geheel waar de programmeur op een eenvoudige manier het programma kan aanpassen.

Ons programma maakt gebruik van zowel sequentiele als multithreaded programeerstijl. De instellingen en initialisatie gebeuren voordat het meetgedeelte start. Na de instellingen en initialisatie maken we gebruik van een aantal multithreaded modules om de meting uit te voeren. Op Figuur 5.10 is de sequentiële structuur aangegeven door de filmstrip. De plaatsen waar meerdere modules binnen een vak staan, worden multithreaded uitgevoerd.



Figuur 5.10: Blockdiagram van main.vi

Locale en globale variabelen

Het meetprogramma maakt gebruik van locale en globale variabelen. Locale variabelen zijn variabelen die enkel bekend zijn binnen de subVI. Dit kunnen de

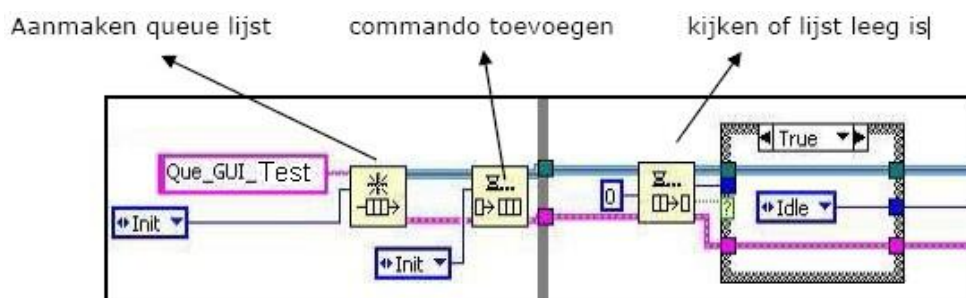
bedieningselementen en indicatoren van het frontpaneel van de subVI zijn.

De meeste programma's bestaan uit verschillende frontpanelen (dus ook subVI's) waarop de gebruiker gegevens kan invullen. Als andere subVI's deze gegevens gebruiken, moet de programmeur deze gegevens in globale variabelen steken. Globale variabelen zijn binnen het hele programma bekend.

De instellingen die de gebruiker doet in de loop van het meetprogramma slaan we op in globale variabelen. Hierdoor kan elke programma onderdeel deze instellingen bekijken en aanpassen indien nodig. Andere variabelen in het meetprogramma zijn meestal lokale variabelen.

Queued message handler

LabVIEW gebruikt de 'queued message handler' om multithreaded blokken met elkaar te laten communiceren. Elk proces heeft een 'queued message handler'. Dit is een FIFO waar de andere parallellopende processen commando's inschuiven. Een proces voert één voor één deze commando's uit. Figuur 5.11 toont een queuelijst.



Figuur 5.11: Voorbeeld van een Queuelijst

Onze meetopstelling gebruikt deze 'message handler' om commando's uit te wisselen tussen de gebruikersinterface, het acquisitieblok en het opslagblok.

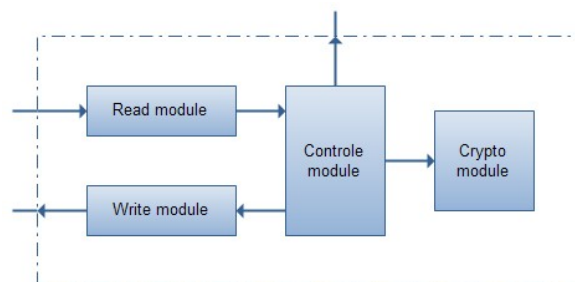
5.2 VHDL-wrapper

De meeste cryptografische bouwblokken zijn zo ontworpen dat er een data-in en data-out beschikbaar is. Verder zijn er ook nog een aantal controlelijnen. De meetopstelling moet startsignalen en data doorgeven aan dit blok. Er moet dus naast het cryptografische

bouwblok ook nog een communicatieblok in de FPGA zitten dat de communicatie tussen de FPGA en de pc regelt.

Aan dit communicatieblok zit het cryptografische blok dat zijn data en controlesignalen ontvangt van het communicatieblok. Elk cryptografisch blok waarop een vermogenaanval moet gebeuren, zal gekoppeld moeten worden aan deze wrapper.

De VHDL-code van deze wrapper bevindt zich op de CD en noemt PWrap en bestaat uit vier modules. De eerste module is het cryptografische blok. Deze module is voor elke vermogenaanval anders. De tweede module (RD-usb) is een module die de controlesignalen en data van de pc binnenkrijgt en doorgeeft aan de vierde module. De derde module (WR-usb) is een module die controlesignalen van de vierde module doorgeeft aan de pc. De vierde module is een controlemodule die ervoor zorgt dat de overige drie modules goed met elkaar verbonden zijn en die ervoor zorgt dat het communicatieprotocol gerespecteerd wordt.



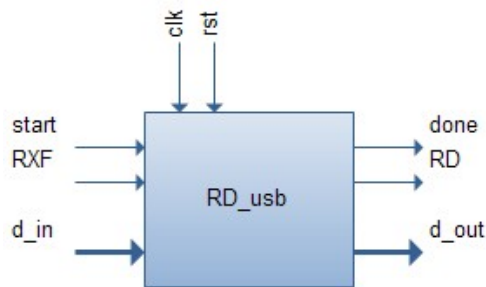
Figuur 5.12: Koppeling tussen de modules van de VHDL-wrapper

5.2.1 RD-usb.vhd

RD_usb.vhd bevat de module die verantwoordelijk is voor de signalen uit te lezen die in de readbuffer van de FT245R zitten.

Port-map van de module

De RD, RXF en d.in van de module, zichtbaar in Figuur 5.13, zijn verbonden met de in- en uitgangspinnen van de FPGA waar ook de RD, RXF en data pinnen van de FT245R mee verbonden zijn. De rst is de asynchrone reset van heel het systeem. De clk is verbonden met de klok van de FPGA.



Figuur 5.13: Port-map van RD_usb module

Werking

Intern werkt deze module met een FSM die werkt zoals in de flowchart van Figuur 5.14 beschreven staat.

Wanneer het systeem gereset wordt, zal de FSM terecht komen in de reset_state. In deze state worden alle uitgangen met de juiste rustwaardes geladen. De volgende klokpuls zal de FSM overgaan naar de start_state.

In de start_state zal de FSM wachten op een startsignaal. De data-uitgang blijft zijn vorige waarde behouden in deze state en de RD-lijn wordt hoog gehouden. Indien het startsignaal hoog wordt, zal de FSM bij de volgende klokpuls overgaan naar de load_state. Als het startsignaal laag blijft, zal de FSM in de start_state blijven.

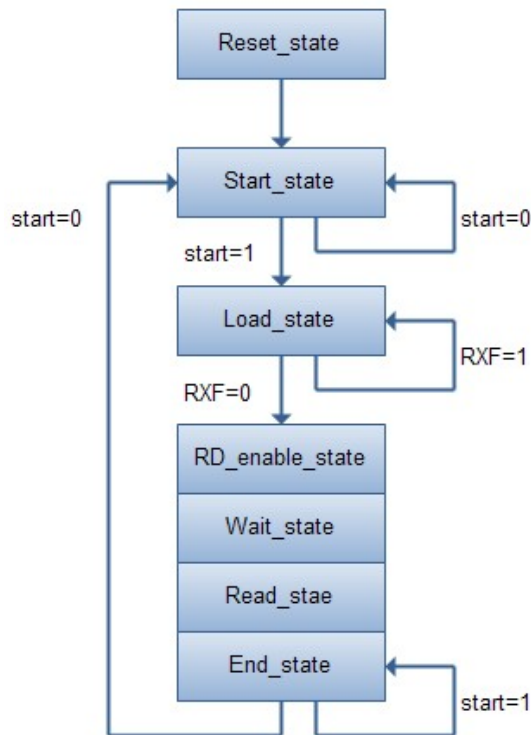
In de load_state kijkt de FSM of de RXF-lijn hoog of laag is. De data-uitgang van de module wordt gereset en de RD-lijn blijft hoog. Indien de RXF-lijn hoog is, zal de FSM in de load_state blijven. Als de RXF-lijn laag is, zal de FSM bij de volgende klokpuls overgaan naar de RDenable_state.

In de RD_enable_state maakt de FSM de RD-lijn laag. Bij de volgende klokpuls gaat de FSM over naar de wait_state.

De wait_state is een dummy-state die ervoor zorgt dat de 'RD Active to Valid Data' tijd gerespecteerd wordt. Na de wait_state gaat de FSM over naar de read_state.

In de read_state wordt de data ingang ingelezen. De data-uitgang neemt de waarde van de data-ingang aan. Na de read_state gaat de FSM over naar de end_state.

In de end_state gaat de FSM het done-sigitaal hoog maken en de RD-lijn terug hoog maken. De data-uitgang blijft zijn vorige waarde behouden. In de end_state wordt gewacht tot het startsignaal terug laag wordt. Als het startsignaal terug laag wordt, gaat de FSM terug over naar de start_state bij de volgende klokpuls. Nu kan er een nieuwe read gedaan worden.

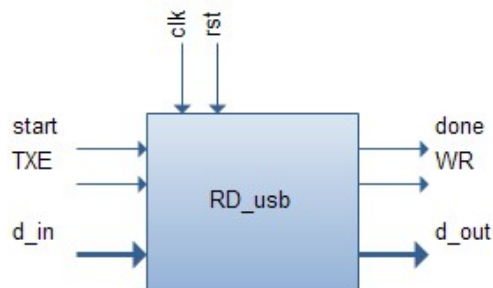


Figuur 5.14: Flowchart van RD_usb FSM

5.2.2 WR_usb.vhd

WR_usb.vhd bevat de module die verantwoordelijk is voor de signalen die afkomstig zijn van de wrapper te schrijven naar de writebuffer van de FT245R. Figuur 5.15 toont de port-map van deze module

Port-map van de module



Figuur 5.15: Port-map van WR_usb module

De WR, TXE en d_out van deze module zijn verbonden met de in- en uitgangspinnen van de FPGA waar ook de WR, TXE en datapinnen van de FT245R mee verbonden zijn. De rst is de asynchrone reset van heel het systeem. De clk is verbonden met de klok van de FPGA.

Werking

Intern werkt deze module met een FSM die werkt zoals in de flowchart van Figuur 5.16 beschreven staat.

Na een reset zal de FSM terechtkomen in de start_state. In de start_state zal de FSM wachten op een startsignaal. De data-uitgang is hoogimpedant in deze state en de WR-lijn wordt laag gehouden. Indien het startsignaal hoog wordt, zal de FSM bij de volgende klokpuls overgaan naar de load_state. Als het startsignaal laag blijft, zal de FSM in de start_state blijven.

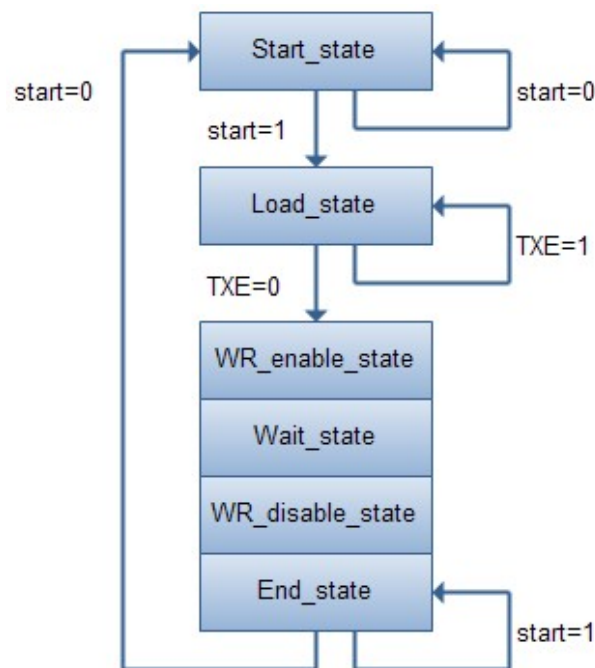
In de load_state kijkt de FSM of de TXE-lijn hoog of laag is. De data-uitgang van de module blijven hoogimpedant en de WR-lijn blijft laag. Indien de TXE-lijn hoog is, zal de FSM in de load_state blijven. Als de TXE-lijn laag is, zal de FSM bij de volgende klokpuls overgaan naar de WRenable_state.

In de WRenable_state maakt de FSM de WR-lijn hoog. Bij de volgende klokpuls gaat de FSM over naar de wait_state.

In de wait_state wordt de WR-lijn hoog gehouden en wordt de data op de data-uitgang gezet. Na de wait_state gaat de FSM over naar de WRdisable_state.

In de WR_disable_state wordt de data die op de uitgang staat in de writebuffer van de FT245R geplaatst. Na de WRdisable_state gaat de FSM over naar de end_state.

In de end_state gaat de FSM het done-sigitaal hoog maken en de WR-lijn laag houden. De data-uitgang is terug hoogimpedant. In de end_state wordt gewacht tot het startsignaal terug laag wordt. Als het startsignaal terug laag wordt, gaat de FSM terug over naar de start_state bij de volgende klokpuls. Nu kan er een nieuwe write gedaan worden.

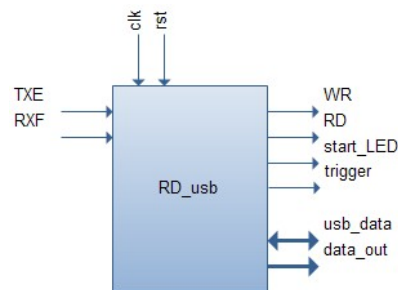


Figuur 5.16: Flowchart van WR_usb FSM

5.2.3 Wrapper.vhd

Wrapper.vhd bevat de top-module die verantwoordelijk is voor de goede samenhang van alle andere modules. Deze module respecteert het afgesproken communicatieprotocol. De port-map van deze module staat in Figuur 5.17

Port-map van de module



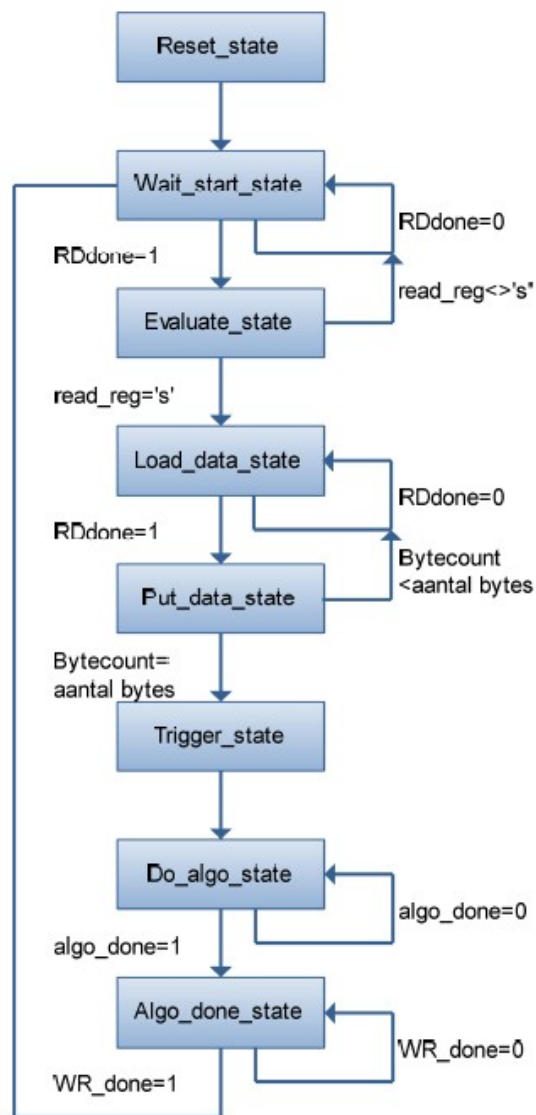
Figuur 5.17: Port-map van de wrapper module

De RD, WR, TXE, RXF en usb_data van deze module zijn verbonden met de in- en uitgangspinnen van de FPGA waar ook de RD, WR, TXE, RXF en de datapinnen van de FT245R mee verbonden zijn. De rst is de asynchrone reset van heel het systeem. De clk is verbonden met de klok van de FPGA.

Het triggersignaal is een signaal dat ervoor zorgt dat de scoop getriggerd wordt op de stijgende flank van dit signaal. De start_LED uitgang dient voor de signalisatie en geeft aan wanneer een nieuwe meetcyclus gestart wordt.

Werking

Intern werkt deze module met een FSM die werkt zoals in de flowchart in Figuur 5.18 beschreven staat.



Figuur 5.18: Flowchart de wrapper FSM

Na een reset zal de FSM terechtkomen in de reset_state. In deze state worden alle uitgangen en interne signalen op een initiële waarde gezet. De FSM zal overgaan naar de wait_start_state bij de volgende klokpuls.

In de wait_start_state wordt het uitlezen van de readbuffer van de FT245R gestart. Zolang de RD_usb module niets ontvangen heeft, blijft de FSM in de wait_start_state. Als de RD_usb module iets ontvangen heeft, wordt RD_done hoog en zal de FSM bij de volgende klokpuls overgaan naar de evaluate_state.

In de `evaluate_state` wordt gekeken of dat het ontvangen karakter een 's' was. Indien dit niet het geval is, gaat de FSM bij de volgende klokpuls terug naar de `wait_start_state`. Als het ontvangen karakters overeenstemt met de 's' gaat de FSM bij de volgende klokpuls over naar de `load_data_state`.

De FSM blijft in de `Load_data_state` zolang er geen karakter ontvangen wordt. In deze state start de FSM weer een leescyclus door een start te geven aan de `RD_usb` module. Als de `RD_usb` module een karakter ontvangen heeft, is dit de eerste byte van de klaartekst. De FSM gaat de volgende klokpuls over naar de `Put_data_state`.

In de `Put_data_state` wordt het ontvangen karakter in het klaartekst register geplaatst. In deze state wordt ook een teller bijgehouden die aangeeft of het aantal databytes die nodig zijn voor het klaartekst register allemaal ontvangen zijn. Indien dit niet het geval is, zal de FSM de volgende klokpuls terug gaan naar de `load_data_state`. Als alle bytes ontvangen zijn, zit het klaartekst register vol en kan er overgegaan worden naar de `trigger_state`.

Wanneer de FSM in de `trigger_state` komt, wordt de scoop getriggerd. De triggering van de scoop moet altijd gebeuren voordat de cryptografische bewerkingen gestart worden. Als de FSM in de `trigger_state` zit, zal hij de volgende klokpuls overgaan naar de `do_algo_state`.

De FSM start in de `do_algo_state` de cryptografische bewerkingen en wacht hier tot deze gedaan zijn. Indien de wrapper een 'algo_done' signaal krijgt van de cryptografische module, gaat de FSM de volgende klokpuls over naar de `algo_done_state`.

De `algo_done_state` is de laatste state van een meetcyclus. De FSM start de `WR_usb` module met het karakter 'd' als data. Dit zorgt ervoor dat 'd' terug gezonden wordt naar het meetprogramma. Op deze manier weet de meetopstelling dat een meetcyclus afgelopen is en dat deze de data van de scoop mag binnenhalen. Na de `algo_done_state` gaat de FSM terug over naar de `wait_start_state`.

