E06/ELO/09
Diepenbeek, 2006

Implementatie van hashfuncties in hardware

rapport over het eindwerk van

Joris PLESSERS en Piet VANGENECHTEN

kandidaten voor de graad van
Industrieel Ingenieur in de elektronica

Promotoren:
dr. lic. L. Batina
ir. N. Mentens

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van grafieken

Abstract

Implementatie van hashfuncties in hardware

door: Joris PLESSERS
Piet VANGENECHTEN
promotoren: ir. N. MENTENS, docent KHLim
dr. lic. L. BATINA, COSIC

Het onderwerp van onze masterproef werd uitgeschreven door COSIC (COmputer Security and Industrial Cryptography), een onderzoeksinstelling binnen de K.U.Leuven, die zich toespitst op cryptografie en netwerkbeveiliging. COSIC telt ongeveer 45 medewerkers, waarvan 8 postdocs en 3 professoren, die onderzoek doen naar zowel de theoretische als de praktische aspecten van cryptografie en netwerkbeveiliging. Dit vertaalt zich in verschillende Europese onderzoeksprojecten en een nauwe samenwerking met de industrie.
Onze opdracht bestond erin de hashfuncties MD5 en Whirlpool te implementeren in hardware en vervolgens de implementaties te vergelijken.
Een hashfunctie is een algoritme dat van een reeks bits van willekeurige lengte een samenvatting berekent met een vast aantal bits, hashwaarde genoemd. Wanneer één of meerdere bits van de data veranderen, zal ook de hashwaarde veranderen. Door de hashwaarde van de ontvangen data te berekenen en deze te vergelijken met de bijgeleverde hashwaarde kan men controleren of die data ongewijzigd gebleven zijn. Hoewel verschillende aanvallen hebben aangetoond dat het veiligheidsniveau van het MD5-algoritme veel lager is dan oorspronkelijk verwacht, wordt MD5 toch nog in verschillende toepassingen gebruikt. Het Whirlpoolalgoritme daarentegen is een vrij nieuwe hashfunctie waarvan minder hardware-implementaties voorhanden zijn. Om een vergelijking te kunnen maken tussen beide hashfuncties hebben we zowel het MD5-algoritme als het Whirlpoolalgoritme geïmplementeerd op een FPGA (Field Programmable Array) met behulp van de hardwarebeschrijvingstaal VHDL (Very High Speed Integrated Circuit Hardware Description Language).
We hebben daarbij meerdere implementaties van beide algoritmes geëvalueerd. Het resultaat is een tabel die enerzijds de verschillende implementaties van eenzelfde algoritme vergelijkt en anderzijds de oppervlakte en de snelheid van Whirlpool en MD5 tegen elkaar afweegt.

Voorwoord

Met deze masterproef willen we onze studies van industrieel ingenieur afronden. Deze masterproef was uiteraard onmogelijk zonder de steun en begeleiding van een aantal mensen die we hierbij willen bedanken.

Eerst en vooral willen we onze promotor, Nele Mentens, bedanken voor al haar toewijding en tijd die ze in de begeleiding van onze masterproef heeft gestoken. Zonder haar hulp zou deze masterproef nooit geworden zijn wat ze nu is.

We zouden ook graag COSIC willen bedanken dat we de kans kregen om deze masterproef uit te voeren. Een speciaal woord van dank gaat uit naar Lejla Batina, onze promotor binnen COSIC, voor haar inzet en hulp.

Verder zouden we graag Ellen Dirikx willen bedanken voor al de tijd en moeite die ze in het nalezen en verbeteren van onze masterproef heeft gestoken.

Ook onze medestudenten die gedurende al deze tijd samen met ons hebben gezwoegd en gezweet willen we graag bedanken voor hun aanmoedigende woorden, de ontspanning tijdens de zware tijden en de toffe sfeer die er altijd heerste. Natuurlijk wil ik, Joris, hierbij Piet in het bijzonder bedanken voor de toffe samenwerking gedurende de afgelopen academiejaren en gedurende het uitwerken van onze masterproef in het bijzonder.
En ik, Piet, wil hierbij ook Joris bedanken voor de toffe medewerking de afgelopen jaren en voor het samen tot stand brengen van het werk dat nu voor u ligt.

Ook wil ik mijn ouders bedanken omdat ze mij de kans gegeven hebben om deze studies tot een goed einde te brengen.

Ik, Joris, zou graag mijn vriendin bedanken voor de steun en moed die ze mij gaf gedurende mijn studies en de uitvoering van deze masterproef. Ook mijn ouders wil ik bedanken voor de kans die ze mij gegeven hebben om deze studies aan te vatten. Een speciaal woord van dank gaat uit naar mijn moeder, die mij na het overlijden van mijn vader de kans en nodige steun heeft gegeven om mijn studies alsnog af te ronden. Ik zou dan ook graag deze masterproef opdragen aan mijn overleden vader.

1 Inleiding

In dit hoofdstuk wordt kort gesitueerd waarover deze masterproef gaat. Er wordt een achtergrond gegeven over cryptografie. Ook komen de mogelijke oplossingen wat hardware betreft aan bod. Een precieze beschrijving van deze masterproef sluit dit hoofdstuk af.

1.1 Situering

Deze masterproef bestaat erin hashfuncties in hardware te implementeren. Deze masterproef willen we tot een goed einde brengen binnen de onderzoeksinstelling COSIC.
Van alle hashfuncties die er bestaan zijn de meeste reeds gebroken, wat wil zeggen dat ze niet meer veilig zijn. Er worden nog altijd nieuwe hashfuncties ontwikkeld die geacht worden veilig te zijn. De meeste van deze nieuwe hashfuncties werden nog niet vaak of nog niet geïmplementeerd. Daarom zullen wij een bekend algoritme implementeren dat reeds gebroken is, alsook een nieuw algoritme dat nog niet gebroken is, zodat we beide implementaties met elkaar kunnen vergelijken.