5.2.4 Opmerkingen

Controle- en cryptografische deel

Het SASEBO evaluatiebord bevat twee FPGA's: één controle FPGA en één cryptografische FPGA. De controle FPGA is een veel grotere FPGA dan de

cryptografische FPGA. Daarom is het veel handiger om de grote FPGA te gebruiken voor zowel de controle als de cryptografische bewerkingen.

Deze wrapper is geschreven voor de controle FPGA. We gaan ervan uit dat de cryptografische bewerkingen ook in deze FPGA uitgevoerd worden. Door de wrapper te herschrijven kan deze opgedeeld worden twee stukken. Het eerste stuk is een communicatiestuk dat uitgevoerd wordt op de controle FPGA. Het tweede stuk is een cryptografisch stuk dat uitgevoerd wordt op de cryptografische FPGA. Enkele verbindingslijnen tussen de FPGA's zorgen voor de juiste overdracht van de controlesignalen en de dataoverdracht tussen beide.

De opsplitsing in beide stukken wordt best gemaakt indien we kleine cryptografische algoritmes aanvallen. Het vermogenverbruik van de cryptografische FPGA bevat minder ruis. Dit komt omdat deze FPGA een stuk kleiner is en beter ontkoppeld.

Beperkingen

Het gebruik van deze wrapper legt enkele regels op aan de cryptografische module. Elke cryptografische module die gemeten wordt met deze wrapper, dient een algo-start, algo-done, data-in en een data-out signaal te hebben.

Cryptografische modules die niet aan deze eisen voldoen, moeten aangepast worden. Een andere mogelijkheid is het aanpassen van de wrapper.

5.3 Struikelblokken

5.3.1 Weergave van de kleine datavensters binnen een algoritme

De meetopstelling moet een gedetailleerde opname kunnen maken van een zeer kort tijdsvenster binnen het te analyseren algoritme. Dit is nodig omdat de gebruiker soms moet kijken naar een vermogenverandering die gebeurt op de stijgende klokflank.

De gedetailleerde opname vormt een probleem bij een lang algoritme waaruit één bepaalde klokpuls belangrijk is. Omdat de oscilloscoop slechts een eindig aantal punten kan opnemen, zal een groot algoritme voor een grotere onnauwkeurigheid zorgen dan een kleiner algoritme. Het startsignaal van de FPGA triggert de oscilloscoop en hierdoor zit er veel onnodige informatie in het oscilloscoopbeeld.

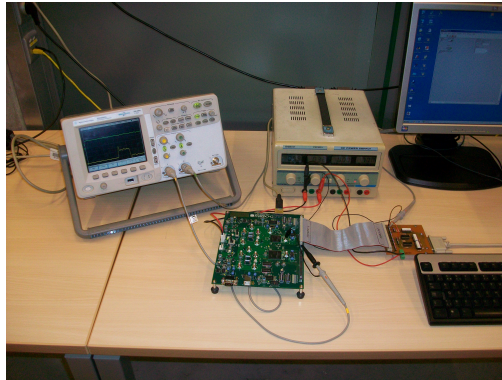
Het triggeren van de oscilloscoop bij het startsignaal van de FPGA is dus niet zo'n goed idee, maar wel nodig wanneer we het algemene vermogenverloop van het algoritme willen bekijken. Hierdoor kwamen we tot het besluit dat we twee extra parameters moesten voorzien voor de keren dat er een gedetailleerde weergave van een stukje binnen het algoritme nodig is.

Deze twee parameters definiëren de plaats waar de nuttige informatie zich bevindt na het starten van het algoritme. De eerste parameter geeft aan waar de nuttige informatie begint, terwijl de tweede parameter zegt hoelang de nuttige informatie duurt.

Met behulp van deze twee parameters kan de oscilloscoop zo ingesteld worden dat een gedetailleerde weergave van een stukje algoritme mogelijk wordt. De MSO6052A oscilloscoop heeft een variabele die aangeeft hoeveel tijd na de echte trigger de oscilloscoop moet triggeren. Deze variabele kan veranderd worden met het SCPI-commando ' :TIM:DEL <delay-time> '. Met de tweede parameter kunnen we de tijdsbasis zo instellen dat de nuttige informatie zeker binnen het beeld valt.

Voordat we deze functies integreerden in de meetopstelling, moesten we ze testen. De precieze werking was immers nog niet volledig duidelijk en we wisten niet of deze doordachte oplossing wel zou werken in de praktijk.

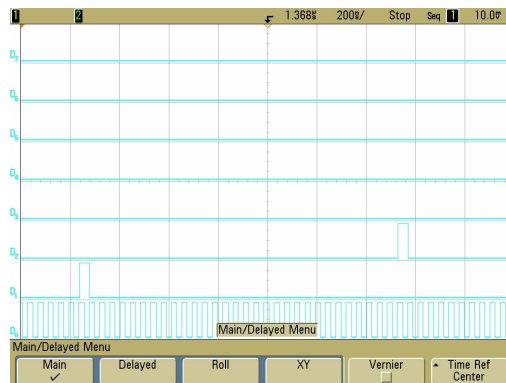
Om de functionaliteiten te testen hebben we eerst een VHDL-file geschreven die een triggerpuls en datapuls genereert. De datapuls volgt een aantal klokpulsen na de triggerpuls. Beide worden naar een uitgang van de FPGA gebracht. Daarna hebben we de delay-parameter van de oscilloscoop manueel ingesteld aan de hand van de tijd tussen de twee pulsen. Na het instellen van de delay-parameter en de trigger-source (kanaal 2) hebben we het eerste kanaal aangesloten op datapuls uitgang en het tweede kanaal op de triggerpuls uitgang. Figuur 5.19 toont deze opstelling.



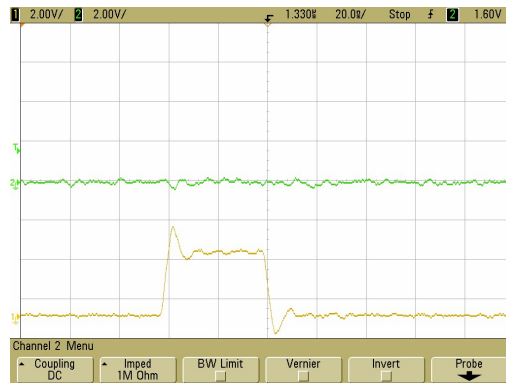
Figuur 5.19: Testopstelling voor gedetailleerde opname van stukken uit een algoritme

De nodige delay volgde uit een vermenigvuldiging van het aantal klokpulsen tussen trigger- en datapuls met de klokperiode van de systeemklok, $32 \times 1/24\text{MHz} = 1,33\mu\text{s}$.

Uit Figuur 5.20 en Figuur 5.21 kunnen we concluderen dat de opname zonder al te veel overbodige informatie opgenomen werd. Dit komt de nauwkeurigheid zeker ten goede.



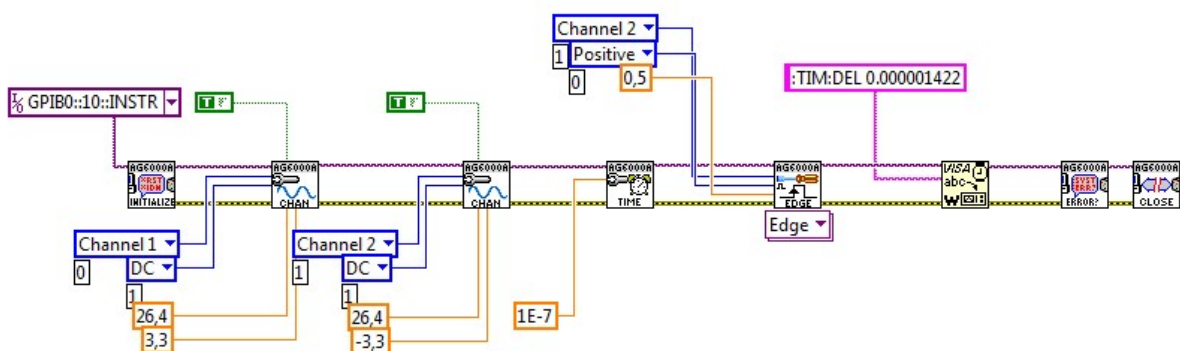
Figuur 5.20: Plot van de trigger- en datapuls in functie van de systeemklok



Figuur 5.21: Opname van de datapuls na een delayed-trigger door de triggerpuls

Het manueel instellen van de oscilloscoop was een feit, maar was dit ook mogelijk met de computer? We hebben een testprogramma in LabVIEW gemaakt, en hieruit bleek dat dit ook geen probleem was. We moesten enkel de manuele instellingen met de computer herhalen. Figuur 5.22 toont dit testprogramma.

- Trigger-source: kanaal 2,
- Triggermode: flank getriggerd,
- Trigger-edge: positief,
- Timedelay: uitgerekende tijd,
- Channel coupling: DC,
- Aquisition mode: single shot.



Figuur 5.22: LabVIEW testprogramma voor automatische instelling van de delayed-trigger

Het testprogramma staat op de bijlage-CD en heeft als naam '2_puls'.

5.3.2 Triggerbron van de Oscilloscoop

Bij het vastleggen van de specificaties van de meetopstelling namen we aan dat het triggersignaal voor de scoop hetzelfde zou zijn als het startsignaal van het SASEBO-bord. Dit bleek achteraf geen goede keuze. Bij het testen van een aantal algoritmes zagen we dat de pulsen voortdurend op een ander plaats kwamen liggen, ondanks de overeenkomstige instellingen van de scoop.

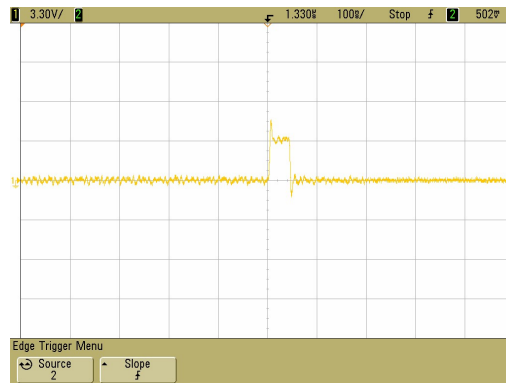
Dit is een probleem wanneer we meerdere metingen willen doen van een vermogenverloop dat plaatsvindt bij een bepaalde klokflank. Doordat bepaalde vermogenanalyses gebeuren door middel van de verschillen te vergelijken tussen dataset's, wordt het algoritme dus meerdere keren gestart.

Om een goede vermogenanalyse te kunnen doen moeten alle datasets exact hetzelfde tijdsvenster hebben binnen het algoritme. Als dit niet zo is, zullen we tussen de datasets verschillen opmerken. Deze verschillen zijn niet te wijten aan een verandering die optreedt door het uitvoeren van het algoritme. De verandering zijn geïntroduceerd door de meting zelf.

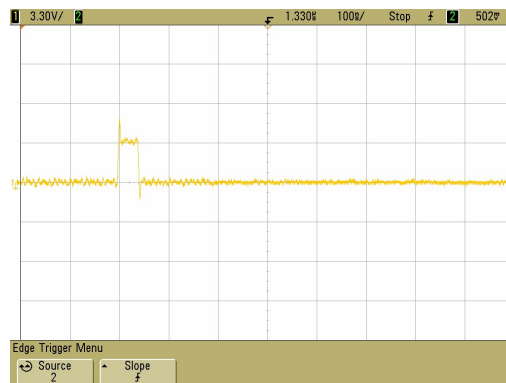
De oorzaak van deze verschuiving ligt bij de state-machine binnen de FPGA. Deze state-machine weet dat het algoritme gestart mag worden als er een startpuls komt. Wanneer de startpuls geweest is, zal de state-machine bij de volgende klokpuls het algoritme starten. Omdat de startpuls op elk moment kan komen zal tussen het starten van het algoritme en de startpuls steeds een verschillende tijd zitten.

Omdat deze startpuls de oscilloscoop onmiddellijk triggert en niet wacht totdat het algoritme gestart wordt, zal het tijdsvenster verschoven zijn. De trigger van de oscilloscoop is dus met andere woorden niet gesynchroniseerd met het starten van het algoritme.

Figuur 5.23 en Figuur 5.24 geven het verschuiven van het tijdsvenster weer. Deze beelden geven de datapuls van het algoritme gebruikt in 5.3.1. Beide oscilloscoop-beelden zijn met dezelfde oscilloscoop-instellingen opgenomen. Ook de trigger van de oscilloscoop was voor beide beelden dezelfde, namelijk het startsignaal voor de FPGA.

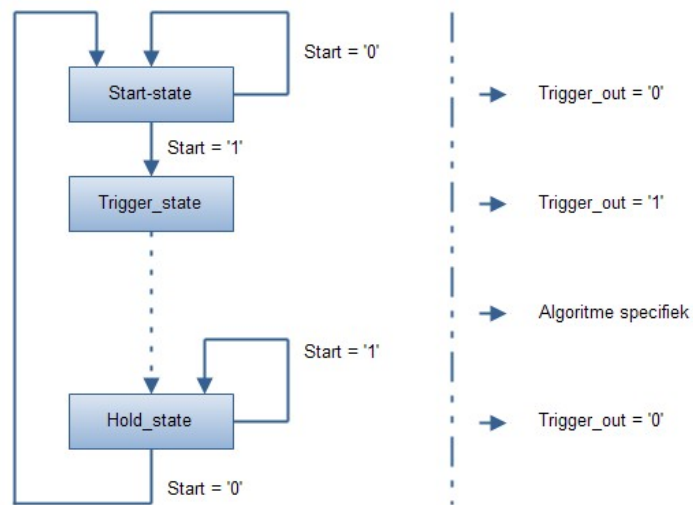


Figuur 5.23: Eerste opname van het gewenste tijdsvenster met triggering door startpuls



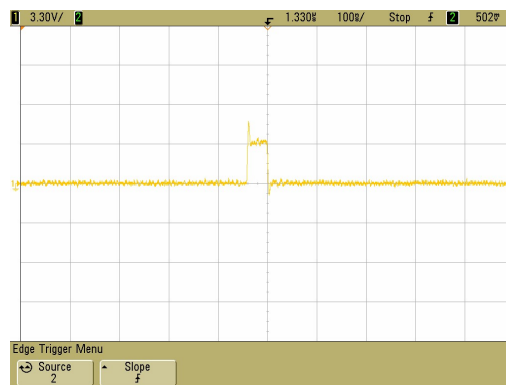
Figuur 5.24: Tweede opname van het gewenste tijdsvenster met triggering door startpuls

Deze verschuiving kunnen we tegen gaan door de FPGA het triggersignaal te laten genereren op het moment dat het algoritme start. Dit triggersignaal wordt hoog in de `trigger_state` direct na de `start_state` zoals weergegeven in Figuur 5.25

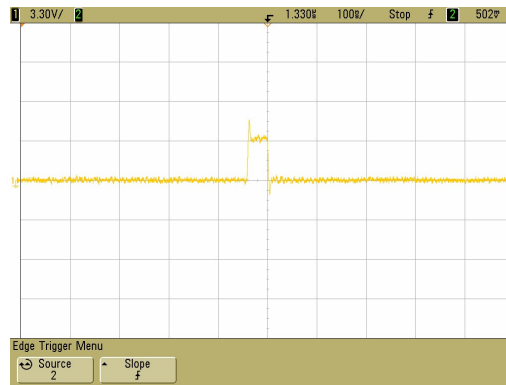


Figuur 5.25: Blokschematische voorstelling van een state-machine die een triggerpuls genereert juist voor het starten van het algoritme

Figuur 5.26 en Figuur 5.27 geven weer dat het tijdsvenster niet meer verschuift. De FPGA genereert nu een triggerpuls voor de oscilloscoop. Op deze manier is de trigger van de oscilloscoop wel gesynchroniseerd met het algoritme in de FPGA.



Figuur 5.26: Eerste opname van het gewenste tijdsvenster met triggering door FPGA



Figuur 5.27: Tweede opname van het gewenste tijdsvenster met triggering door FPGA

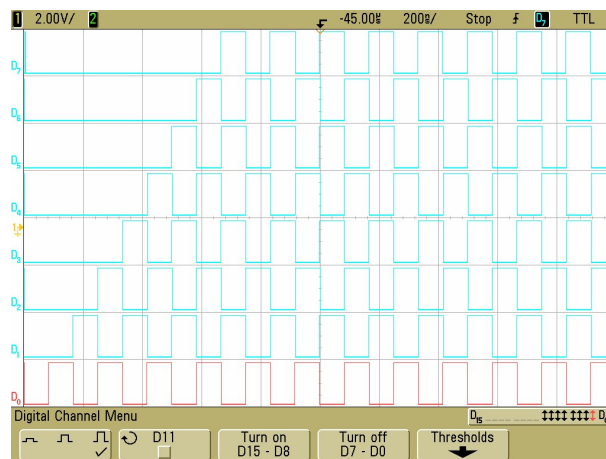
Hoofdstuk 6

Testanalyses

6.1 Vermogenanalyse van een schuifregister

6.1.1 Inleiding

De eerste vermogenanalyse deden we op een schuifregister waarvan de in te schuiven bit telkens wisselt van waarde. De ene keer is de bit één en de andere keer nul. Dit heeft als gevolg dat naarmate het schuifregister vol begint te raken, er steeds meer flip-flops omslaan. Dit principe is verduidelijkt door Figuur 6.1 en Figuur 6.2



Figuur 6.1: Basisprincipe van het te maken schuifregister

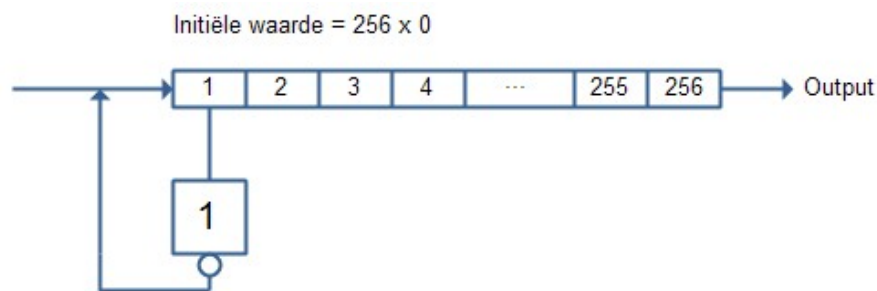
Deze analyse is geen analyse van een cryptografische bewerking, maar is bedoeld om met een eenvoudig voorbeeld te leren werken met met het SASEBO bord en de

meetopstelling. Het is de bedoeling dat we patronen leren herkennen in de vermogengrafiek en dit patroon linken aan de geschreven VHDL-code.

6.1.2 Opbouw van het algoritme

Het algoritme wordt gestart door een signaal van buitenaf. Dit signaal wordt gegeven door de meetopstelling. Voordat de meetopstelling het startsignaal aan de FPGA geeft, zal hij de oscilloscoop starten door een commando via GPIB door te sturen.