1.2 COSIC

COSIC staat voor "Computer Security and Industrial Cryptography", en is een onderzoeksinstelling binnen de Katholieke Universiteit van Leuven, departement ESAT-Elektrotechniek.[1]
Het doel van het onderzoek binnen COSIC is het creëren van elektronische equivalenten voor cryptografische primitieven in de fysische wereld zoals vertrouwelijkheid, handtekeningen, identificatie, anonimiteit, notariële bekrachtigingen en betalingen.
Om dit doel te bereiken spitst de onderzoeksinstelling zich toe op het ontwikkelen, evalueren en implementeren van cryptografische algoritmes en protocollen en op het ontwikkelen van veiligheidssystemen voor computersystemen en telecommunicatienetwerken.
Het theoretische onderzoek van COSIC op cryptografische algoritmes en protocollen is hoofdzakelijk gebaseerd op discrete wiskunde. Andere wiskundige takken die relevant zijn voor het onderzoek zijn statistiek en optimalisatie. Het doel is om efficiënte en aantoonbaar veilige oplossingen te creëren.
COSIC heeft de intentie om deze oplossingen te integreren in verschillende toepassingen zoals computersystemen, telecommunicatiesystemen (bv. veiligheid van internettoepassingen, mobiele communicatie) en betalingssystemen. Een belangrijk aspect hierbij is de efficiënte implementatie (in zowel software als hardware) van cryptografische primitieven en de evaluatie van de veiligheid van de componenten en systemen, inclusief smartcards.
COSIC geeft raad op het gebied van computerveiligheid en cryptografie en werkt wereldwijd samen met zowel universiteiten als bedrijven en organisaties.

[1] KATHOLIEKE UNIVERSITEIT LEUVEN, *Computer Security and Industrial Cryptography*, online, http://www.esat.kuleuven.be/cosic/, 1 december 2005.

1.3 Cryptografie

Het woord cryptografie stamt af van de Griekse woorden 'kruptos' en 'graphein', die respectievelijk 'verborgen' en 'schrijven' betekenen. Cryptografie is dus een ander woord voor geheimschrift.
Cryptografie werd reeds gebruikt door de Egyptenaren, zowat 4000 jaar geleden. Sommige hiërogliefen, teruggevonden op monumenten van de oude Egyptenaren, vertoonden al een vorm van cryptografie.[2]
Cryptografie is vooral bekend voor het coderen van een bericht aan de kant van de zender, zodat een niet-gemachtigd persoon die de gecodeerde tekst kan onderscheppen, deze tekst niet kan begrijpen. Aan de kant van de ontvanger wordt de tekst weer gedecodeerd. Het coderen gebeurt met behulp van een sleutel die essentieel is. Wanneer de niet-gemachtigde persoon deze sleutel kent, is de encryptie waardeloos. Dat cryptografie echter meer is dan dit, wordt duidelijk gemaakt in de volgende paragrafen.

1.3.1 Begrippen

Een definitie van cryptografie is: 'Cryptografie is de studie die zich bezighoudt met wiskundige technieken, verbonden met beveiliging van informatie zoals betrouwbaarheid, data-integriteit, identificatie en echtheid van de databron.'[3]
Cryptografie is niet alleen het beveiligen van informatie, maar slaat ook op het geheel van technieken die beveiliging van informatie proberen te waarborgen.
Enkele doelen van cryptografie om informatie te beveiligen worden in Tabel 1.1 verklaard.

Betrouwbaarheid of geheimhouding	Informatie geheimhouden voor iedereen die niet gemachtigd is over deze informatie te beschikken
Data-integriteit	Verzekeren dat de informatie niet veranderd werd door onbevoegde of ongekende middelen
Entiteit, echtheid of identificatie	Bevestiging van de identiteit van een entiteit (bv. een persoon, computer terminal, ...)
Echtheid van de boodschap	Bevestiging van de databron (ook bekend als echtheid van de bron van de data)
Handtekening	Een middel om informatie aan een entiteit te verbinden
Autorisatie	Machtiging, tegenover een entiteit, om officieel iets te doen of te zijn
Validatie	Een middel om een tijdloze machtiging te geven om informatie of hulpbronnen te gebruiken of te veranderen
Toegangscontrole	Het beperken van toegang tot hulpbronnen voor bevoorrechte entiteiten
Certificatie	Bevestigen van informatie door een betrouwbare entiteit
Timestamping	Het opslaan van het tijdstip waarop informatie wordt gecreëerd of de tijdspanne waarin informatie bestaat
Getuigen	Het verifiëren van de creatie of het bestaan van informatie door een entiteit verschillend van de schepper

[2] CYPHER RESEARCH LABORATORIES PTY LTD, *History of cryptography*, online, http://www.cypher.com.au/crypto_history.htm, 1 december 2005
[3] MENEZES, A. J., *Handbook of applied cryptography*, online, http://www.cacr.math.uwaterloo.ca/hac/, 1 december 2005.

Betrouwbaarheid of geheimhouding	Informatie geheimhouden voor iedereen die niet gemachtigd is over deze informatie te beschikken
Ontvangstbewijs	Erkennen dat de informatie ontvangen werd
Bevestiging	Erkennen dat diensten bewezen werden
Bezit	Een middel om een entiteit te voorzien van een legaal recht om een hulpmiddel te gebruiken of door te geven aan anderen
Anonimiteit	Het geheimhouden van de identiteit van een entiteit betrokken in een proces
Onweerlegbaarheid	Het verhinderen dat men eerdere acties of betrokkenheid ontkent
Revocatie	Het terugnemen van een certificaat of autorisatie

Tabel 1.1: Enkele doelen van cryptografie

Alle doelen om informatie te beveiligen, weergegeven in Tabel 1.1, stammen af van de volgende vier doelen:

- Betrouwbaarheid of geheimhouding heeft tot doel om de inhoud van informatie geheim te houden voor iedereen die niet gemachtigd is hierover te beschikken.