Na het starten van het algoritme wordt het 256-bits schuifregister geladen met nullen. Daarna zal het algoritme om de twee klokpulsen een nieuwe bit inschuiven in het register. Deze bit is het inverse van de vorige ingeschoven bit. Dit wordt 256 keer gedaan of met andere woorden totdat het schuifregister volledig gevuld is.

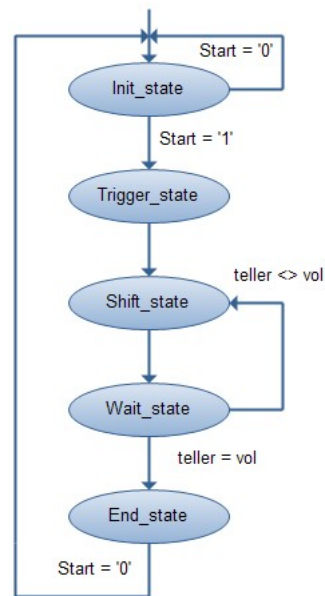


Figuur 6.2: Algoritme met schuifregister

Daarna zal de FPGA de stopuitgang als schuifregister hoog maken om aan te geven dat het algoritme afgelopen is. Na het detecteren van het stopsignaal zal de meetopstelling een stopcommando naar de oscilloscoop zenden via zijn GPIB interface. Gelijktijdig maakt de FPGA het schuifregister terug leeg. De FPGA is klaar voor een nieuwe meting.

6.1.3 Realisatie in VHDL

De VHDL-code van dit algoritme staat op de CD en noemt 'schuif_reg'. We hebben alles rondom een FSM opgebouwd waarvan het blokschema in Figuur 6.3 staat weergegeven.



Figuur 6.3: Blokschema van de FSM van het schuifregister algoritme

In de `Init_state` worden de tellers, die bijhouden hoeveel keer er geschoven werd, terug op nul gezet. De FPGA zal het schuifregister volledig opvullen met nullen en de stopuitgang hoog zetten/houden.

In de `Trigger_state` genereert de FPGA een puls die ervoor zal zorgen dat de oscilloscoop triggert.

In de `Shift_state` verhoogt de FPGA de teller met één en schuift de inverse van de vorige bit in. De stopuitgang is laag.

In de `Wait_state` gebeurt er niets. Deze state dient om een vertraging van één klokcyclus in te laten tussen twee schuifoperaties. Hier blijft de stopuitgang laag.

In de `End_state` blijft de statemachine wachten zolang het startsignaal hoog is. Indien het startsignaal laag wordt, gaat de statemachine terug naar de `Init_state`.

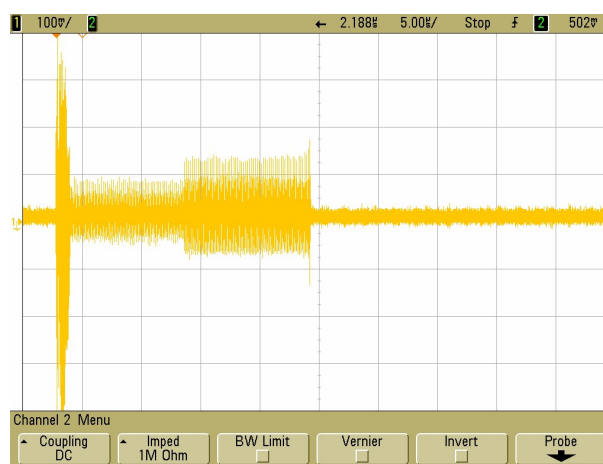
6.1.4 Resultaten

Handmatige vermogenanalyse

Hoewel dit schuifregister bedoeld was om de meetopstelling uit te testen, hebben we eerst een handmatige vermogenanalyse gedaan om te achterhalen welk resultaat we moesten

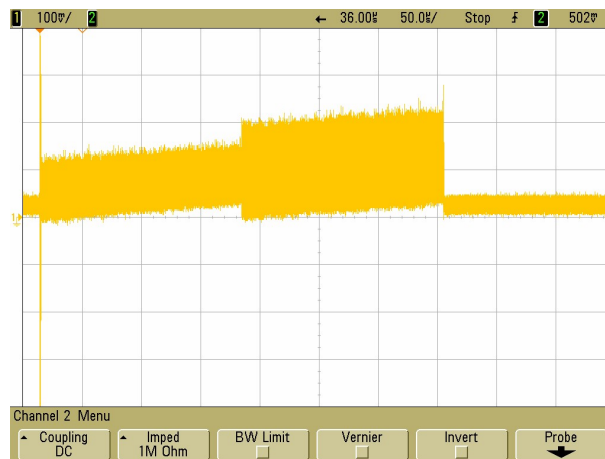
krijgen. Hoe kan je immers een resultaat evalueren, zonder het gewenste resultaat te kennen?

We hebben eerst een 256-bits schuifregister gebouwd zoals in 6.1.2 beschreven staat. Daarna hebben we het vermogen gemeten. Na metingen volgde dat er niets te zien was, buiten heel wat vervelende ruis. Dit is te wijten aan de slechte signaal/ruis verhouding. We meten immers over een één ohm weerstand, wat resulteert in een zeer kleine spanning die het vermogen voorstelt. Deze spanning zit in ordegrootte van 20 á 30 mV. Figuur 6.4 toont deze meting.



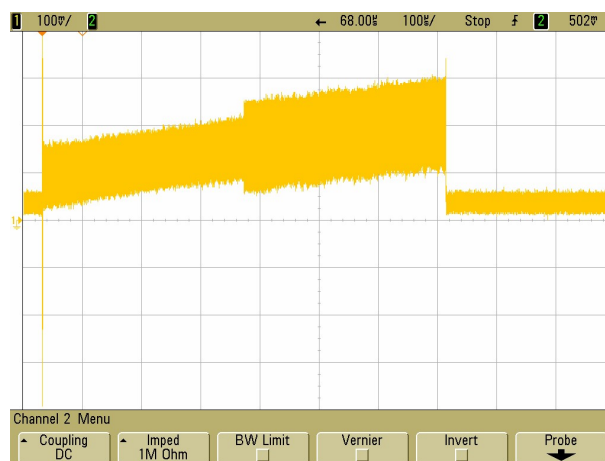
Figuur 6.4: Vermogen verloop over een één ohm weerstand bij een 256-bits schuifregister

We zochten naar een mogelijke verklaring voor het feit dat er geen echt vermogenpatroon zichtbaar was. We kwamen al snel tot de conclusie dat een 256-bits schuifregister veel te klein was om een vermogenpatroon uit de vermogenmeting te onderscheiden. We hebben het schuifregister vergroot tot een 4096-bits schuifregister en de vermogenanalyse opnieuw uitgevoerd. Deze keer was er wel een patroon te zien in de vermogenmeting. Figuur 6.5 toont deze meting.



Figuur 6.5: Vermogen verloop over een één ohm weerstand bij een 4096-bits schuifregister

Dit alles hebben nog eens overgedaan voor een 8192-bits schuifregister. Hier is het vermogenpatroon nog meer te onderscheiden dan bij het 4096-bits schuifregister. Figuur 6.6 toont deze meting.



Figuur 6.6: Vermogen verloop over een één ohm weerstand bij een 8192-bits schuifregister

Het valt op dat er op een bepaald moment in het vermogen verloop een sprong voorkomt. Deze sprong is het gevolg van het togglen van een uitgang. In de VHDL-file hebben we immers beschreven dat we het signaal in het midden van het schuifregister aftakken en naar buiten brengen.

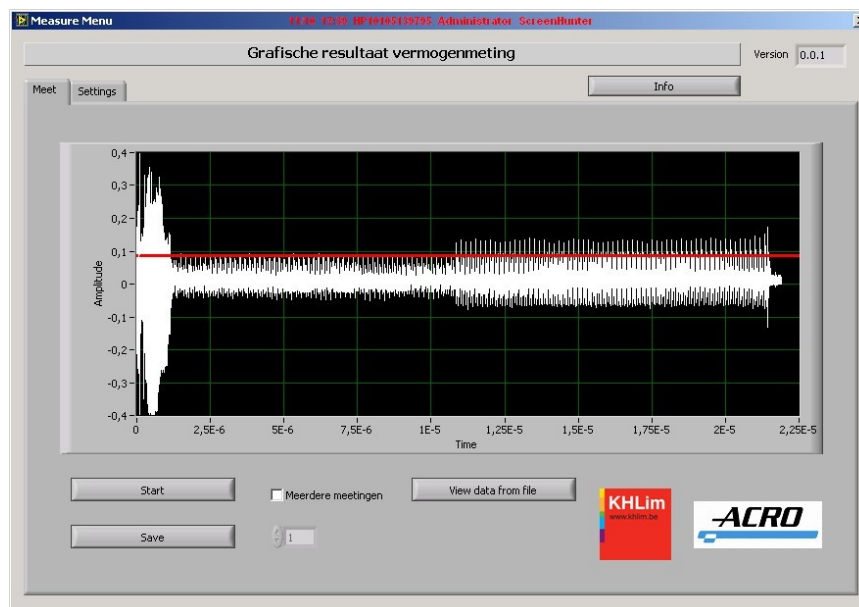
Na deze handmatige analyse weten we wat het resultaat is en wat we moeten bekomen met de meetopstelling. Deze tussenstap was zeker de moeite uit te voeren omdat we nu

weten dat een schuifregister van 256-bits niet voldoende is om een vermogenpatroon uit de vermogen analyse te herkennen. Wanneer we deze stap niet hadden gedaan twijfelden we na de analyse met de meetopstelling misschien aan de meetopstelling zelf, terwijl het eigenlijk het algoritme was dat niet deugde.

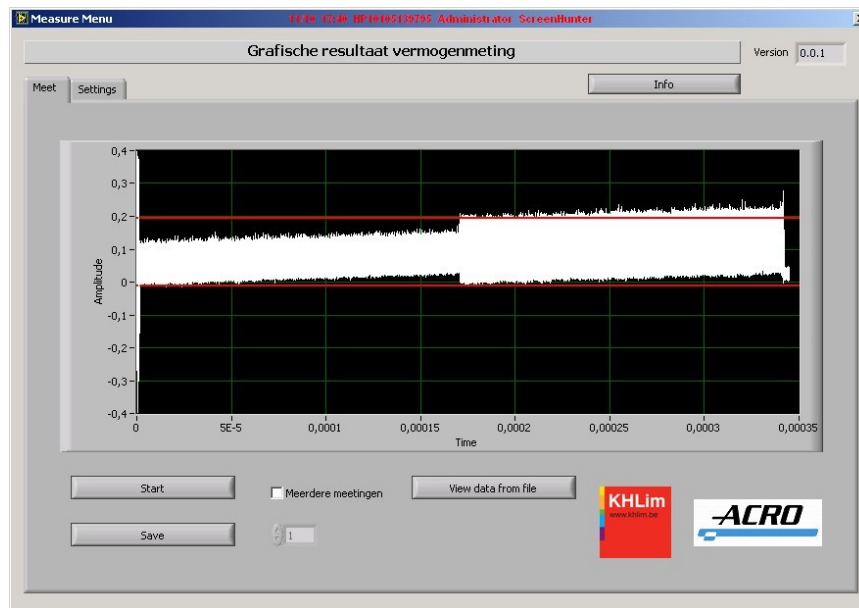
Vermogenanalyse met de meetopstelling

Na de manuele vermogenanalyse hebben we de vermogenanalyse uitgevoerd met de meetopstelling. Op deze manier kunnen we de werking van onze meetopstelling testen.

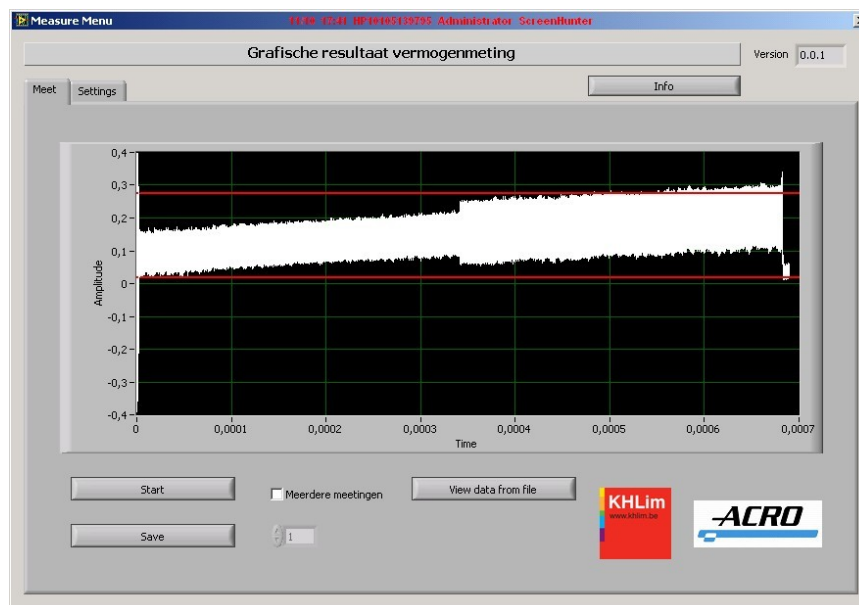
Deze vermogenanalyse is met de eerste versie van de meetopstelling gedaan en ziet er behoorlijk betrouwbaar uit. Figuur 6.7, Figuur 6.8 en Figuur 6.9 zijn screenshots van de analyses van het 256-, 4096- en 8192-bits schuifregister.



Figuur 6.7: Vermogenanalyse met behulp van de meetopstelling van het 256-bits schuifregister



Figuur 6.8: Vermogenanalyse met behulp van de meetopstelling van het 4096-bits schuifregister



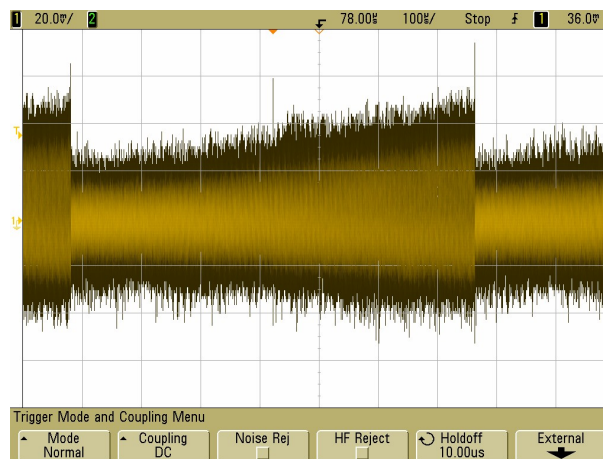
Figuur 6.9: Vermogenanalyse met behulp van de meetopstelling van het 8192-bits schuifregister

Wanneer we de resultaten van de handmatige analyse vergelijken met de analyse van de meetopstelling, zijn er geen verschillen. Hierdoor weten we dat de metingen correct zijn en kunnen we de meetopstelling verder perfectioneren.

6.1.5 Struikelblokken

Foute resultaten

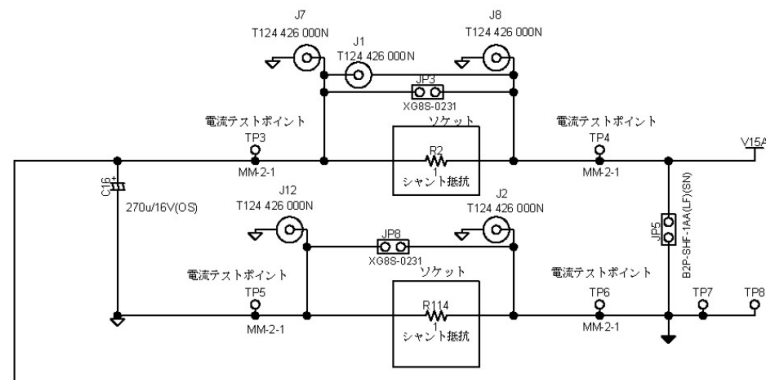
Nadat we de handmatige vermogenanalyse uitgevoerd hadden, zagen we dat het vermogenverloop niet aan de verwachtingen voldeed. Het opgemeten vermogen was zeer klein en werd zelfs negatief. Figuur 6.10 toont deze meting.



Figuur 6.10: Vermogen verloop over een één ohm weerstand bij een 8192-bits schuifregister

Het negatief worden van het vermogen is iets wat absoluut niet kan. We werken immers met een gelijkspanningsvoeding waarin we een zeer kleine seriële weerstand zetten om het vermogen te meten. We verwachtten dat het vermogen in rust niet nul is, maar een bepaalde waarde heeft. Dit vermogenverbruik zou moeten stijgen naarmate er meer flip-flop's togglen van het schuifregister.

Na een tijdje zoeken vonden we de fout. De manier waar op we meette was wel goed maar de instellingen van het SASEBO-bord waren fout. Figuur 6.11 is een schema van de meetinrichting op het SASEBO-bord.



Figuur 6.11: Vermogen meetinrichting van SASEBO-G

In deze vermogenmeetinrichting heeft men een serieweerstand in de voedingslijn en massalijn van de FPGA-core gezet. Zo kunnen we kiezen over welke weerstanden we meten. Door twee jumpers JP8 en JP3 te plaatsen worden deze weerstanden kortgesloten en is het alsof er geen weerstanden in de voedings- en massalijn staan.

Wanneer het vermogen in de voedingslijn gemeten wordt, is het nodig om te meten met differentieel probes. Als we dit niet doen, zal de voedingslijn via de oscilloscoop met de massalijn verbonden worden. Dit heeft een kortsluiting als gevolg.

Wanneer we het vermogen in de massalijn meten, moeten we dit niet differentieel doen. We kunnen de massaklem van de scoop aan TP5 hangen, omdat dit ook de massa van de schakeling is. Het signaal voor de oscilloscoop is afkomstig van TP6.

In eerste instantie meten we met probes tussen TP5 en TP6. Wanneer we verdere algoritmes analyseren met het SASEBO-bord kiezen we voor een BNC naar BNC connectie voor ruis te elimineren. Eén kant van de BNC-kabel wordt aangesloten op de oscilloscoop, terwijl de andere kant aangesloten wordt op J2.

Nu we dit weten, kunnen we verklaren wat er fout gegaan is bij de eerste meting. Tijdens de eerste meting was de probe aangesloten tussen TP5 en TP6. Deze aansluiting was goed maar om de meting te doen moest ook JP8 verwijderd worden en dat was niet gebeurd. Dit veroorzaakte de foute meting.

6.2 SPA-vermogenanalyse van een simpel algoritme

6.2.1 Inleiding

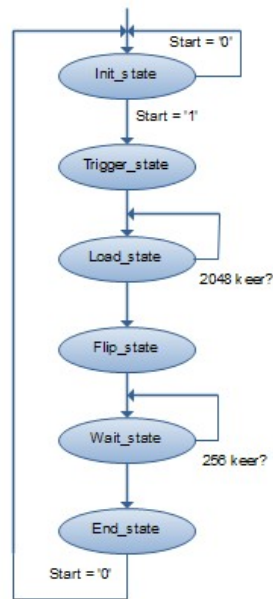
In deze sectie gebruiken we een zelfgeschreven algoritme waarop we een vermogenanalyse gaan doen. Dit algoritme is zo opgebouwd dat systematisch alle sleutelbits worden afgegaan. Afhankelijk van de waarde van de sleutelbit, wordt een bepaald register getoggled of niet.