- Data-integriteit heeft tot doel de verandering van data door onbevoegden op te sporen. Deze verandering omvat zowel het toevoegen, verwijderen als het veranderen van data.

- Echtheid hangt samen met identificatie en heeft zowel betrekking op de entiteit als op de data zelf. Wanneer twee partijen willen communiceren, moeten ze elkaar kunnen identificeren. Informatie die over een kanaal verzonden wordt, moet gecontroleerd worden op de bron, de datum van de bron, de tijd van verzending enz. Daarom wordt dit aspect van cryptografie meestal in twee verdeeld: echtheid van de entiteit en echtheid van de data. Echtheid van de databron betekent impliciet ook integriteit van de data. Wanneer een boodschap veranderd is, gebeurde dit door een andere bron, wat dus wil zeggen dat de bron veranderd is.

- Onweerlegbaarheid heeft tot doel dat een entiteit voorgaande acties of betrokkenheid niet kan ontkennen. Wanneer er onenigheid ontstaat omdat een entiteit ontkent dat bepaalde acties ondernomen werden, is er een middel nodig om deze situatie op te lossen. Een procedure met een derde vertrouwde partij is nodig om de onenigheid op te lossen.

Het is een basisdoel van cryptografie deze vier doelen om informatie te beveiligen zowel in theorie als in praktijk adequaat te behandelen. Cryptografie wil fraude en andere criminele activiteiten tegengaan en detecteren.
De cryptografische algoritmes kunnen opgesplitst worden in drie gebieden, die in de volgende paragrafen worden geïntroduceerd.[3]

[3] MENEZES, A. J., *Handbook of applied cryptography*, online, http://www.cacr.math.uwaterloo.ca/hac/, 1 december 2005.

1.3.2 Symmetrische-sleutelalgoritme

Bij een symmetrische-sleutelalgoritme gebruiken zender en ontvanger dezelfde geheime sleutel.[4] De zender codeert met de geheime sleutel de data die hij wil verzenden. De ontvanger kan met de geheime sleutel de ontvangen gecodeerde data decoderen. Een aanvaller die geïnteresseerd is in de data kan het kanaal tussen de zender en de ontvanger afluisteren, waarover de data verzonden worden. Over dit kanaal worden enkel de gecodeerde data verzonden, die zonder de geheime sleutel onbegrijpelijk zijn.

1.3.3 Publieke-sleutelalgoritme

Dit soort algoritme wordt ook wel een asymmetrische-sleutelalgoritme genoemd. In tegenstelling tot het symmetrische-sleutelalgoritme wordt er bij dit algoritme gebruik gemaakt van twee verschillende sleutels, die samen een sleutelpaar vormen.[4] Elke sleutel van dit paar vormt de inverse van de andere, wat betekent dat data, gecodeerd met de ene sleutel, gedecodeerd kunnen worden met de andere sleutel. Aan de hand van één van de sleutels kan men de andere sleutel niet achterhalen.
De publieke sleutel kan door iedereen opgevraagd worden terwijl de private sleutel slechts door één entiteit gekend is.
Men kan dit algoritme voor twee doelen gebruiken: ofwel om de zender van een bericht te verifiëren, ofwel om er zeker van te zijn dat een bericht enkel door de houder van de private sleutel gelezen kan worden.
Wanneer er met de private sleutel een bericht gecodeerd wordt, kan men bewijzen dat de houder van de private sleutel dit bericht verstuurd heeft.
Indien met de publieke sleutel een bericht gecodeerd wordt, weet men zeker dat enkel de houder van de private sleutel dit bericht zal kunnen decoderen.

1.3.4 Hashfuncties

Een hashfunctie is een efficiënt algoritme dat een reeks bits van willekeurige lengte afbeeldt op een reeks bits van vaste lengte, hashwaarde genoemd.[3] Wanneer de data veranderen, zal ook de hashwaarde veranderen; op deze manier is het dus vrij eenvoudig om er zeker van te zijn dat de data onveranderd zijn.
In tegenstelling tot symmetrische- en publieke-sleutelalgoritmes is een hashfunctie onomkeerbaar. Dit wil zeggen dat wanneer een hashwaarde gegeven is, het onmogelijk is om hieruit de boodschap waaruit de hashwaarde berekend werd te vinden.
Indien een aanvaller de hashwaarde kent, en zelfs de originele boodschap heeft, is en blijft het toch bijna onmogelijk om een andere boodschap te vinden die dezelfde hashwaarde geeft. Indien men een handtekening plaatst op deze hashwaarde, heeft dat dezelfde waarde als wanneer men een handtekening plaatst op de originele boodschap zelf.
Het is bijna onmogelijk om twee boodschappen te vinden die dezelfde hashwaarde geven, anders zou het ondertekenen van een hashwaarde voor beide boodschappen gelden.

---

[4] OPEN FORTRESS, *Toolbox der Cryptografie*, online, http://openfortress.nl/doc/essay/CryptoToolbox/index.nl.html, 1 december 2005.
[4] OPEN FORTRESS, *Toolbox der Cryptografie*, online, http://openfortress.nl/doc/essay/CryptoToolbox/index.nl.html, 1 december 2005.
[3] MENEZES, A. J., *Handbook of applied cryptography*, online, http://www.cacr.math.uwaterloo.ca/hac/, 1 december 2005.

1.4 VHDL

VHDL staat voor VHSIC Hardware Description Language, waarbij VHSIC staat voor Very High Speed Integrated Circuit.
VHDL is een taal die hardware beschrijft. Een software programmeertaal wordt uitgevoerd op bestaande hardware. Bij een taal die hardware beschrijft, wordt de code gesynthetiseerd tot componenten, die in hardware vertaald worden. Men beschrijft dus aan de hand van de code hoe componenten met elkaar verbonden worden en hoe deze componenten werken. De synthese geeft de componenten weer die het systeem bevat, en deze synthese wordt in een programmeerbare chip geladen of van deze synthese wordt een ASIC gemaakt. Deze chip zal dan de hardware bevatten die in de code beschreven werd. Als programmeerbare chip kunnen we zowel een FPGA als een CPLD gebruiken.

1.5 Mogelijke chips

1.5.1 ASIC

ASIC staat voor Application-specific Integrated Circuit.[5] Deze IC is specifiek ontworpen voor de toepassing waarvoor hij moet dienen. Hierdoor zal de ASIC qua snelheid de beste oplossing bieden.
Aangezien een ASIC specifiek is voor een bepaald ontwerp, moet deze ook specifiek ontworpen worden, waardoor de productiekosten en productietijd hoog zijn. Indien men geen massaproductie gaat toepassen, wordt er enkel gebruik gemaakt van ASIC indien de snelheid van belang is.

1.5.2 FPGA

FPGA staat voor Field Programmable Gate Array.[6] Een FPGA is een chip die programmeerbare logische componenten en programmeerbare verbindingen tussen deze componenten bevat. De programmeerbare logische componenten kunnen zowel voor logische poorten gebruikt worden (en-, of- en niet-poorten) als voor meer complexe combinatorische functies, zoals schuifregisters, tellers, vermenigvuldigers enz. Over het algemeen zijn FPGAs trager dan ASICs en verbruiken ze meer vermogen. Toch hebben FPGAs ook voordelen, omdat ze gemakkelijk opnieuw geprogrammeerd kunnen worden en minder ontwikkeltijd vragen.

1.5.3 CPLD

CPLD staat voor Complex Programmable Logic Device.[7] Net als een FPGA bevat een CPLD programmeerbare logische componenten en programmeerbare verbindingen tussen deze componenten. Een CPLD heeft in de chip, in tegenstelling tot een FPGA, niet-vluchtig geheugen, wat betekent dat ook na spanningsonderbreking de data in het geheugen blijven

[5] WIKIPEDIA, *Application-specific integrated circuit*, online, http://en.wikipedia.org/wiki/Asic, 1 december 2005.
[6] WIKIPEDIA, *Field-programmable gate array*, online, http://en.wikipedia.org/wiki/FPGA, 1 december 2005.
[7] WIKIPEDIA, *CPLD*, online, http://en.wikipedia.org/wiki/CPLD, 1 december 2005.

zitten. Een FPGA daarentegen zal na spanningsonderbreking niet meer de juiste data bevatten in het geheugen. De nadelen die een FPGA heeft t.o.v. een ASIC gelden ook voor een CPLD.

1.6 De opdracht

De algoritmes die geïmplementeerd zullen worden zijn MD5, een algoritme dat tot op zeker niveau gebroken is, en Whirlpool, een meer recent algoritme dat momenteel nog als veilig wordt beschouwd. MD5 is een algoritme dat nog steeds vaak gebruikt wordt. Aangezien het Whirlpoolalgoritme nog vrij recent is, zijn hiervan minder implementaties beschikbaar waardoor een grondige vergelijking ontbreekt in de beschikbare literatuur. De opdracht bestaat uit verschillende stappen die hierna toegelicht worden.

1.6.1 Hashfuncties bestuderen

Het eerste deel bestaat uit het bestuderen en begrijpen van de werking van het MD5- en Whirlpoolalgoritme. Dit is noodzakelijk om de hardware-implementaties op een goede manier te kunnen voltooien.

1.6.2 Hashfuncties implementeren en vergelijken

De opdracht bestaat uit het implementeren van het MD5- en Whirlpoolalgoritme in hardware. Na deze implementatie worden ze vergeleken. De vergelijking op basis van benodigde oppervlakte kunnen we doen aan de hand van het aantal slices dat nodig is voor de implementatie op de FPGA. Verder zal er ook een vergelijking gemaakt worden op basis van het aantal poorten dat nodig is op een ASIC. Ook zullen van beide algoritmes de hashwaarde van eenzelfde boodschap berekend worden en de tijden die hiervoor nodig zijn met elkaar vergeleken worden.

1.7 Overzicht

In Hoofdstuk 2 wordt de werking van het MD5-algoritme besproken. Hoofdstuk 3 doet de implementatie van dit algoritme uit de doeken. In Hoofdstuk 4 wordt de werking van het Whirlpoolalgoritme besproken. Hoofdstuk 5 behandelt de implementatie van het Whirlpoolalgoritme. Het 6de Hoofdstuk bespreekt de resultaten. In Hoofdstuk 7 wordt er een algemeen besluit gegeven.

2 Het MD5-algoritme

In dit hoofdstuk zal het MD5-algoritme in detail besproken.

2.1 Ontstaan

Het MD5-algoritme werd ontwikkeld door Ron Rivest in 1991 en was lange tijd het meest gebruikte veilige hashalgoritme.[8] [9]
MD5 staat voor Message Digest Algorithm 5, wat letterlijk "samenvatting-van-de-boodschapalgoritme" betekent. Van een boodschap van willekeurige lengte wordt met het MD5-algoritme een samenvatting van 128 bits gegenereerd. Deze samenvatting wordt hashwaarde genoemd.

2.2 Het algoritme

Het algoritme zet een boodschap van willekeurige lengte om in een hashwaarde van 128 bits (deze boodschap mag ook 0 bit bedragen).
Het algoritme kan omschreven worden in 5 stappen.
In de eerste 2 stappen zal de boodschap van willekeurige lengte verlengd worden; de uiteindelijke boodschap, die we data gaan noemen, heeft als lengte een veelvoud van 512 bits. Van deze data zal uiteindelijk de MD5-hashwaarde berekend worden.