Het uitgedachte algoritme is niet echt nuttig op vlak van cryptografie maar is vooral nuttig voor de werking van de Simple Power Attack aan te tonen. Door dit algoritme te analyseren krijgen we een beter inzicht over hoe je een SPA-vermogenaanval kan doen. Dit algoritme geeft een goed inzicht over hoe een SPA-gevoelig algoritme tot stand komt.

De principes die we leren uit deze analyse houden we in het achterhoofd om bestaande cryptografische algoritmes te verbeteren en te optimaliseren.

6.2.2 Opbouw van het algoritme

De FSM uit Figuur 6.12 bestaat uit verschillende states. Zo is er de `Init_state`, `Trigger_state`, `Load_state`, `Flip_state`, `Wait_state` en `End_state`. Elke state heeft een bepaalde functie die we hieronder kort toelichten.



Figuur 6.12: Opbouw van de FSM rondom het simpel SPA-algoritme

De `Init_state` is de state waarin de statemachine initieel begint. Bij een reset zal de FSM steeds naar deze state gaan. In deze state maakt de FPGA alle tellers en een flip-register leeg. Als dit allemaal gebeurt is mag de pc het algoritme starten. Indien de statemachine een startsignaal binnenkrijgt, zal hij overgaan naar de `Trigger_state`.

In de `Trigger_state` genereert de FPGA een puls op een uitgang. Deze puls dient als trigger voor de oscilloscoop. De statemachine verblijft slechts één klokpuls in deze state. Daarna gaat hij over naar de `Load_state`.

In de `Load_state` wordt het flip-register geladen met enen en nullen. Dit gebeurt op dezelfde manier als het schuifregisteralgoritme van 6.1. We gebruiken een 2048-bits flip-register. De statemachine blijft 2048 klokpulsen in deze state. Daarna gaat deze over naar de `flip_state`.

In de `Flip_state` zet de FPGA een vlag die het flip-register laat doorschuiven. Dit doorschuiven is afhankelijk van de waarde van de sleutelbits. Als de huidige sleutelbit één is, zal het flip-register gedurende 256 klokpulsen doorschuiven. Indien de huidige sleutelbit nul is, zal het flip-register gedurende 256 klokpulsen niet doorschuiven. In de `Flip_state` zorgt de FPGA ook voor het doorschuiven van de sleutel. Dit zorgt ervoor dat er telkens een andere bit geëvalueerd wordt. De statemachine verblijft slechts één

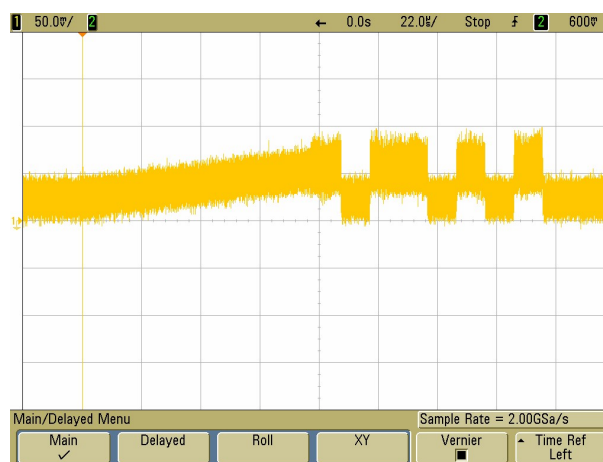
klokpuls in de Flip_state. Daarna gaat hij over naar de wait_state.

Na de Flip_state komt de Wait_state. In de Wait_state wacht de statemachine 256 klokpulsen terwijl het Flip-register al dan niet aan het schuiven is. In deze state kijkt de FPGA of alle sleutelbits overlopen werden. Indien dit het geval is, zal hij na de 256 klokpulsen terug naar de Init_state gaan. Indien nog niet alle sleutelbits overlopen zijn, zal de statemachine na de 256 klokpulsen terug naar de Flip_state gaan.

De VHDL-code van dit algoritme is terug te vinden op de CD en noemt 'SPA_ex1'.

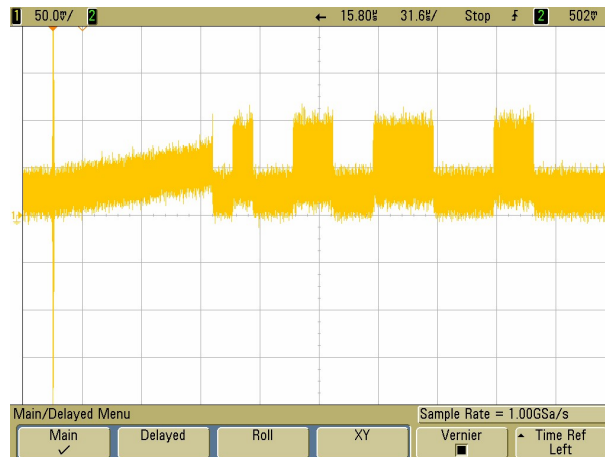
6.2.3 Resultaten

Figuur 6.13 geeft de resultaten van de vermogenanalyse. We onderscheiden twee grote delen. Als eerste zien we het vermogenverbruik van de initialisatiefase van het flip-register terugkomen. Als tweede stuk zien we het vermogenverbruik tijdens het sleutelafhankelijke schuiven van het flip-register.



Figuur 6.13: Resultaat van de SPA-vermogenanalyse met 8-bit sleutel

In bovenstaande analyse hebben we een 8-bits sleutel gebruikt met als waarde 10110101. We hebben deze analyse nog eens opnieuw uitgevoerd maar dan met een 16-bit sleutel. We hebben deze 16-bit sleutel de waarde 0100110011100011 gegeven.



Figuur 6.14: Resultaat van de SPA-vermogenanalyse met 16-bit sleutel

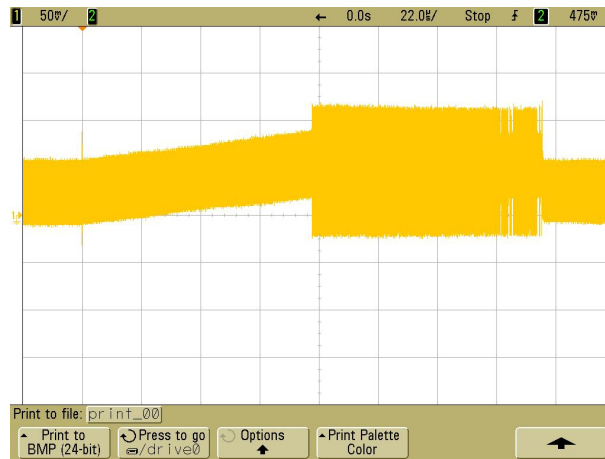
Uit Figuur 6.13 en Figuur 6.14 blijkt dat dit algoritme niet bestand is tegen een Simple Power Attack. De sleutel is immers rechtstreeks af te leiden uit het vermogenverbruik.

6.2.4 Verhinderen van een SPA-vermogenaanval

Het algoritme kan op een vrij simpele manier aangepast worden dat een Simple Power Attack niet meer mogelijk is. We moeten zorgen dat, onafhankelijk van de waarde van de sleutelbits, evenveel flip-flops in de FPGA togglen.

In ons algoritme kunnen we dit doen door een dummy-register in te voeren dat even groot is als het flip-register. Dit dummy-register krijgt in de Load_state dezelfde waarde als het flip-register. Wanneer we dit register laten schuiven wanneer de sleutelbit nul is, krijgen we hetzelfde vermogenverloop als we hebben wanneer de sleutelbit één is.

Figuur 6.15 geeft het resultaat weer bij het gebruik van een 8-bits sleutel.



Figuur 6.15: Resultaat van de SPA-vermogenanalyse met 8-bit sleutel

Hoofdstuk 7

SPA-aanval op een ECC-algoritme gebaseerd op Edwards-krommen

7.1 Inleiding

Het laatste deel van onze masterproefopdracht is het uitvoeren van een vermogenanalyse op een aantal cryptografische algoritmes.

In dit hoofdstuk doen we SPA-aanval op een ECC-algoritme dat gebaseerd is op Edwards-krommen. Twee ex-studenten van Katholieke Hogeschool Limburg, Niels Cornelissen en Christian Peeters, schreven dit algoritme voor hun masterproef in academiejaar 2008-2009. Voor een gedetailleerde omschrijving van dit algoritme verwijzen we naar hun eindwerktekst 'Compacte FPGA-implementatie van een cryptografisch systeem op basis van Edwards-krommen' [25].

Een inleiding tot de ECC-cryptografie is terug te vinden in hoofdstuk 2 van dit eindwerk. Informatie over SPA-aanvallen staat in hoofdstuk 3.

7.2 Werking van het algoritme

In deze sectie staat de minimale kennis die nodig is van het algoritme om de SPA-aanval te kunnen doen. Voordat we de vermogenanalyse beschrijven, is het immers nodig om kort toe te lichten volgens welk patroon de encryptie gebeurt.

Het cryptografische blok dat de encryptie uitvoert, krijgt de X-, Y- en Z-coördinaat van een punt op de Edwards-kromme als ingang. Na een enablesignaal encrypteert het blok

dit punt met een sleutel die intern in het blok is opgeslagen.

Het blok doet de versleuteling van het punt volgens het algoritme in Figuur 7.1. We veronderstellen dat de eerste bit van de sleutel altijd één is.

```
Gegeven:   klaartekst  $\rightarrow P(x,y,z)$   
             sleutel met lengte k  $\rightarrow S$   
             eerste sleutelbit = 1  
gevraagd: cijfertekst  $\rightarrow C(x,y,z)$   
Algoritme:  
  
1:    $P_1 = P$   
2:    $P_2 = 2.P$   
3:   for (l from  $k-2$  down to 0) do  
4:       if ( $S_l = 1$ ) then  
5:            $P_1 = P_1 + P_2$   
6:            $P_2 = 2.P_2$   
7:       else  
8:            $P_2 = P_1 + P_2$   
9:            $P_1 = 2.P_1$   
10:      end if  
11:  end for  
12:   $C = P_1$ 
```

Figuur 7.1: ECC-versleutelingsalgoritme

De FPGA overloopt de volledige sleutel. Op basis van de sleutelbit beslist de FPGA welke bewerkingen er gebeuren. Voor beide waarden gebeurt er een puntoptelling en een puntverdubbeling die even lang duren, maar uitgevoerd worden op andere registers. Theoretisch is dit algoritme dus beveiligd tegen SPA-aanvallen.

Het ontwerp van Niels Cornelissen en Christian Peeters werkt met een 163-bits sleutel. Uit hun metingen blijkt dat een puntoptelling 4968 klokcycli duurt. Een puntverdubbeling duurt 2124 klokcycli. De puntverdubbeling en puntoptelling bevatten een aantal modulaire vermenigvuldigingen en modulaire optellingen. Deze duren respectievelijk 180 en 16 klokcycli. Tabel 7.1 somt deze cycli op.

Tabel 7.1: Aantal klokcycli van ECC-algoritme N. Cornelissen & C. Peeters

Bewerking	Aantal klokcycli
Vermenigvuldiging	180
Optelling	16
Puntoptelling	4968
Puntverdubbeling	2124
Puntvermenigvuldiging	1203896

7.3 Communicatielogica en algoritme in controle-FPGA

Bij deze aanval kozen we ervoor om het algoritme en de communicatielogica samen in de controle-FPGA te implementeren. De controle-FPGA is immers groot genoeg en rondom deze FPGA zijn aansluitpunten voorzien om een vermogenmeting te doen.

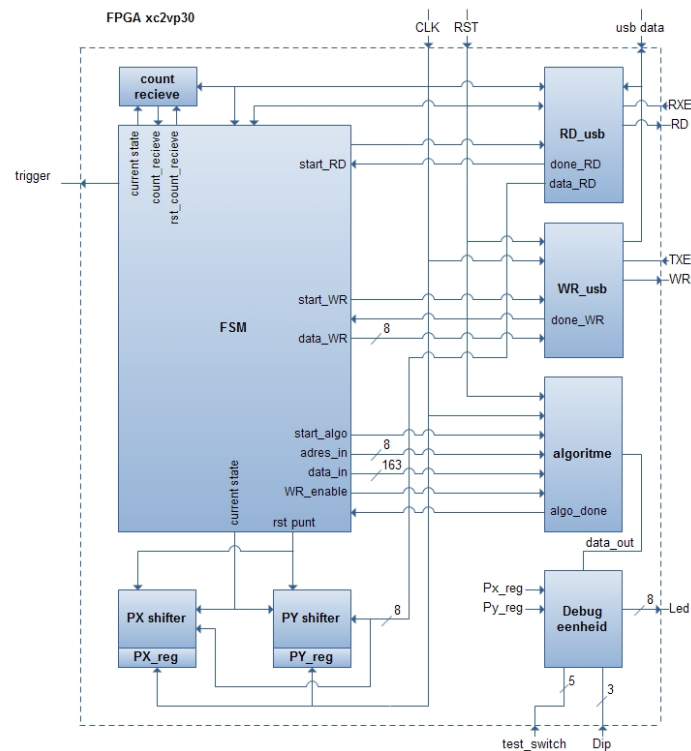
7.3.1 informeert over de algemene opbouw van de hardware in de controle-FPGA en 7.3.2 licht de werking toe van de FSM die de blokken aanstuurt.

7.3.1 Algemene opbouw

Figuur 7.2 geeft schematisch weer welke blokken we in de controle FPGA implementeerden om data te zenden/ontvangen naar/van de pc en het algoritme aan te sturen.

Alles is opgebouwd rondom een FSM die beschreven staat in Figuur 7.3. Deze FSM stuurt een aantal aparte blokken aan. RD_usb en WR_usb staan beschreven in 5.2.1 en 5.2.2. Het algoritme is het ECC-algoritme van N. Cornelissen & C. Peeters [25]. De debugeenheid stuurt een aantal LED's aan om de juiste werking van het algoritme aan te tonen.

Verder gebruikt de FSM nog een aantal registers, die gebruikt worden als teller.



Figuur 7.2: Algemeen blokschema van de VHDL code in de controle FPGA

7.3.2 FSM-flowchart

Figuur 7.3 geeft schematisch de werking van de FSM weer. Deze FSM begint altijd vanuit de Reset_state. De FPGA controleert voortdurend of hij bytes ontvangen heeft van de computer. Indien deze byte een 's' voorstelt, weet de FPGA dat er data gaat volgen.

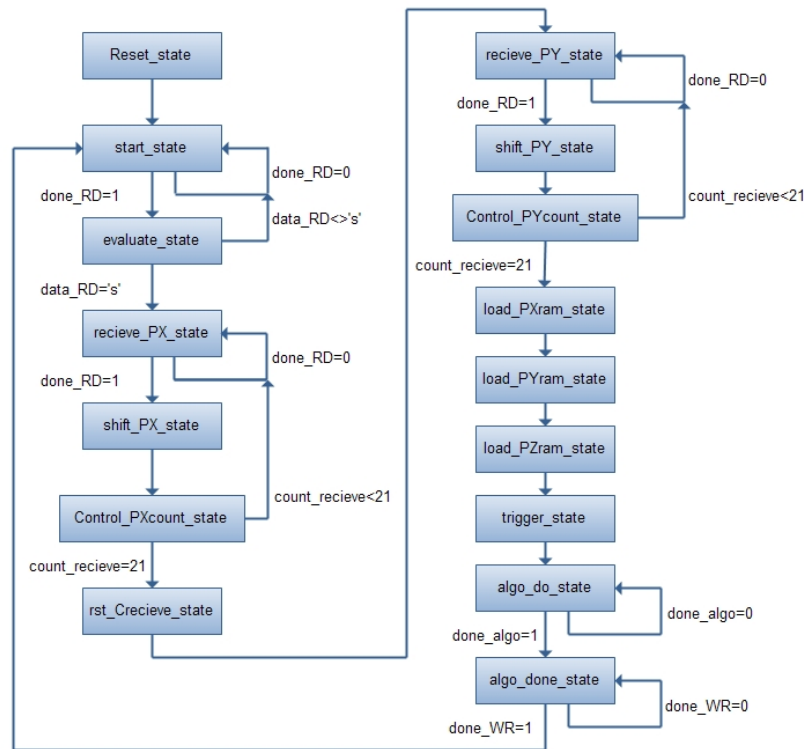
De data komt per byte binnen. Eerst wordt de X-waarde van het punt doorgestuurd, daarna volgt de Y-waarde. Beide waarden zijn 21 bytes groot. Telkens wanneer de FPGA een byte ontvangt, schuift deze in het juiste register.

Als alle bytes ontvangen zijn, gaat de FPGA de X-,Y- en Z-coördinaat van het punt in het RAM-geheugen van het algoritme laden. Daarna geeft de FSM een startsignaal aan het algoritme.

De FSM blijft wachten in de algo_do_state tot hij een done-sigitaal van het algoritme krijgt. Daarna zendt de FPGA een byte die een 'd' voorstelt terug naar de pc, om aan te geven dat het algoritme klaar is.

De FPGA gaat daarna terug naar de start_state waar hij wacht op een nieuw startsignaal van de pc.

De programmacode van deze FSM is terug te vinden op de CD.



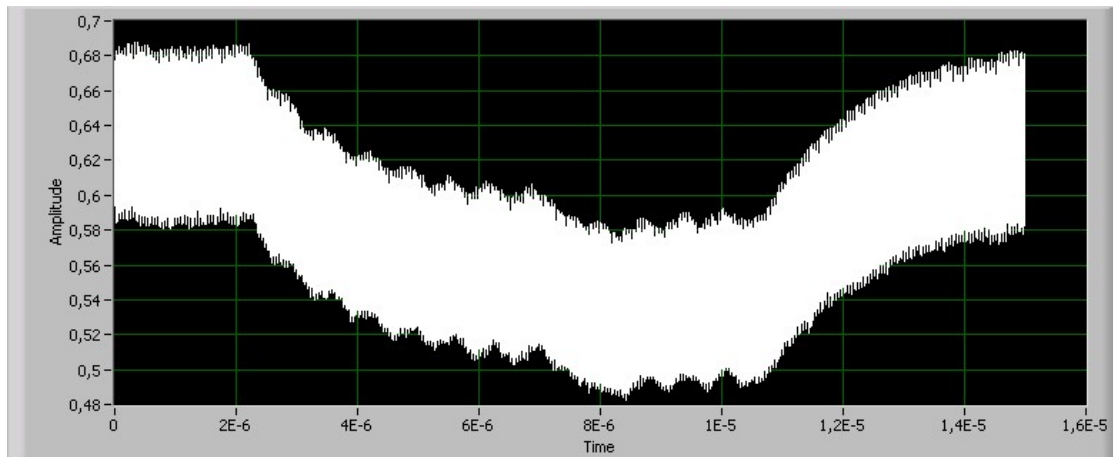
Figuur 7.3: Flowchart van de VHDL code in de controle FPGA

7.3.3 Resultaat

Figuur 7.4 geeft het vermogenverbruik van de FPGA tijdens de onregelmatigheid net na de vierde cyclus. In dit interval evalueert de FPGA de volgende sleutelbit en start op basis hiervan de volgende puntoptelling en puntverdubbeling. We zien duidelijk dat in deze opname veel capacatieve effecten zitten. Deze capacatieve effecten leiden ertoe dat veel nuttige informatie verdwijnt.