2.2.1 Stap 1: het toevoegen van opvullingbits aan de boodschap

Er worden opvullingbits toegevoegd aan de boodschap zodat de lengte altijd 64 bits minder is dan een veelvoud van 512 bits (ofwel lengte = 448 modulo 512). Dit wordt weergegeven in Figuur 2.1. Het toevoegen van deze bits gebeurt altijd, ook wanneer de boodschap reeds een lengte heeft van 448 modulo 512. Bijvoorbeeld, bij een boodschap van 448 bits worden 512 opvullingbits toegevoegd om zo een boodschap met opvullingbits te krijgen met een lengte van 960 bits, wat weer overeenkomt met 448 modulo 512. Het aantal opvullingbits bedraagt dus minimum 1 bit en maximum 512 bits. Deze opvullingbits bestaan uit een 1, gevolgd door 0'en, en worden toegevoegd aan de kant van de meest beduidende bit. De 1 wordt toegevoegd aan de kant van de meest beduidende bit van de boodschap (en is dus de minst beduidende bit van de opvullingbits).

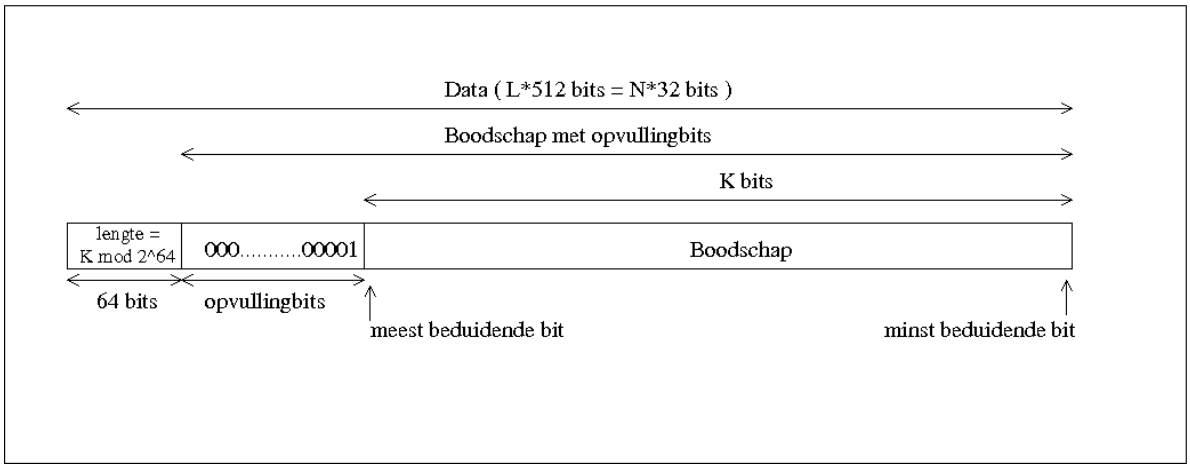
2.2.2 Stap 2: het toevoegen van de lengte aan de boodschap met opvullingbits

Tot slot worden er bij de boodschap met opvullingbits nog 64 bits toegevoegd; zo bekomt men de data die verwerkt zullen worden (zie Figuur 2.1). Deze 64 bits die toegevoegd worden, geven de lengte van de boodschap in bits weer. Deze bits worden weer toegevoegd aan de kant van de meest beduidende bit van de opgevulde boodschap; de meest beduidende bit van de lengte vormt de meest beduidende bit van de data. Indien de boodschap groter is dan 2^{64}, worden enkel de minst beduidende bits van de lengte gebruikt.

---

[8] Stallings, W., *Cryptography and Network Security*, *Principles and Practices*, derde druk, Prentice Hall, Upper Saddle River, 2003.
[9] ADVAMEG, INC., *RFC1321*, online, http://www.faqs.org/rfcs/rfc1321.html, 1 december 2005.

Deze data zijn nu een veelvoud van 512 bits; deze 512 bits kunnen opgesplitst worden in 16 blokken van 32 bits.
De hashwaarde van deze data zal berekend worden per blok van 512 bits. Intern zal er gerekend worden met 16 blokken van 32 bits. Elke ronde bestaat uit 16 stappen, waarbij iedere stap 32 bits van de data gebruikt.



Figuur 2.1: Het toevoegen van opvullingbits (MD5-algoritme)

2.2.3 Stap 3: het initialiseren van de MD-buffer

Om de tussenresultaten te bewaren wordt er gebruikgemaakt van een buffer van 128 bits. De buffer kan ook voorgesteld worden door 4 registers van 32 bits (a, b, c, d). Bij de verwerking van de eerste 512 bits krijgen deze de volgende initiële waardes mee:[9]

a = 0x67452301
b = 0xefcdab89
c = 0x98badcfe
d = 0x10325476

Deze waardes zijn hexadecimaal weergegeven, wat wordt aangegeven door 0x. Het minst beduidende hexadecimale getal staat rechts en het meest beduidende hexadecimale getal staat links.

2.2.4 Stap 4: het verwerken van het bericht in blokken van 512 bits

De kern van het algoritme is een compressiefunctie die bestaat uit vier rondes, die een gelijke structuur hebben, maar verschillen in de primitieve logische functie die ze gebruiken:

F = (X and Y) or (not (X) and Z)
G = (X and Z) or (Y and not (Z))
H = X xor Y xor Z
I = Y xor (X or (not (Z)))

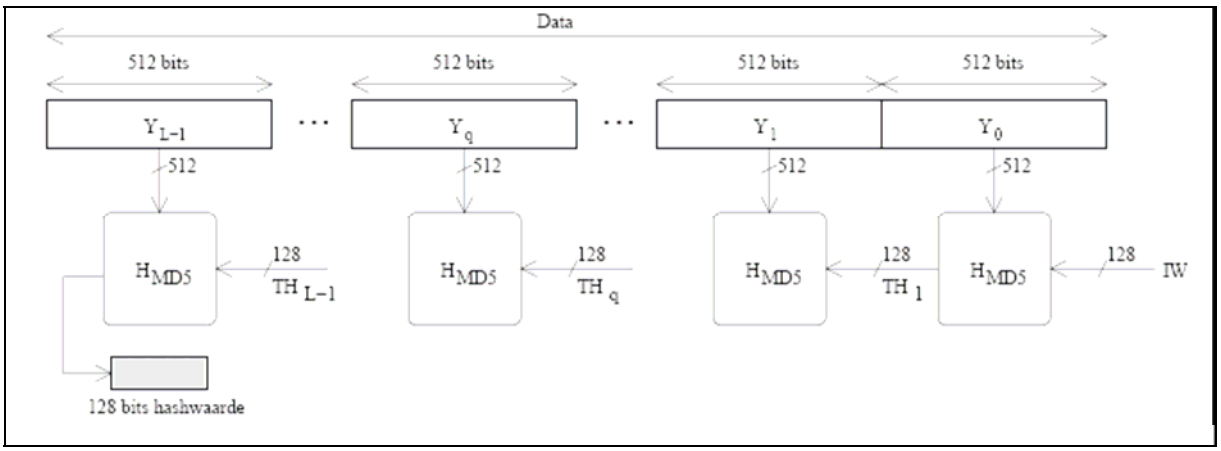
[9] ADVAMEG, INC., *RFC1321*, online, http://www.faqs.org/rfcs/rfc1321.html, 1 december 2005.

Per logische functie zijn deze vier rondes nogmaals opgesplitst in 16 rondes. Dit geeft in totaal 64 rondes. Indien er verderop over een ronde gesproken wordt, wordt hiermee 1 van deze 64 rondes bedoeld.
Wanneer de boodschap uit meer dan 512 bits bestaat, worden deze per blok van 512 bits berekend. De tussentijdse hashwaarde van elk blok wordt berekend aan de hand van deze vier rondes.
Elke ronde heeft als ingang het blok van 512 bits dat verwerkt wordt, en de MD-buffer van 128 bits. Elke ronde zal de waarde van de MD-buffer aanpassen. Ook wordt er gebruik gemaakt van 16 waardes uit een tabel van 64 elementen, die berekend werd aan de hand van de sinusfunctie. Het i'de element wordt als volgt berekend: $T[i]=2^{32} * abs(sin(i))$ (met i in radialen; alleen het gehele gedeelte wordt gebruikt; i begint vanaf 1). Omdat de sinus een getal is tussen 0 en 1 zal de waarde die we bekomen voorgesteld kunnen worden door 32 bits. Deze tabel geeft 32 willekeurige bits die ervoor zorgen dat er een willekeurig patroon aan de ingang ontstaat. De functie van de eerste 16 rondes ziet er als volgt uit:

$a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)$

a, b, c en d worden bij de eerste stap van de eerste ronde uit de MD-buffer gelezen. De waardes van a, b, c en d komen overeen met een welbepaald deel uit de MD-buffer. Voor de eerste, vijfde, negende en dertiende ronde is dit respectievelijk A, B, C en D; voor de tweede, zesde, tiende en veertiende ronde is dit D, A, B en C; voor de derde, zevende, elfde en vijftiende ronde is dit C, D, A en B. Voor de vierde, achtste, twaalfde en zestiende ronde ten slotte is dit B, C, D en A. Dit wil dus zeggen dat in de eerste stap A wordt aangepast, in de tweede stap D enzovoort.
X[k] zijn 32 bits van de 512 bits van de data die verwerkt worden; T[i] is een element uit de tabel berekend aan de hand van de sinus. De <<<s betekent dat er s keer een circulaire schuifactie moet gebeuren naar links. Dit wil zeggen dat de meest beduidende bit die uit het register geschoven wordt, weer ingeschoven wordt als minst beduidende bit.

De functies van rondes 2, 3 en 4 zijn gelijkaardig, alleen gebruiken deze respectievelijk G, H en I als functie.
De uitgang van de laatste ronde wordt opgeteld bij de ingang van de eerste ronde. Deze optelling gebeurt per 32 bits (A, B, C, D), de uitkomst is opnieuw 32 bits. Er wordt geen rekening gehouden met de overdrachtsbit. Wanneer men A, B, C en D achter elkaar plaatst, met als meest beduidende bit de meest beduidende bit van D, en als minst beduidende bit de minst beduidende bit van A, krijgt men de tussentijdse hashwaarde.
Indien de data, waarvan de hashwaarde berekend moet worden, meer dan 512 bits bedragen, zal deze uitgang, de tussentijdse hashwaarde, in de MD-buffer geschreven worden, en zal dus gebruikt worden als ingang van de eerste ronde bij de volgende 512 bits. Dit wordt weergegeven in Figuur 2.2.



Figuur 2.2: De werking van het MD5-algoritme

(IW = initiële waarde van de buffer, Y_q = het q-de blok van 512 bits van de data, L = het aantal blokken van 512 bits waaruit de data bestaan, TH_q = tussentijdse hashwaarde)

2.2.5 Stap 5: de uitgang

Na de verwerking van alle blokken van 512 bits is de tussentijdse hashwaarde van het laatste blok van 512 bits gelijk aan de hashwaarde van de boodschap die 128 bits groot is.

3 De implementatie van het MD5-algoritme

In dit hoofdstuk wordt besproken hoe het MD5-algoritme geïmplementeerd werd.

3.1 De ingangssignalen

Bij de implementatie worden volgende ingangssignalen gebruikt :

- *clk*: dit is de klokfrequentie waarop de gehele implementatie zal werken.
- *data_in*: via deze 32 lijnen wordt de boodschap ingelezen.
- *shift_data_in*: dit signaal duidt aan dat er 512 nieuwe bits ingelezen kunnen worden.
- *rst*: als dit signaal hoog wordt worden alle registers naar hun beginwaarde (0) gezet.
- *start*: met behulp van dit signaal wordt aangeduid of een boodschap uit meerdere blokken van 512 bits bestaat.
- *shift_data_out*: dit signaal geeft aan dat de hashwaarde naar buiten gestuurd kan worden.