De schakeling rond de controle FPGA is niet geoptimaliseerd om vermogenmetingen uit te voeren. Het algoritme moet geïmplementeerd worden op de cryptografische FPGA en de controle FPGA moet de communicatie tussen de cryptografische FPGA en de computer doen. In 7.4 is de implementatie op bovenstaande manier gedaan en zijn de resultaten

ervan weergegeven.



Figuur 7.4: Vermogenverbruik van de controle FPGA tijdens de onregelmatigheid net na de vierde cyclus

7.4 Opsplitsing in controle- en cryptografische-FPGA

De implementatie van het algoritme en de controlelogica op de controle FPGA zorgden voor een groot probleem. Bij de vermogenmetingen die we uitvoerden waren belangrijke capacatieve effecten zichtbaar. Door deze capacatieve effecten waren de metingen niet duidelijk genoeg om er de juiste conclusies uit te trekken.

Door het algoritme op de cryptografische FPGA te implementeren, verwachtten we minder last te hebben van capacatieve effecten. De cryptografische FPGA is immers kleiner dan de controle FPGA en de randelektronica rondom is beperkt.

De pc communiceert nog altijd met het SASEBO-bord via de controle FPGA omdat de cryptografische FPGA geen USB-aansluiting heeft. Om de data over te brengen naar de cryptografische FPGA gebruiken we de parallele communicatieinterface tussen de cryptografische en controle FPGA.

7.4.1 VHDL-wrapper in controle-FPGA

Er zijn twee VHDL-wrappers nodig door de opslitsing die we gemaakt hebben. Ten eerste is er de VHDL-wrapper in de controle-FPGA en ten tweede is er de VHDL-wrapper in de cryptografische-FPGA.

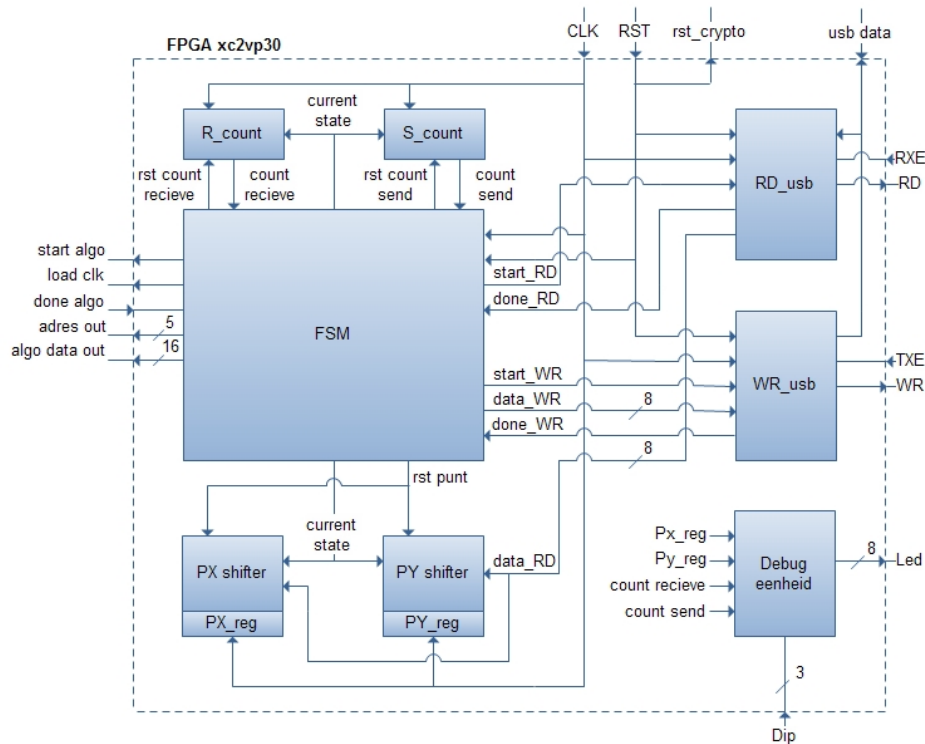
Deze sectie beschrijft de VHDL-wrapper die we gebruiken in de controle-FPGA. We geven de algemene opbouw van deze wrapper weer en beschrijven kort de werking van de gebruikte FSM.

Algemene opbouw

Figuur 7.5 geeft schematisch weer welke blokken we in de controle FPGA implementeerde om data te zenden/ontvangen naar/van de pc en data te zenden/ontvangen naar/van de cryptografische FPGA.

Alles is opgebouwd rondom een FSM die beschreven staat in Figuur 7.6. RD_usb en WR_usb staan beschreven in 5.2.1 en 5.2.2. De debugeenheid stuurt een aantal LED's aan om de juiste werking van de controlewrapper aan te tonen.

Verder heeft de FSM nog een aantal registers die gebruikt worden als teller.



Figuur 7.5: Algemeen blokschema van het programma in de controle FPGA voor het ECC-algoritme

FSM-flowchart

Figuur 7.6 geeft schematisch de werking van de FSM in de controle FPGA weer. Deze FSM begint altijd vanuit Reset_state. De FPGA controleert voortdurend of hij bytes ontvangen heeft van de computer. Indien deze byte een 's' voorstelt, weet de FPGA dat er data gaat volgen.

De data komt per byte binnen. Eerst wordt de X-waarde van het punt doorgestuurd, daarna volgt de Y-waarde. Beide waardes zijn 21 bytes groot. Telkens wanneer de FPGA een byte ontvangt, schuift hij dit in het juiste register.

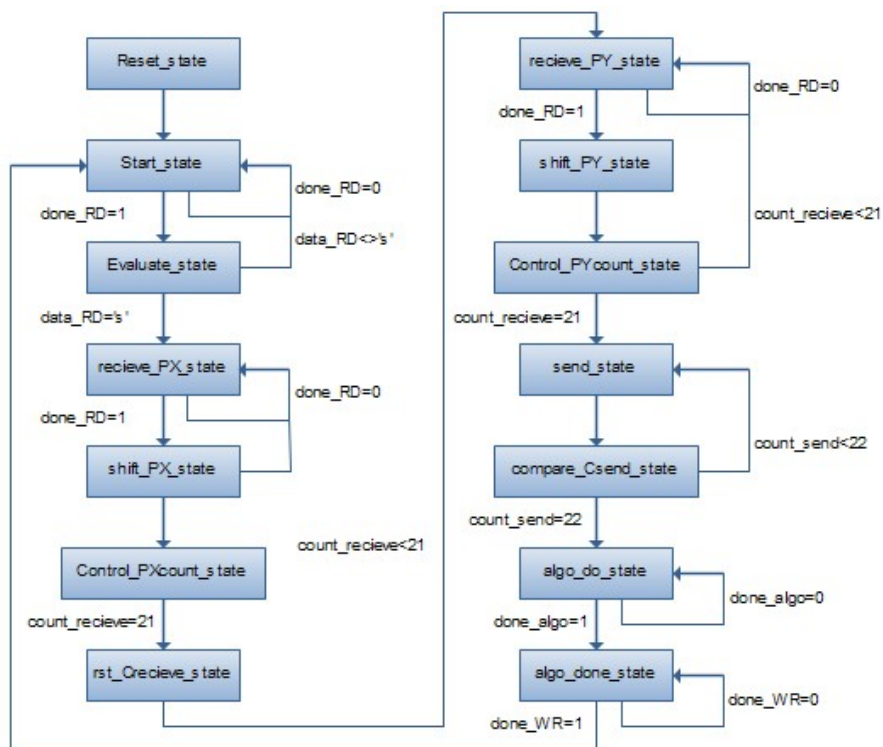
Als alle bytes ontvangen zijn, gaat hij deze per 16-bits doorsturen naar de cryptografische FPGA. Nadat de X- en Y-waarde van het punt doorgestuurd zijn, geeft de controle FPGA een startsignaal aan de cryptografische FPGA.

De FSM blijft wachten in de algo_do.state tot hij een done-sigitaal van de cryptografische

FPGA krijgt. Daarna zendt de FPGA een byte die een 'd' voorstelt terug naar de pc om aan te geven dat het algoritme klaar is.

De FPGA gaat daarna terug naar de start_state waar hij wacht op een nieuw startsignaal van de pc.

De programmacode van deze FSM is terug te vinden op de CD.



Figuur 7.6: Flowchart van de FSM in de controle FPGA voor het ECC-algoritme

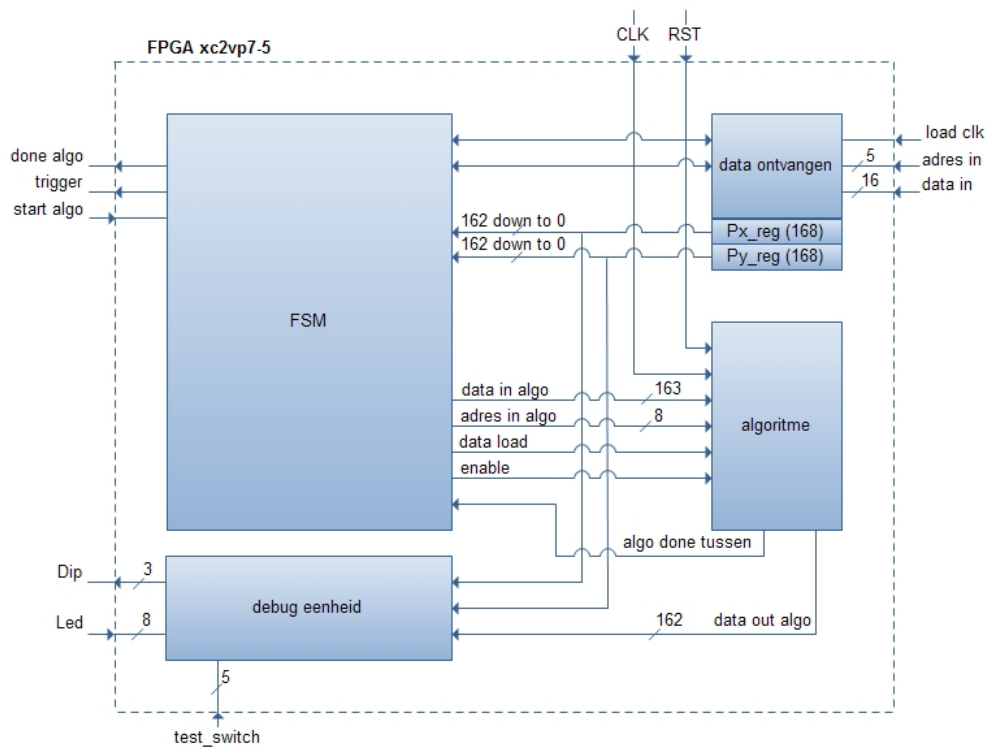
7.4.2 VHDL-wrapper in cryptografische-FPGA

In sectie 7.4.1 beschreven we de VHDL-wrapper in de controle-FPGA. In deze sectie beschrijven we de VHDL-wrapper in de cryptografische FPGA.

Algemene opbouw

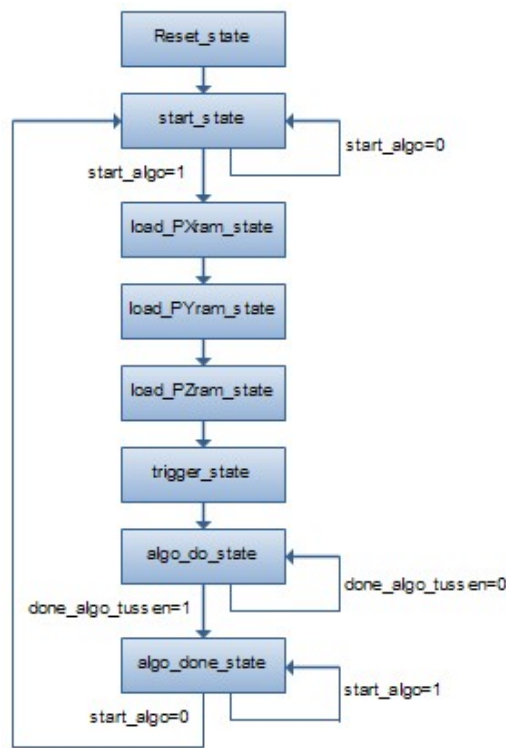
Figuur 7.7 geeft schematisch weer welke blokken we in de cryptografische FPGA implementeerde om data te zenden/ontvangen naar/van de controle FPGA.

Alles is opgebouwd rondom een FSM die beschreven staat in Figuur 7.8



Figuur 7.7: Algemeen blokschema van het programma in de cryptografische FPGA voor het ECC-algoritme

FSM-flowchart



Figuur 7.8: Flowchart van de FSM in de cryptografische FPGA voor het ECC-algoritme

Figuur 7.8 geeft schematisch de werking van de FSM in de cryptografische FPGA weer. Deze FSM in de cryptografische FPGA begint altijd in de `reset_state`. Daarna gaat hij over naar de `start_state`. Hier zal hij wachten op een startsignaal dat afkomstig is van de controle FPGA.

Eens dit startsignaal gegeven is, gaat de FPGA de X-waarde en Y-waarde van het punt in het geheugen van het algoritme plaatsen. Voor de Z-waarde van het punt wordt altijd 1 gekozen.

De FPGA gaat het cryptografisch algoritme starten zodra alles ingeladen is. Daarna wacht hij tot dit algoritme uitgevoerd is. Als het algoritme klaar is, wordt het done-signaal voor de controle FPGA hoog gemaakt en gaat de FSM terug naar de `start_state`.

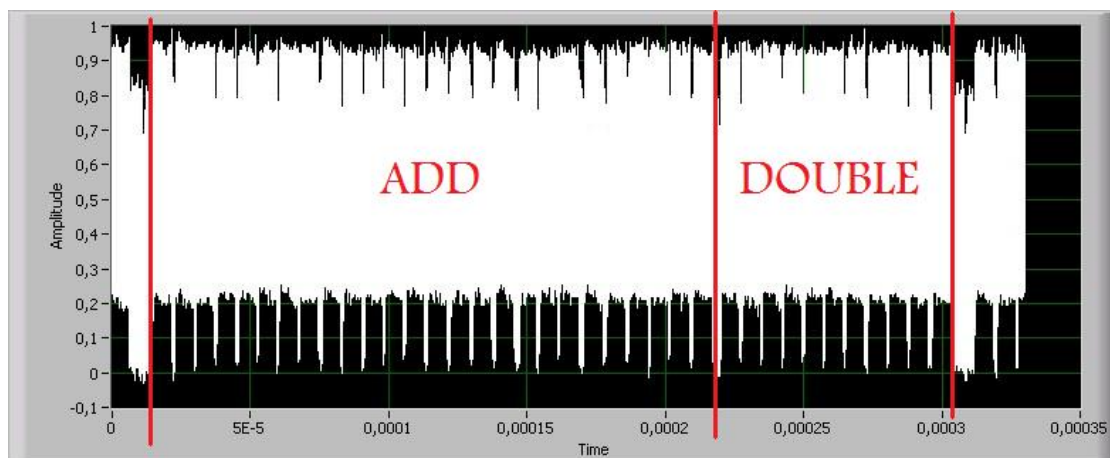
De programmacode van deze FSM is terug te vinden op de CD.

7.4.3 Resultaat

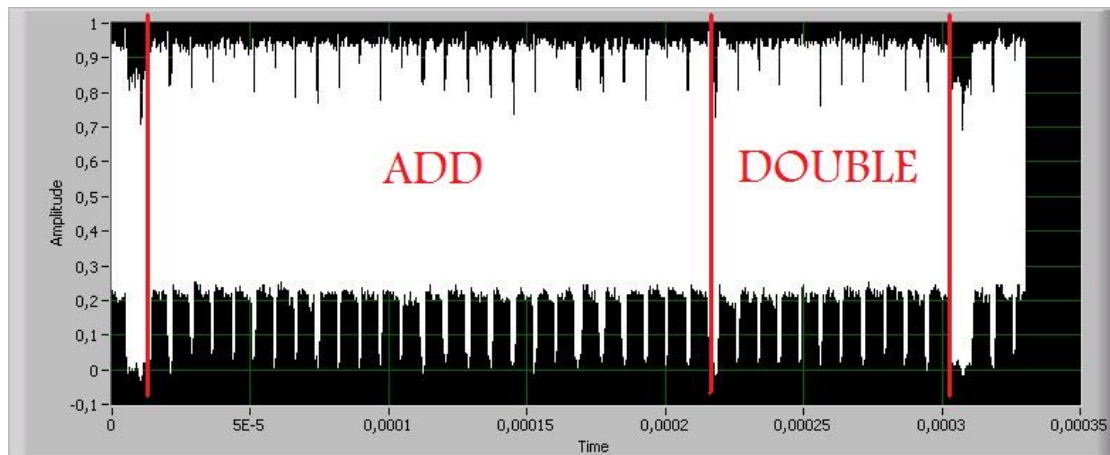
We weten dat het volledige algoritme bestaat uit 163 puntoptellingen, die telkens gevolgd worden door een puntverdubbelingen. Het is mogelijk om het vermogenverbruik op te delen in 163 stukken waarbij elk stuk één cyclus voorstelt.

Zo een cyclus bevat één puntoptelling en één puntverdubbeling. Afhankelijk van de waarde van de sleutelbit zal het resultaat van puntoptelling en dat van puntverdubbeling in andere registers opgeslagen worden. Normaal is het niet zichtbaar naar welk register geschreven wordt.

We hebben de eerste tien cycli bekeken. Figuur 7.9 geeft het vermogenverbruik van de vierde cyclus weer. Figuur 7.10 geeft het vermogenverbruik van de achtste cyclus weer. De reden waarom we juist deze gekozen hebben volgt later.