3.2 De uitgangssignalen

Bij de implementatie worden volgende uitgangssignalen gebruikt :

- *data_out*: via deze 32 lijnen wordt de hashwaarde naar buiten gestuurd.
- *outputready*: dit signaal geeft aan dat de hashwaarde berekend is.

3.3 Afspraken

Er werd gekozen om het toevoegen van de opvullingsbits door software te laten uitvoeren. Hierdoor zullen de data steeds een veelvoud van 512 bits bedragen. De software zal de data per 512 bits aanbieden aan de ingang van de implementatie. Ook zal deze software ervoor zorgen dat de andere ingangssignalen op het juiste tijdstip worden aangeboden en dat de uitgangssignalen op het juiste tijdstip worden uitgelezen zodat er geen informatie verloren kan gaan.

3.3.1 Het inlezen van de boodschap

Om de boodschap in te lezen worden volgende signalen gebruikt : *data_in*, *shift_data_in*, *start* en *outputready*. Via *data_in* zal de boodschap ingelezen worden. Aangezien een boodschap 512 bits telt en er voor *data_in* 32 bits voorzien zijn zal het inlezen van een boodschap 16 klokperiodes in beslag nemen.

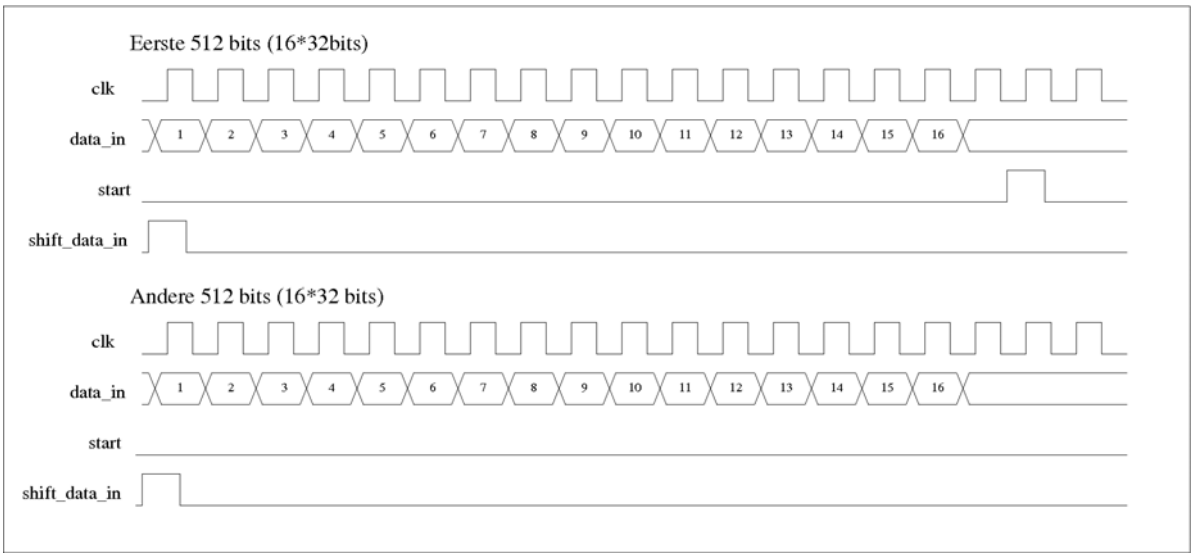
3.3.1.1 Eerste 512 bits inlezen

Op het moment dat *shift_data_in* hoog wordt moeten de 32 minst beduidende bits van de boodschap aan *data_in* aangeboden worden. Op de stijgende klokflank zullen, terwijl *shift_data_in* hoog is, deze 32 minst beduidende bits van de boodschap ingelezen en opgeslagen worden in een register. Na het inlezen van de eerste 32 bits wordt de ingang *shift_data_in* terug laag gemaakt. Hierna zal bij iedere stijgende klokflank opnieuw *data_in*

ingelezen worden, waar iedere keer de 32 volgende bits van de boodschap aangeboden worden. Bij de 16de stijgende klokflank worden de 32 meest beduidende bits van de boodschap ingelezen. Hierna zijn 512 bits ingelezen, en kan de hashwaarde van deze 512 bits berekend worden. Voor de 18de stijgende klokflank moet start hoog gemaakt zijn, om duidelijk te maken dat het de eerste 512 bits zijn. Figuur 3.1 geeft dit weer.

3.3.1.2 Volgende 512 bitsblokken inlezen

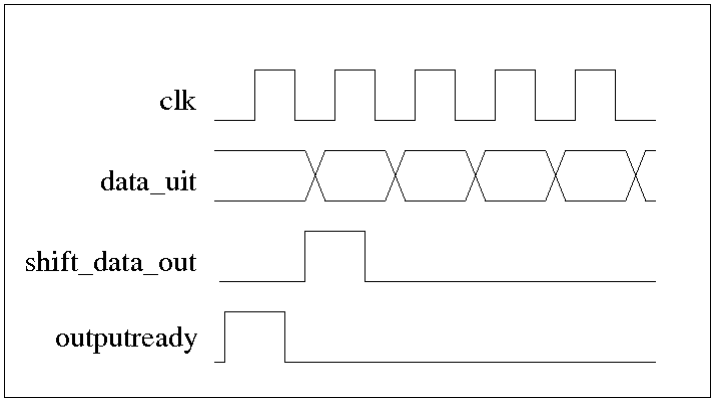
Een eventueel volgend deel van de totale boodschap kan pas ingelezen worden op het moment dat het uitgangssignaal *outputready* hoog geworden is. Dit geeft aan dat de vorige 512 bits verwerkt werden. Het inlezen gebeurt op dezelfde manier als het inlezen van de eerste 512 bits maar het ingangssignaal *start* zal laag blijven. Dit wordt weergegeven in Figuur 3.1.