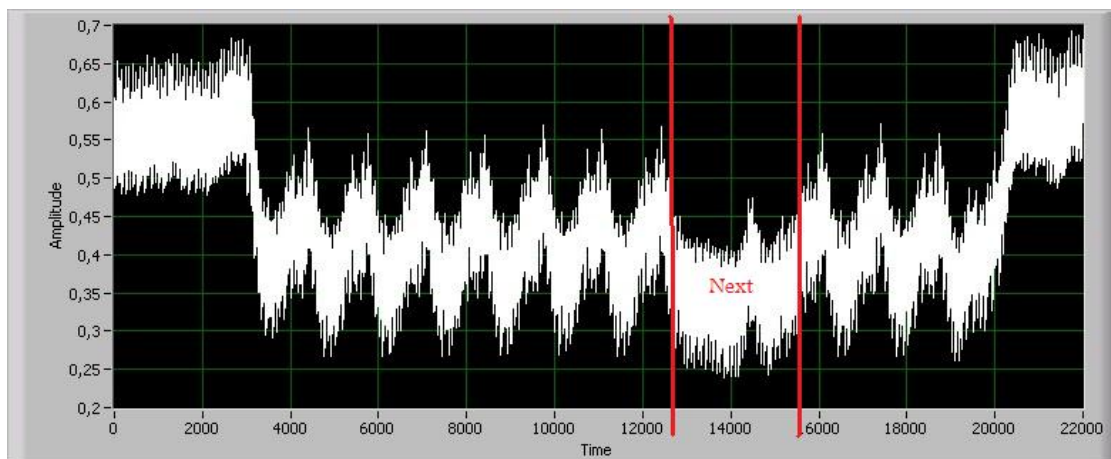
Figuur 7.9: Vermogenverbruik van de cryptografische FPGA tijdens de vierde cyclus



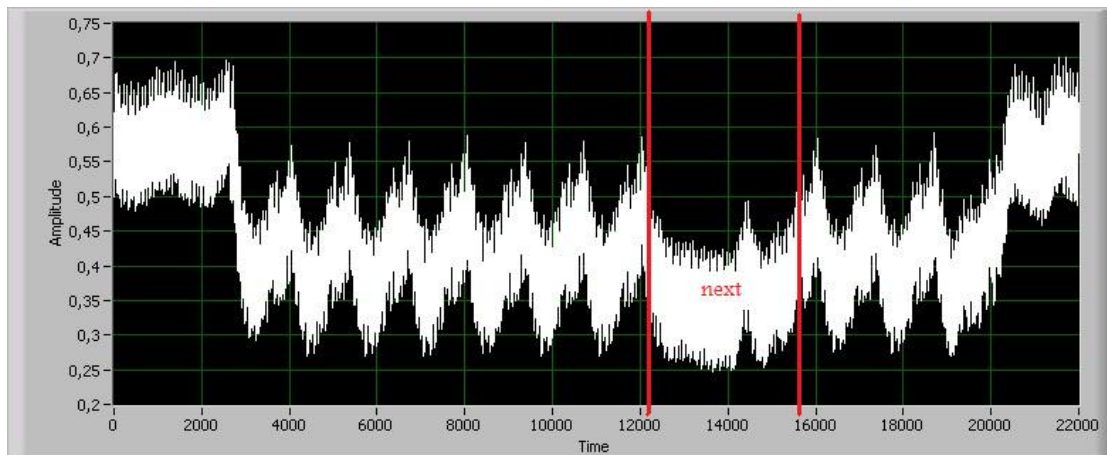
Figuur 7.10: Vermogenverbruik van de cryptografische FPGA tijdens de achtste cyclus

Op het eerste zicht is tussen bovenstaande figuren geen verschil. Daarom zijn we gaan inzoomen op de vermogendrop na de double. In dit interval gaat het algoritme een beslissing nemen naar welke registers de FPGA moet schrijven tijdens de volgende cyclus. Deze beslissing gebeurt aan de hand van de volgende sleutelbit.

Figuur 7.11 en Figuur 7.12 bevatten het vermogenverbruik van de cryptografische FPGA tijdens de vermogendrop na de vierde en achtste cyclus.



Figuur 7.11: Vermogenverbruik van de cryptografische FPGA tijdens de vermogendrop na de vierde cyclus

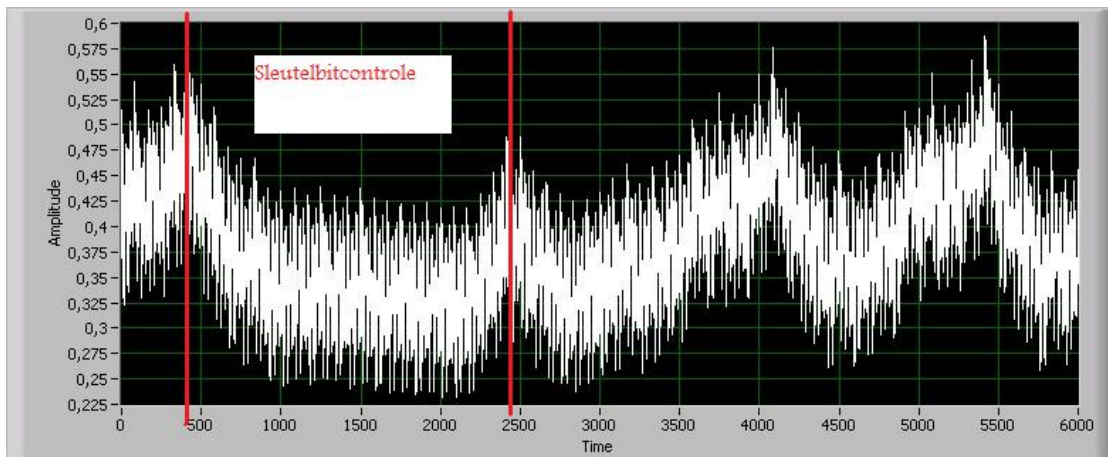


Figuur 7.12: Vermogenverbruik van de cryptografische FPGA tijdens de vermogendrop na de achtste cyclus

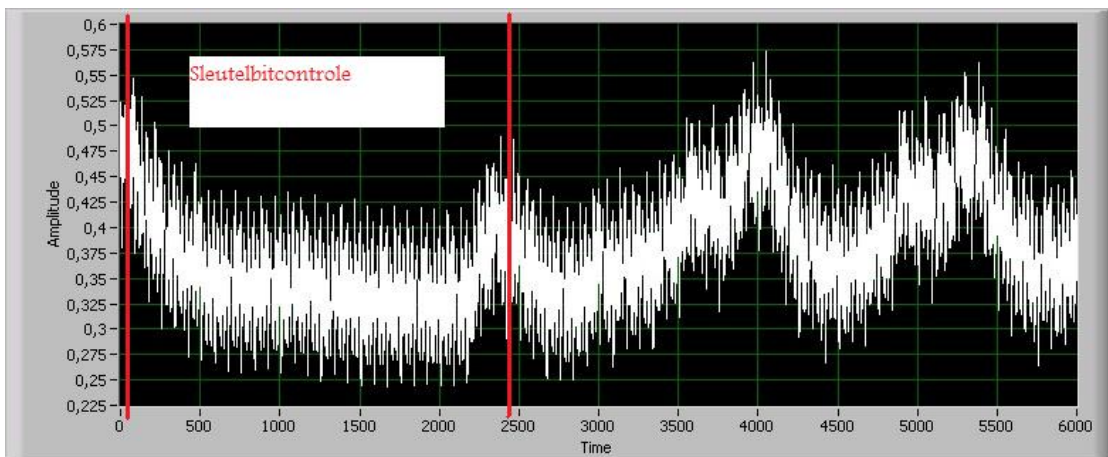
We zien nog altijd geen verschil en kunnen dus nog verder inzoomen op de onregelmatigheid in dit interval. We veronderstellen dat op die plaats het algoritme de beslissing neemt welke registers er gebruikt worden voor de volgende cyclus. De piek hierin, stelt hoogstwaarschijnlijk een shift van de sleutel voor.

Al deze informatie hebben we afgeleid uit het programmaverloop van het algoritme. Dit staat beschreven in de bijlage van de thesis van N. Cornelissen & C. Peeters.

Figuur 7.13 en Figuur 7.14 geven de uitvergroting van het zojuist beschreven interval. Dit is ook weer het vermogenverbruik van de cryptografische FPGA net na de vierde en achtste cyclus.



Figuur 7.13: Vermogenverbruik van de cryptografische FPGA tijdens de onregelmatigheid net na de vierde cyclus



Figuur 7.14: Vermogenverbruik van de cryptografische FPGA tijdens de onregelmatigheid net na de achtste cyclus

In dit interval zien we wél degelijk een verschil. De tijd tussen de rode lijnen op de figuren hierboven is langer voor de ene cyclus dan voor de andere cyclus.

Dit is de reden waarvoor we gekozen hadden voor de vierde en achtste klokpuls. De eerste zeven cycli zien eruit als de vierde cyclus. De achtste verschilde hiervan.

We wisten op voorhand wat de sleutel was. Omdat de sleutel begon met 0x01... konden we hieruit afleiden dat de kortste tijd in de figuur een nul voorstelt en de langste tijd een

één. Wanneer we nu alle 163 intervallen aflopen en telkens inzoomen op de stukken zoals hierboven, kunnen we de sleutel toch aflezen uit het vermogenverbruik.

Ondanks het gebruik van een algoritme dat veel bescherming zou kunnen bieden tegen SPA-aanvallen, zorgt een slordige implementatie ervoor dat dit voordeel verloren gaat. Waarom er een verschil is in vermogenverbruik en wat de oplossing is, halen we aan in volgende sectie.

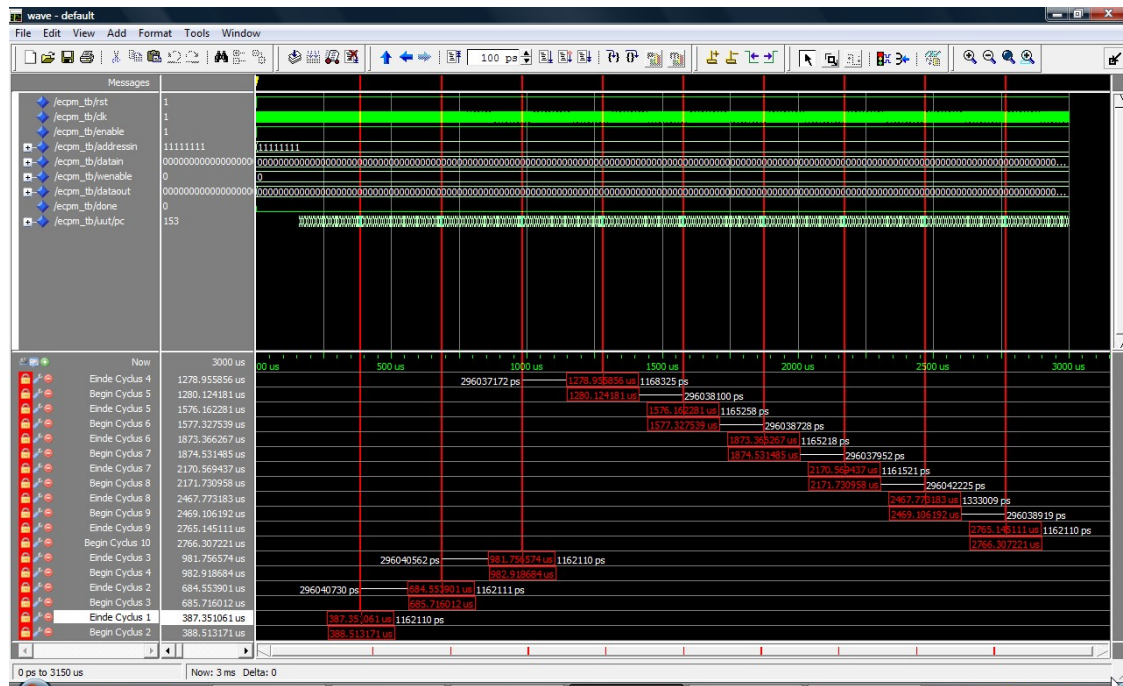
7.5 Maatregelen

In deze sectie gaan we kijken hoe het komt dat we de sleutel uit het vermogenverbruik kunnen halen en hoe we dit kunnen oplossen.

Om te kunnen antwoorden op de vraag 'hoe?' hebben we een simulatie uitgevoerd. Deze simulatie startte een encryptie met een bekende sleutel en klaartekst. We hebben enkel de acht eerste cycli bekeken.

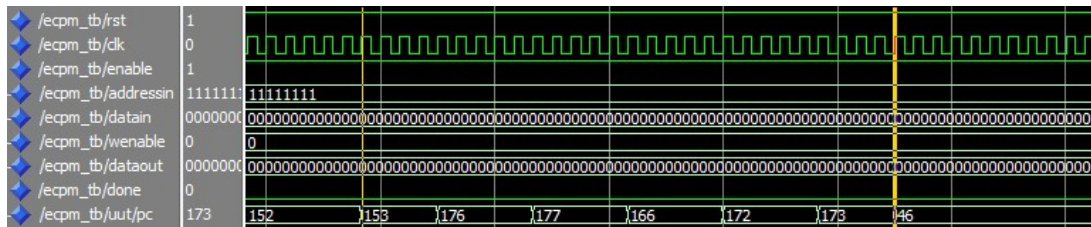
Figuur 7.15 geeft een algemeen resultaat van deze simulatie weer. De rode lijnen zijn markers die staan op de plaats waar de overgang van de ene cyclus naar de andere wordt gemaakt.

SPA-aanval op een ECC-algoritme gebaseerd op Edwards-krommen

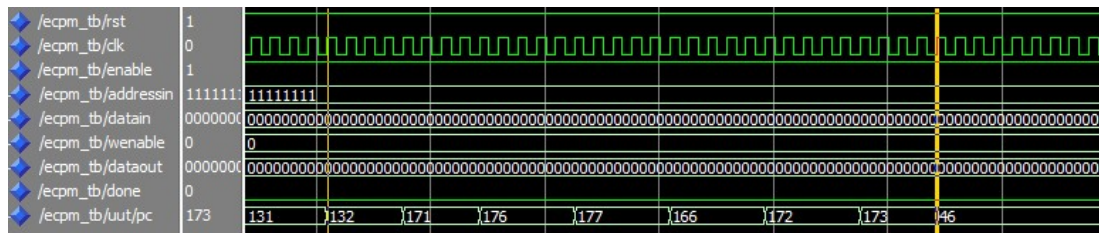


Figuur 7.15: Algemeen beeld van de simulatie van de eerste acht cycli van het ECC-algoritme

Als we inzoomen op het einde van de vierde en achtste cyclus, zien we een verschil in uitvoeringstijd. In Figuur 7.16 en Figuur 7.17 is dit verschil te zien.



Figuur 7.16: Uitvoeringstijd van de nodige instructies na de vierde cyclus



Figuur 7.17: Uitvoeringstijd van de nodige instructies na de achtste cyclus

Dit verschil in uitvoeringstijd ontstaat doordat er een extra instructie wordt uitgevoerd als de sleutelbit één is. In de programmacode uit de bijlage van de thesis van N. Cornelissen & C. Peeters kunnen we halen welke programmaflow er gevolgd wordt, indien de sleutelbit een één of een nul is. Tabel 7.2 en Tabel 7.3 geven de programmaflow voor beide gevallen.

Tabel 7.2: Programmaflow als de sleutelbit één is

Instructie	Adres	Voorwaarde
...	...	
cjmp	172	sleutelbit = '0'
storePC	169	
jmp	1	
storePC	171	
jmp	112	
jmp	176	
...	...	

Tabel 7.3: Programmaflow als de sleutelbit nul is

Instructie	Adres	Voorwaarde
...	...	
cjmp	172	sleutelbit = '0'
storePC	174	
jmp	46	
storePC	176	
jmp	133	
...	...	

Als de sleutelbit één is, maakt het programma een extra jump. Door deze extra jump zal het programma een extra vertraging hebben als de sleutelbit één is. Dit probleem is gemakkelijk op te lossen door een extra jump te plaatsten als de sleutelbit nul is. Deze jump is dan een 'dummy' jump die gewoon naar de volgende regel springt. In Tabel 7.4 staat de verbeterde programmaflow als de sleutelbit één is. Tabel 7.5 bevat de verbeterde programmaflow als de sleutelbit nul is.

Tabel 7.4: Programmaflow als de sleutelbit één is na verbetering

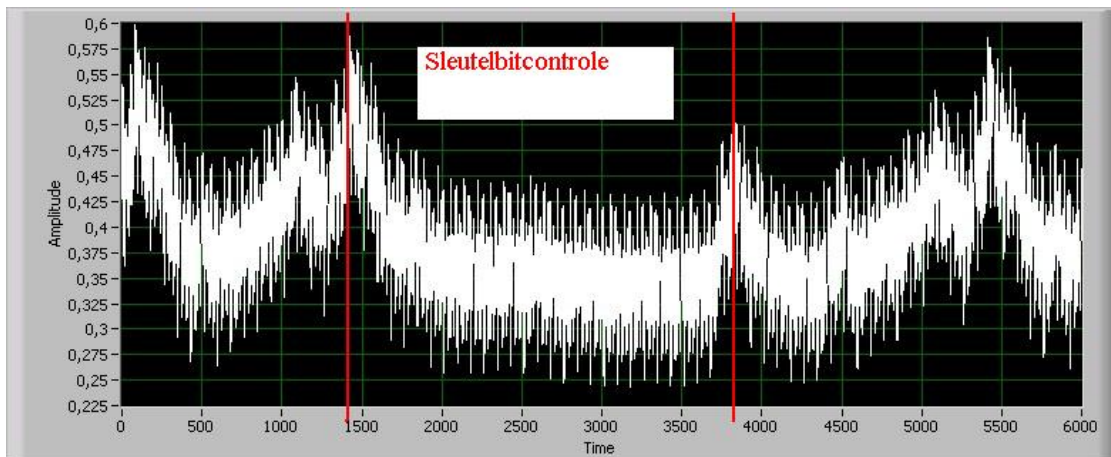
Instructie	Adres	Voorwaarde
...	...	
cjmp	172	sleutelbit = '0'
storePC	169	
jmp	1	
storePC	171	
jmp	112	
jmp	177	
...	...	

Tabel 7.5: Programmaflow als de sleutelbit nul is na verbetering

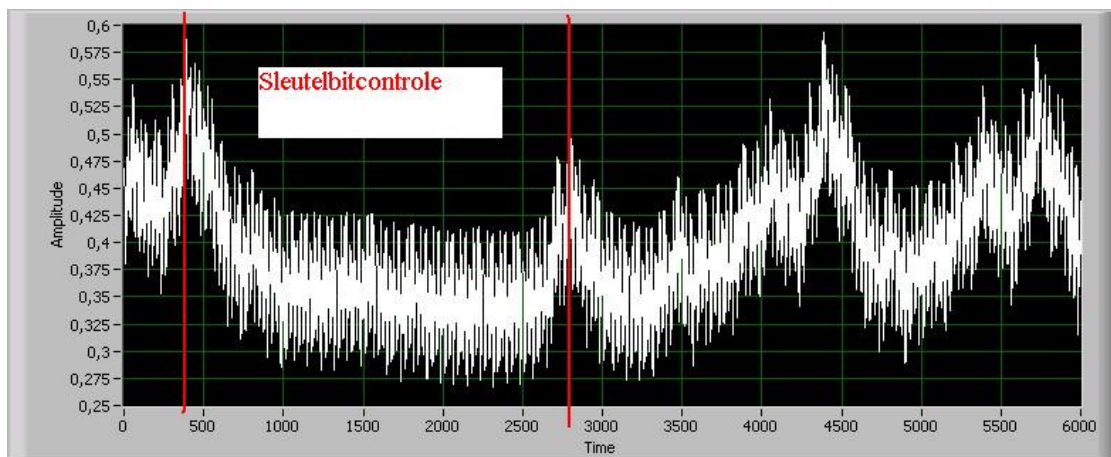
Instructie	Adres	Voorwaarde
...	...	
cjmp	172	sleutelbit = '0'
storePC	174	
jmp	46	
storePC	176	
jmp	133	
jmp	177	
...	...	

7.5.1 Resultaat

Voordat we een vermogenmeting deden op het aangepaste programma, hebben we eerst de simulatie opnieuw uitgevoerd. Uit de simulatie kunnen we opmaken dat de uitvoeringstijd voor beide waarden van de sleutelbit nu wél hetzelfde is. Figuur 7.18 is de



Figuur 7.20: Vermogenverbruik van de cryptografische FPGA na de vierde cyclus



Figuur 7.21: Vermogenverbruik van de cryptografische FPGA na de achtste cyclus

Door de extra jump toe te voegen zijn we niet meer in staat om de sleutel uit het vermogenverbruik af te leiden met een SPA-aanval.

Hoofdstuk 8

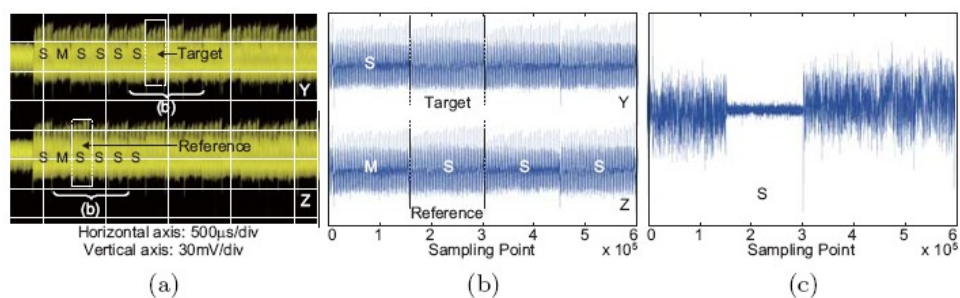
CPA-aanval op een ECC-algoritme gebaseerd op Edwards-krommen

8.1 Inleiding

In 2008 publiceerde 'The International Association for Cryptologic Research' een paper [31] over een nieuwe vermogenaanval. Deze aanval noemde ze de Comparative Power Analysis of CPA. Wetenschappers ontwikkelden deze aanval voor cryptografische systemen die gebaseerd zijn op modulaire machtsverheffing.

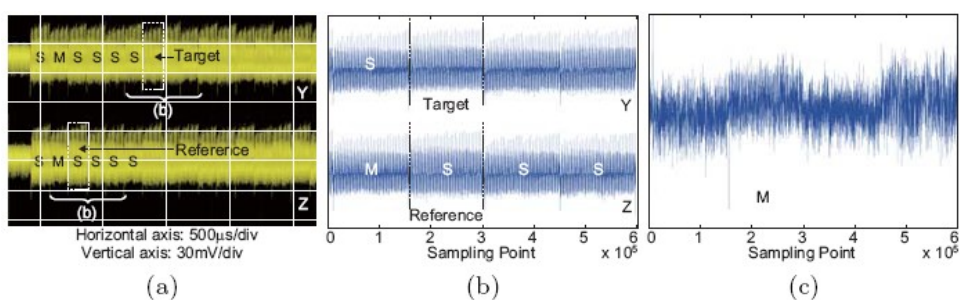
Bij deze aanval gaan we ervan uit dat een bewerking op dezelfde getallen hetzelfde vermogenverbruik heeft. Als deze bewerkingen in de tijd zijn verschoven, kunnen we de twee bewerkingen met elkaar vergelijken. De aftrekking van deze tijdsintervallen geeft nul indien het resultaat van de bewerkingen gelijk zijn. Als het resultaat van de bewerkingen niet hetzelfde is, zal de aftrekking niet nul geven.

Als we de sleutelbit van een interval kennen, kunnen we de sleutelbit van het andere interval hieruit afleiden. Figuur 8.1 en Figuur 8.2 geven het principe van de aanval weer. Let op: om de aanval te doen, moeten we twee inputgetallen kiezen die aan elkaar gerelateerd zijn.



Figuur 8.1: Principe van de CPA-aanval uit [31]: dezelfde bewerkingen

In Figuur 8.1 is a) het vermogenverloop van de bewerkingen op twee inputgetallen, b) de zoom van de twee bewerkingen die we willen vergelijken en c) het verschil van de twee bewerkingen.



Figuur 8.2: Principe van de CPA-aanval uit [31]: verschillende bewerkingen

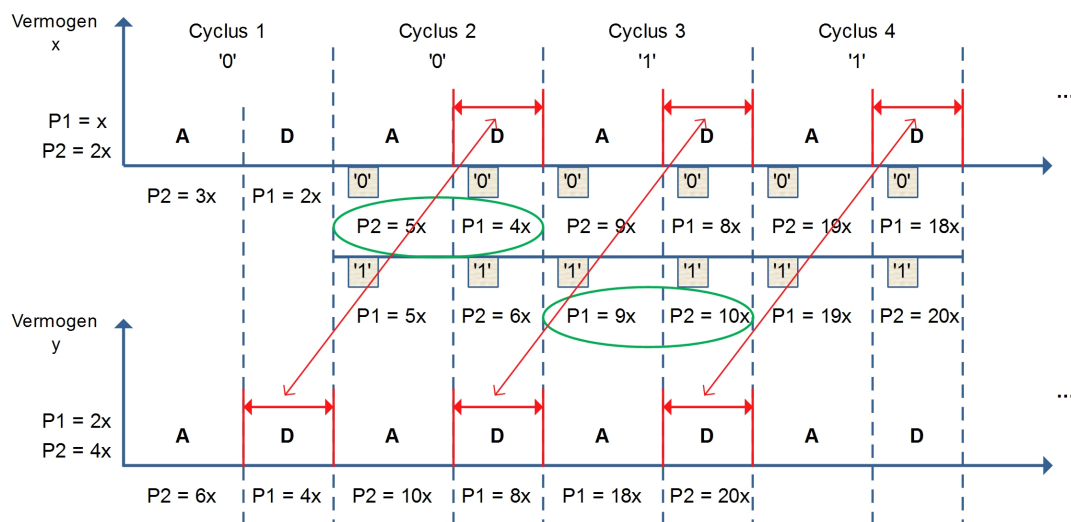
In Figuur 8.2 is a) het vermogenverloop van de bewerkingen op twee inputgetallen, b) de zoom van de twee bewerkingen die we willen vergelijken en c) het verschil van de twee bewerkingen.

8.2 Modulaire machtsverheffing VS. Modulaire optelling/verdubbeling

Zoals in 8.1 vermeld, is de CPA-aanval ontworpen voor algoritmes die gebruik maken van modulaire machtsverheffingen. Om de aanval te gebruiken voor het ECC-algoritme van N. Cornelissen & C. Peeters [25] moeten we de aanval aanpassen zodat deze ook bruikbaar wordt voor modulaire optellingen/verdubbelingen.

Figuur 8.3 is een schematische voorstelling van het aangepaste algoritme. De werkwijze om de aanval te doen, is iets anders dan de werkwijze beschreven in de paper [31]:

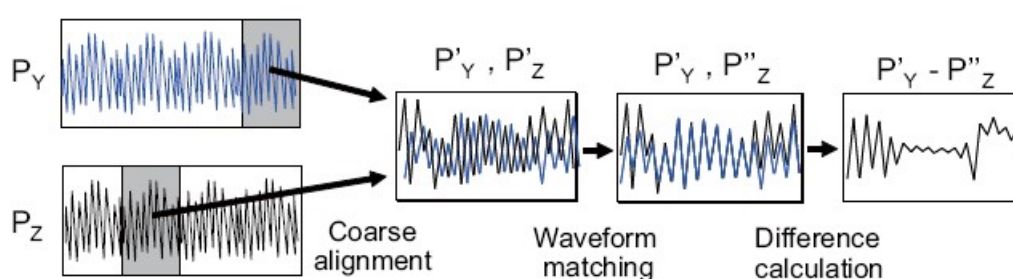
- Bij de CPA-aanval op een algoritme gebaseerd op modulaire machtsverheffing, kijken we naar het volledige interval waar een sleutelafhankelijke bewerking gebeurt. Bij het ECC-algoritme doet de FPGA een modulaire optelling gevolgd door een modulaire verdubbeling per sleutelbit. We kijken enkel naar de puntverdubbelingen;
- We gebruiken twee ingangspunten. Het eerste punt is een gekozen punt op de elliptische kromme. Het tweede punt is het dubbele van dit punt en is ook op de elliptische kromme gelegen;
- We weten dat de eerste sleutelbit altijd één is. We kunnen hierdoor de beginsituatie berekenen. We veronderstellen dat de tweede bit nul is. Als na evaluatie van alle sleutelbits blijkt dat deze veronderstelling fout was, moeten we de geschatte sleutelbit inverteren.
- We meten cyclus x van het dubbele punt en cyclus $x+1$ van het enkele punt. Wanneer het vermogenverbruik van beide metingen gelijk is, weten we dat de sleutelbit waarmee de FPGA cyclus $x+1$ versleutelde gelijk is aan de sleutelbit waarmee de FPGA cyclus x versleutelde.



Figuur 8.3: Principe van de CPA-aanval op het ECC-algoritme van N. Cornelissen & C. Peeters [25]

8.3 Matching van het vermogenverbruik van de twee metingen

Het spreekt voor zich dat we de opname van de twee vermogenverbruiken perfect moeten matchen voordat we ze kunnen aftrekken. In paper [31] gebruiken de wetenschappers hiervoor een techniek die ze 'phase-based waveform matching' noemen. Deze techniek staat ook beschreven in een paper [32] die 'The International Association for Cryptologic Research' publiceerde. Figuur 8.4 toont de stappen die we moeten volgen bij het berekenen van het verschil tussen de twee waveforms.



Figuur 8.4: Stappen voor waveform-matching uit [31]

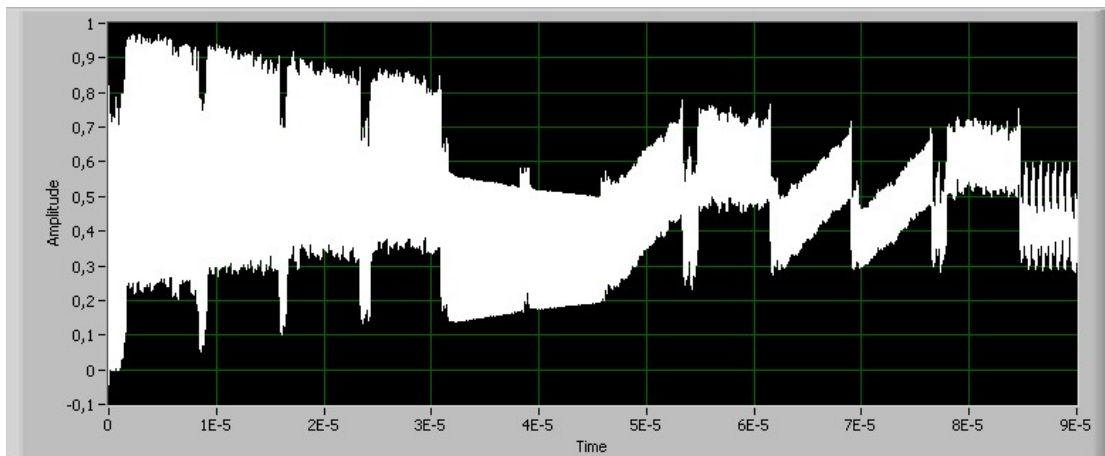
Phase-based waveform matching is een zeer goede maar ook zeer ingewikkelde manier van waveform matching. Met phase-based waveform matching is het mogelijk om zelfs waveforms te matchen binnen een sample-interval. Deze nauwkeurigheid is handig maar niet noodzakelijk volgens ons. Het matchen van de waveforms op het sample-interval is genoeg om verschil te zien tussen twee dezelfde en twee verschillende bewerkingen.

Wij maken gebruik van de correlatiecoëfficiënt tussen de twee waveforms om ze te matchen. Daar waar de hoogste correlatie is, matchen de twee het beste. Aan de hand van deze informatie verschuiven we de waveforms ten opzichte van elkaar. Daarna nemen we het verschil van de twee.

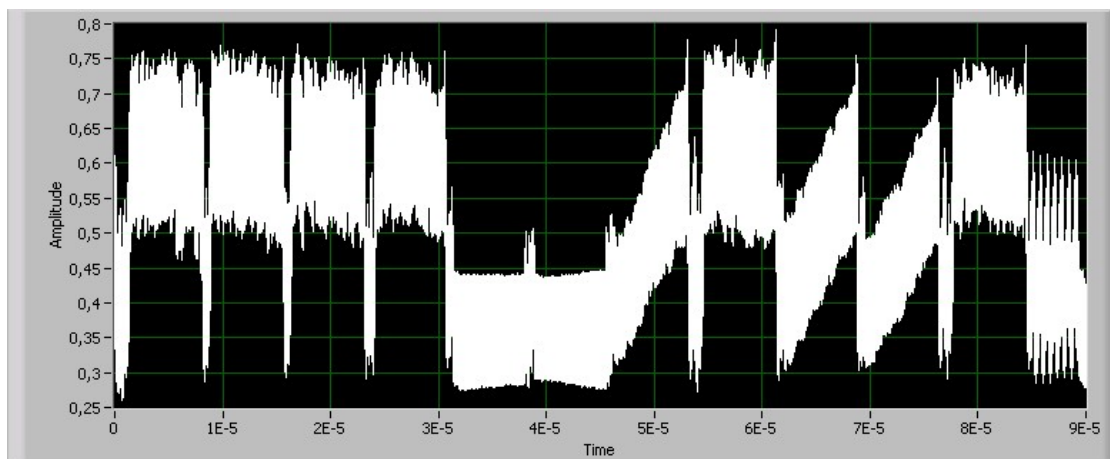
8.4 Resultaat

De aanval die we uitvoerden op het ECC-algoritme deden we met twee punten op de Edwards-kromme. Punten op de kromme worden eerst omgezet naar standaard projectieve coördinaten, voor redenen uitgelegd in het eindwerk van N. Cornelissen & C.

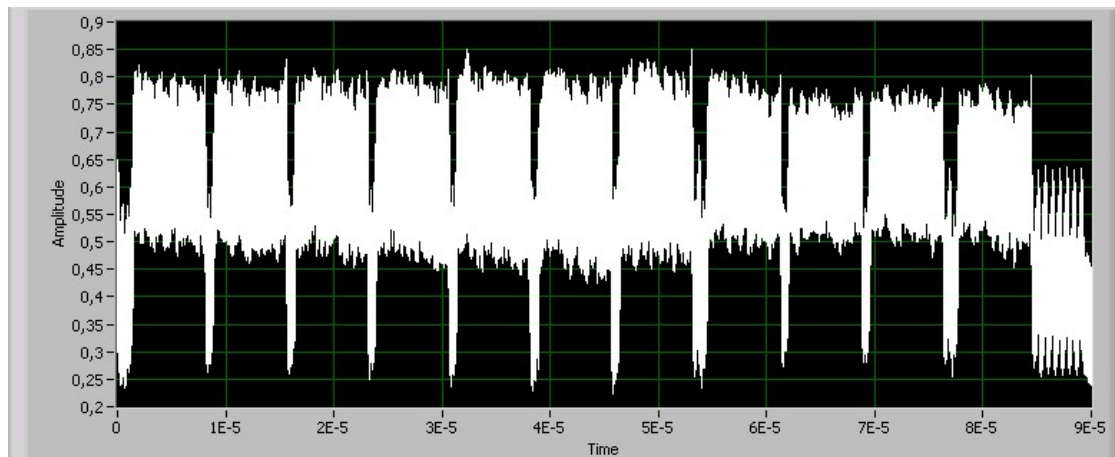
Als de Z-waarde van het ingangspunt één is, is dit zeer goed zichtbaar in het vermogenverloop. In Figuur 8.5 is het vermogenverloop van de puntverdubbeling in de initialisatie zichtbaar. Figuur 8.6 toont het vermogenverloop van de puntverdubbeling bij de evaluatie van de eerste sleutelbit als die nul is, terwijl Figuur 8.7 dit vermogenverloop toont als de sleutelbit één is.



Figuur 8.5: Vermogenverloop van de puntverdubbeling tijdens de initialisatie



Figuur 8.6: Vermogenverloop van de puntverdubbeling tijdens de evaluatie van de eerste sleutelbit als die nul is.

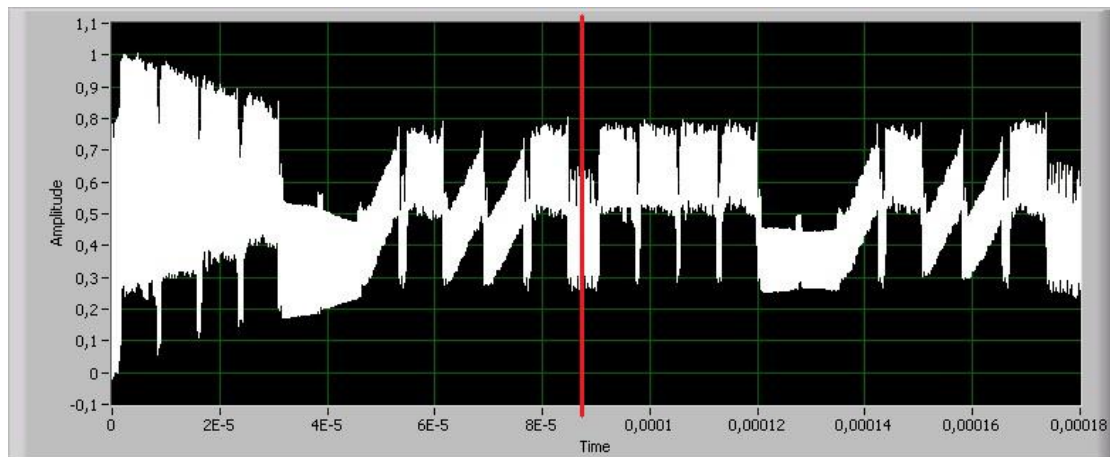


Figuur 8.7: Vermogenverloop van de puntverdubbeling tijdens de evaluatie van de eerste sleutelbit als die één is.

We zien dat het vermogenverloop van Figuur 8.5 en Figuur 8.6 ongeveer hetzelfde is. Het vermogenverloop van Figuur 8.5 en Figuur 8.7 zijn totaal niet hetzelfde.

Dat het vermogenverloop van Figuur 8.5 en Figuur 8.6 niet volledig gelijk zijn is te wijten aan een overgangsverschijnsel bij het starten. Dit overgangsverschijnsel treedt op door spoelen op het SASEBO-bord.

Om aan te tonen dat dit wel degelijk een overgangsverschijnsel is, hebben we twee keer achter elkaar de initialisatie uitgevoerd. Deze meting staat in Figuur 8.8 weergegeven. Wanneer we de tweede initialisatie vergelijken met Figuur 8.6 zien we dat deze identiek zijn.



Figuur 8.8: Overgangsverschijnsel bij het starten van het algoritme

De oplossing

Het ECC-algoritme van N. Cornelissen & C. Peeters werkt met standaard projectieve coördinaten (X, Y, Z) omdat op deze manier een puntvermenigvuldiging sneller uitgevoerd kan worden. Waarom dat dit zo is staat beschreven in de doctoraatsthesis van N. Mentens [16]

De berichten worden voorgesteld als punten op de elliptische krommen, in een affiene coördinaten stelsel. De omzetting naar standaard projectieve coördinaten kan met behulp van onderstaande formules:

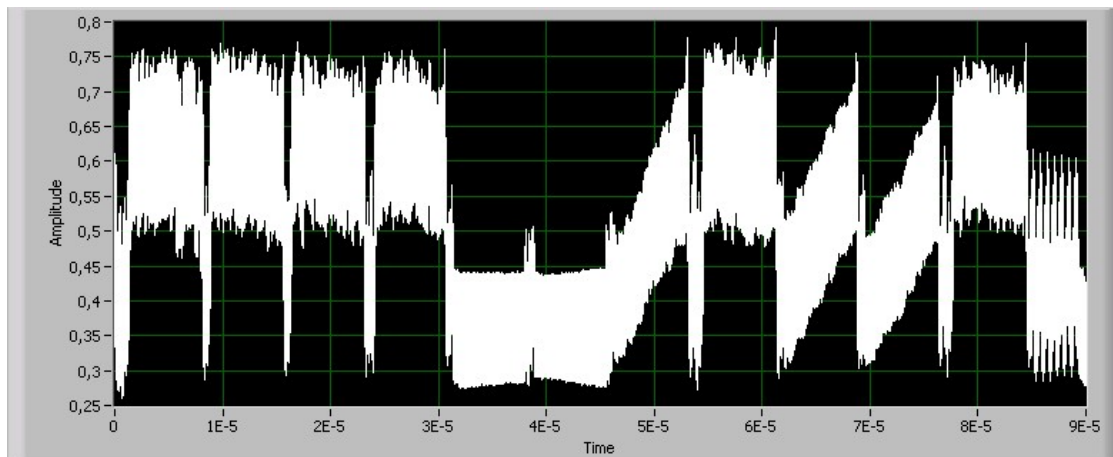
$$X = x.Z$$

$$Y = y.Z$$

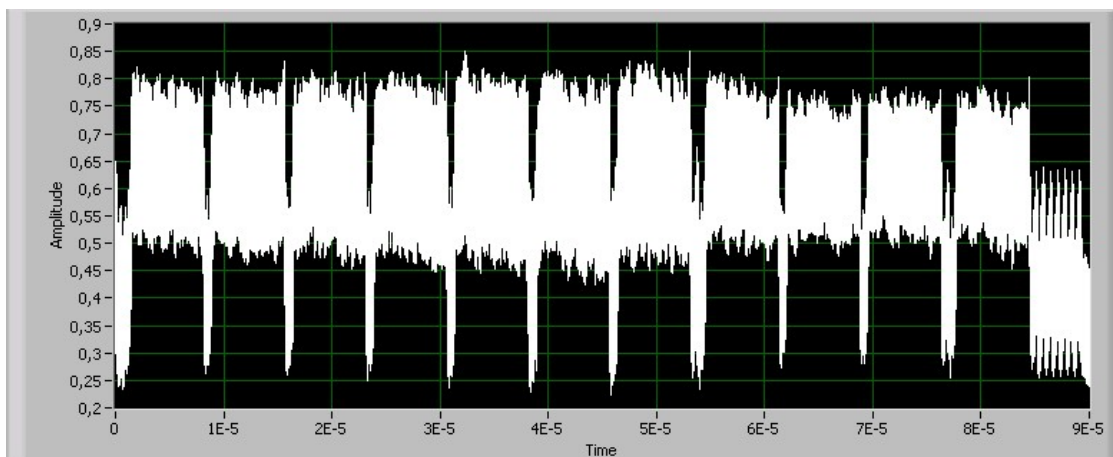
Op deze manier kunnen we Z kiezen en de overeenkomstige X en Y berekenen. Door Z verschillend van één te kiezen, is het verschil in vermogenverbruik tussen de initialisatie en de evaluatie van de eerste sleutelbit minder zichtbaar. Het is nog wel mogelijk om met een CPA-aanval het verschil te zien.

Het probleem is niet volledig verholpen, maar het is moeilijker geworden. Vandaar dat we de raad geven om altijd Z -waardes te kiezen die verschillend zijn van één.

In Figuur 8.9 is het vermogenverbruik van de initialisatie weergegeven als Z gelijk is aan één. In Figuur 8.10 is het vermogenverbruik van de initialisatie weergegeven als Z verschillend is van één.



Figuur 8.9: Vermogenverloop van de initialisatie als Z gelijk is aan één



Figuur 8.10: Vermogenverloop van de initialisatie als Z verschillend is van één

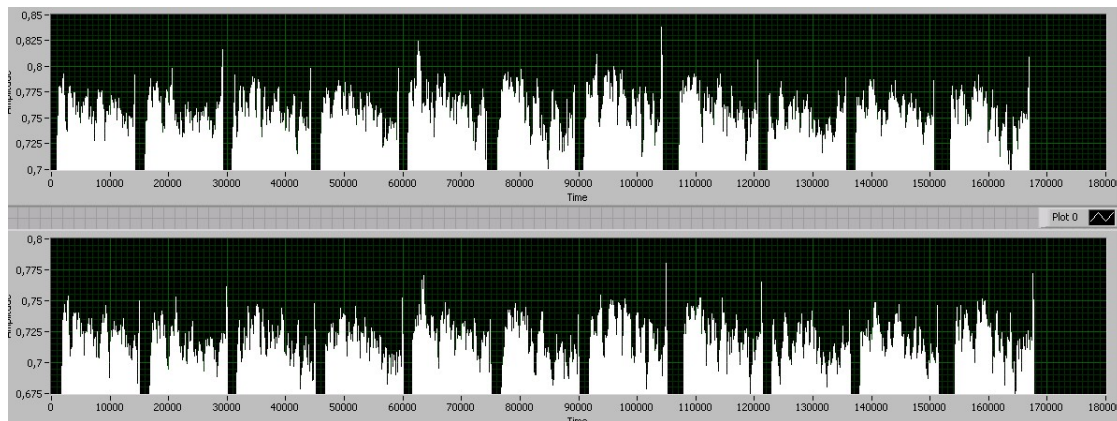
8.4.2 Onthullen van de andere sleutelbits

De andere sleutelbits kunnen we onthullen zoals beschreven in 8.2. Figuur 8.11 toont bovenaan het vermogenverloop van de puntverdubbeling van de eerste cyclus. Dit vermogenverloop is opgenomen met het dubbele ingangspunt. Figuur 8.11 toont onderaan het vermogenverloop van de puntverdubbeling van de tweede cyclus. Dit vermogenverloop is opgenomen met het enkele ingangspunt. Bij Figuur 8.11 hebben we ingezoomd op de toppen van het vermogenverbruik.

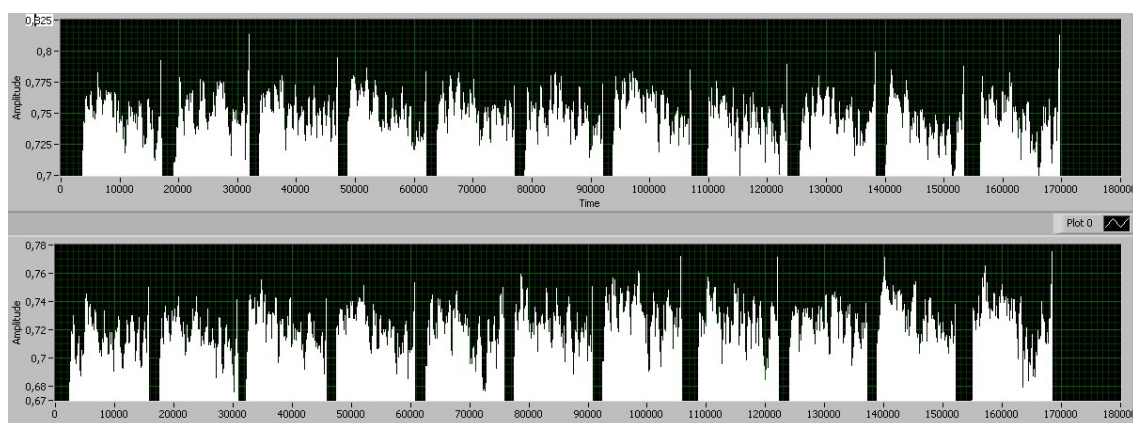
Vermits de sleutelbit van de eerste en tweede cyclus niet verschillen, zal het vermogenverbruik gelijk zijn. De verschillen tussen beide zijn miniem.

Figuur 8.12 toont bovenaan het vermogenverloop van de puntverdubbeling van de tweede cyclus. Dit vermogenverloop is opgenomen met het dubbele ingangspunt. Figuur 8.12 toont onderaan het vermogenverloop van de puntverdubbeling van de derde cyclus. Dit vermogenverloop is opgenomen met het enkele ingangspunt. Bij Figuur 8.12 hebben we ingezoomd op de toppen van het vermogenverbruik.

Vermits de sleutelbit van de eerste en tweede cyclus verschillen, zal het vermogenverbruik niet gelijk zijn. De gelijkenissen tussen beide zijn miniem.



Figuur 8.11: Vergelijking tussen cyclus 1 (dubbel ingangspunt) en cyclus 2(enkel ingangspunt)



Figuur 8.12: Vergelijking tussen cyclus 2 (dubbel ingangspunt) en cyclus 3(enkel ingangspunt)

De gelijkenissen tussen de twee cycli van Figuur 8.11 zijn goed zichtbaar. De verschillen tussen de twee cycli van Figuur 8.12 zijn ook goed zichtbaar. Toch gaf onze uitgedacht waveform matching uit 8.3 slechte resultaten.

We hebben deze matching helaas niet meer kunnen bekijken wegens tijdsgebrek. Wij raden aan om toch de phase-based waveform matching uit [32] te implementeren. Wij verwachten dat het resultaat hierdoor zal verbeteren.

De vermogenpatronen uit Figuur 8.11 en Figuur 8.12 zijn uitgemiddeld over 2000 metingen. De gelijkenis tussen twee verschillende cycli met dezelfde sleutelbit en dubbel/enkel ingangspunt zal nog spectaculairder worden door uit te middelen over 10000 metingen. Ook hiervoor hebben we jammer genoeg te weinig tijd gehad.

Hoofdstuk 9

Besluit

Het doel van dit eindwerk was een aantal vermogenaanvallen uit te voeren op een bestaande implementatie van een elliptische kromme algoritme. Indien deze implementatie op FPGA niet bestand bleek tegen deze vermogenaanvallen, moesten we de implementatie herzien.

Om de vermogenaanvallen te doen was een meetopstelling nodig die binnen de KHLim niet beschikbaar was. Deze vermogenmeetopstelling moesten wij nog ontwerpen. Enkele belangrijke factoren waren de stabiliteit en de nauwkeurigheid. Verder moest de meetopstelling geautomatiseerd zijn, en moest het mogelijk zijn om meerdere metingen achter elkaar uit te voeren en om de resultaten op te slaan.

Tijdens het eerste semester ontworpen we de meetopstelling. Het gebruiksgemak van de opstelling was voor ons een belangrijk punt in het ontwerp. Uit een aantal testprocedures bleek dat de meetopstelling de gewenste nauwkeurigheid en stabiliteit bezat én ook gemakkelijk was in gebruik.

Tijdens het tweede semester deden we de vermogenaanvallen op de bestaande implementatie van het elliptische kromme algoritme. We deden twee vermogenaanvallen: een SPA-aanval en een CBA-aanval.

De eerste aanval die we deden was de SPA-aanval. Het algoritme bleek hiertegen niet voldoende bestand te zijn. De volgende stap die we uitvoerden was een verbetering van de implementatie. Een aantal wijzigingen aan deze implementatie zorgde ervoor dat de implementatie wél bestand werd tegen de SPA-aanval.

De tweede aanval die we deden was de CBA-aanval. Door problemen met het softwarepakket waarmee we de FPGA-software schreven, hebben we heel wat tijd verloren. Naast deze problemen verliep de uitvoering van de CBA-aanval niet zo vlot als we voorzagen. Omwille van deze twee redenen hebben we de aanval niet volledig kunnen afronden. Suggesties voor de verdere uitvoering van deze aanval zijn opgenomen in hoofdstuk 8.

Als we terugkijken op deze masterproef zijn we zeer tevreden over het behaalde resultaat. Ten eerste heeft de KHLim een vermogenmeetopstelling die studenten/onderzoekers in de toekomst kunnen gebruiken voor allerlei vermogenaanvallen op cyptografische algoritmes. Ten tweede is een bestaande implementatie van een elliptische kromme algoritme beter beveiligd tegen SPA-aanvallen en is de aanzet gegeven om dit algoritme te evalueren op veiligheid tegen CBA-aanvallen.

Bibliografie

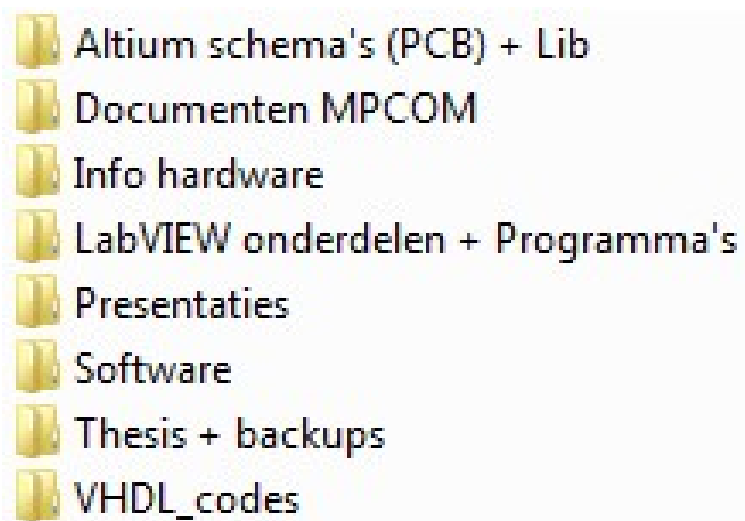
- [1] Geschiedenis van de cryptografie. Laatste raadpleging 31/4/2010. [Online beschikbaar]: http://en.wikipedia.org/wiki/History_of_cryptography
- [2] Tijdslijn van de cryptografische toestellen. Laatste raadpleging 31/4/2010. [Online beschikbaar]: <http://users.telenet.be/d.rijmenants/nl/timeline.htm>
- [3] Introduction to cryptografie. Laatste raadpleging 31/4/2010. [Online beschikbaar]: <http://www.esat.kuleuven.be/cosic/intro/>
- [4] A. Menezes, P. van Oorschot, en S. Vanstone, *Handbook of Applied Cryptography*. CRC press, 1997.
- [5] Website van het RCIS, ontwikkelaars van SASEBO. Laatste raadpleging 31/4/2010. [Online beschikbaar]: <http://www.rcis.aist.go.jp/special/SASEBO/>
- [6] S. Singh, *The codebook*. Fourth Estate Ltd, 1999.
- [7] Research Center for Information Security, “Side-channel attack standard evaluation board, SASEBO-G specifications,” 2008.
- [8] G. Johnson en R. Jennings, *LabVIEW Graphical Programming*. McGraw-Hill Professional, 2006.
- [9] How to use the parallel port in LabVIEW. Laatste raadpleging 31/4/2010. [Online beschikbaar]: <http://digital.ni.com/public.nsf/allkb/B937AC4D8664E37886257206000551CB>
- [10] Website van FTDI, ontwikkelaar van de USB naar parallele FIFO omzetter. Laatste raadpleging 08/4/2010. [Online beschikbaar]: <http://www.ftdichip.com>
- [11] Website van Agilent, fabricant van de MSO6052A. Laatste raadpleging 7/12/2009. [Online beschikbaar]: <http://www.home.agilent.com>

BIBLIOGRAFIE

- [12] Website met de geschiedenis van GPIB. Laatste raadpleging 15/12/2009. [Online beschikbaar]: <http://www.commsdesign.com/main/9803fe3.htm>
- [13] Website met de specificaties van GPIB. Laatste raadpleging 13/12/2009. [Online beschikbaar]: <http://cnx.org/content/m12283/latest/>
- [14] Website met informatie over de IEEE-standaard die GPIB gebruikt. Laatste raadpleging 8/12/2009. [Online beschikbaar]: <http://en.wikipedia.org/wiki/IEEE-488>
- [15] Tutorial over het gebruik van SCPI-commando's. Laatste raadpleging 30/01/2009. [Online beschikbaar]: http://www.jpacsoft.com/scpi_explained.htm
- [16] N. Mentens, "Secure and efficient coprocessor design for cryptographic applications on FPGAs," Ph.D. dissertatie, KULeuven, 2007.
- [17] N. Mentens, "Vermogen- en tijdsanalyse van cryptosystemen gebaseerd op elliptische krommen," Master thesis, KULeuven, 2003.
- [18] P. Buysschaert en E. D. Mulder, "Elektromagnetische analyse (ema) van een fpga implementatie van een elliptische krommen cryptosysteem," Master thesis, KULeuven, 2004.
- [19] P. Kocher, J. Jaffe, en B. Jun, *Differential Power Analysis*. Cryptography Research Inc, 1998.
- [20] National Bureau of Standards, *Data Encryption Standard*. NIST, 1977.
- [21] W. Diffie en M. E. Hellman, *New Directions in Cryptography*. IEEE Transactions on Information Theory, 1976.
- [22] National Bureau of Standards, *Advanced Encryption Standard (AES)*. NIST, 2000.
- [23] Website met informatie over RC4. Laatste raadpleging 22/04/2010. [Online beschikbaar]: <http://en.wikipedia.org/wiki/RC4>
- [24] Website met informatie over A5/1. Laatste raadpleging 22/04/2010. [Online beschikbaar]: <http://en.wikipedia.org/wiki/A5/1>
- [25] N. C. en Christian Peeters, "Compacte fpga-implementatie van een cryptografisch systeem op basis van edwards-krommen," Master thesis, KHLim, 2009.

- [26] Federal Information Processing Standards, *Digital Signature Standaard*. FIPS, 1994.
- [27] Website met informatie over hash-functies. Laatste raadpleging 22/04/2010. [Online beschikbaar]: <http://www.benjaminheek.nl/pws/index.php>
- [28] Website met informatie over de MD5 standaard. Laatste raadpleging 22/04/2010. [Online beschikbaar]: <http://nl.wikipedia.org/wiki/MD5>
- [29] Website met informatie over de SHA-1 standaard. Laatste raadpleging 22/04/2010. [Online beschikbaar]: <http://nl.wikipedia.org/wiki/SHA-familie>
- [30] SCPI Consortium, *Standard Commands for Programmable Instruments, Style and Syntax*. SCPI Consortium, 1999.
- [31] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, en A. Shamir, *Collision-based power analysis of modular exponentiation using chosen-message pairs*. International Association for Cryptologic Research, 2008.
- [32] N. Homma, T. Aoki, A. Satoh, S. Nagashima, en Y. Imai, *High-resolution side-channel attack using phase-based waveform matching*. International Association for Cryptologic Research, 2006.

Inhoud van de cd met bijlages



Figuur 1: Inhoudoverzicht van de cd