Figuur 3.1: Timing van het inlezen

3.3.2 Het uitlezen van de hashwaarde

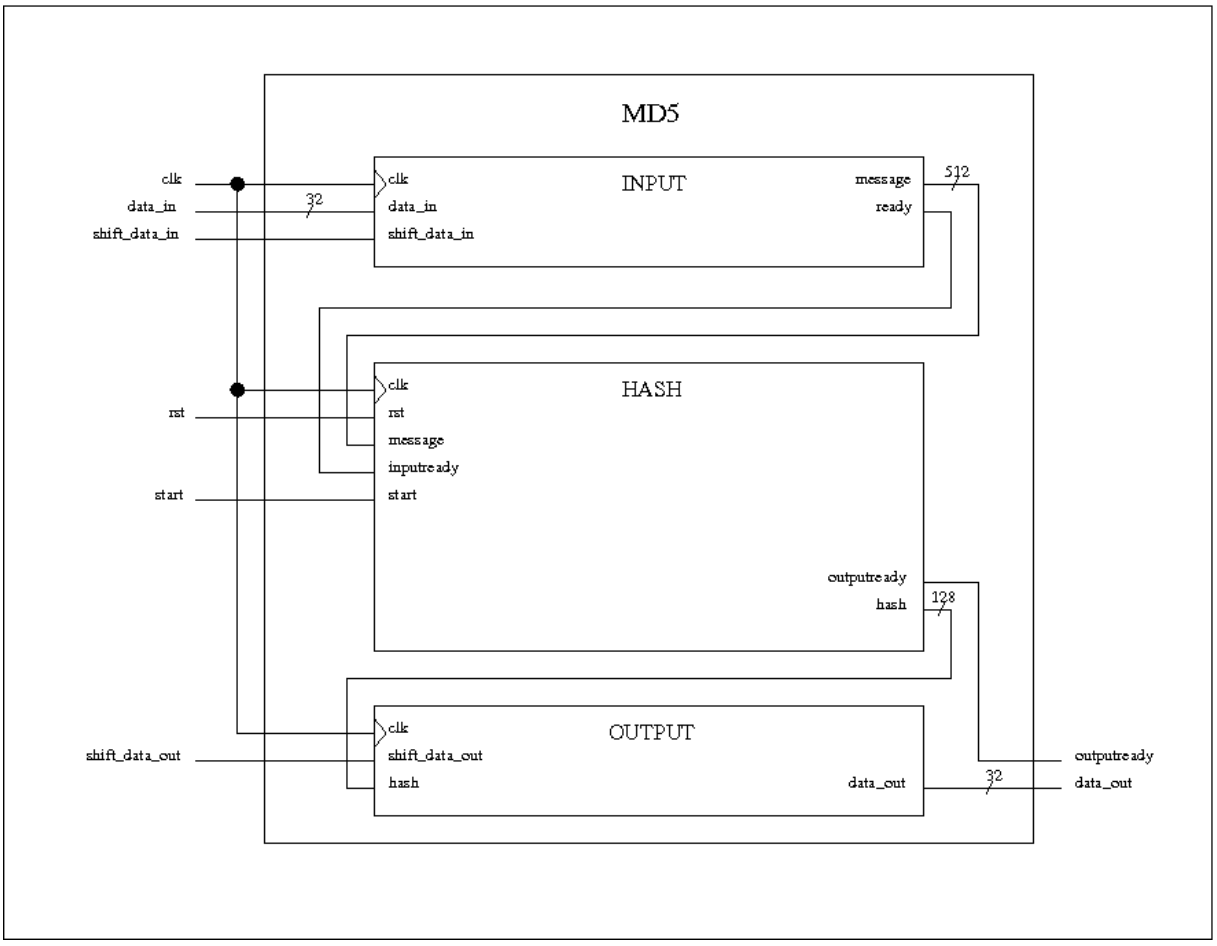
Om de hashwaarde uit te lezen worden volgende signalen gebruikt: *data_out*, *shift_data_out* en *outputready*. Via *data_out* zal de hashwaarde uitgestuurd worden. Aangezien een hashwaarde 128 bits groot is en er voor *data_out* 32 bits voorzien zijn zal het uitlezen van de boodschap 4 klokperiodes in beslag nemen.
Op het moment dat de hashwaarde klaar staat om uitgelezen te worden zal het uitgangssignaal *outputready* gedurende 1 klokperiode hoog worden. Hierna zal de software het ingangssignaal *shift_data_out* gedurende 1 klokperiode hoog maken, wat aanduidt dat de software klaar is om de hashwaarde uit te lezen. Dit wordt weergegeven in Figuur 3.2.
De 32 minst beduidende bits van de hashwaarde worden aan de uitgang aangeboden bij de stijgende klokflank terwijl *shift_data_out* hoog is. Hierna worden bij iedere stijgende klokflank de volgende 32 bits aangeboden tot bij de 4de stijgende klokflank de 32 meest beduidende bits aangeboden worden.



Figuur 3.2: Timing van het uitlezen

3.4 MD5-bouwblok

Alle ingangssignalen worden aan MD5 aangeboden en na verwerking zal MD5 de uitgangssignalen aansturen. MD5 is opgebouwd uit drie bouwblokken, namelijk Input, Hash en Output, zoals weergegeven in Figuur 3.3. MD5 zorgt ervoor dat deze drie blokken de nodige ingangen aangeboden krijgen en dat de uitgangen van deze blokken uitgelezen worden. Na het inlezen van 512 bits in Input, worden deze 512 bits aangeboden aan Hash. Hierin zal de hashwaarde berekend worden die hierna aan Output wordt aangeboden.



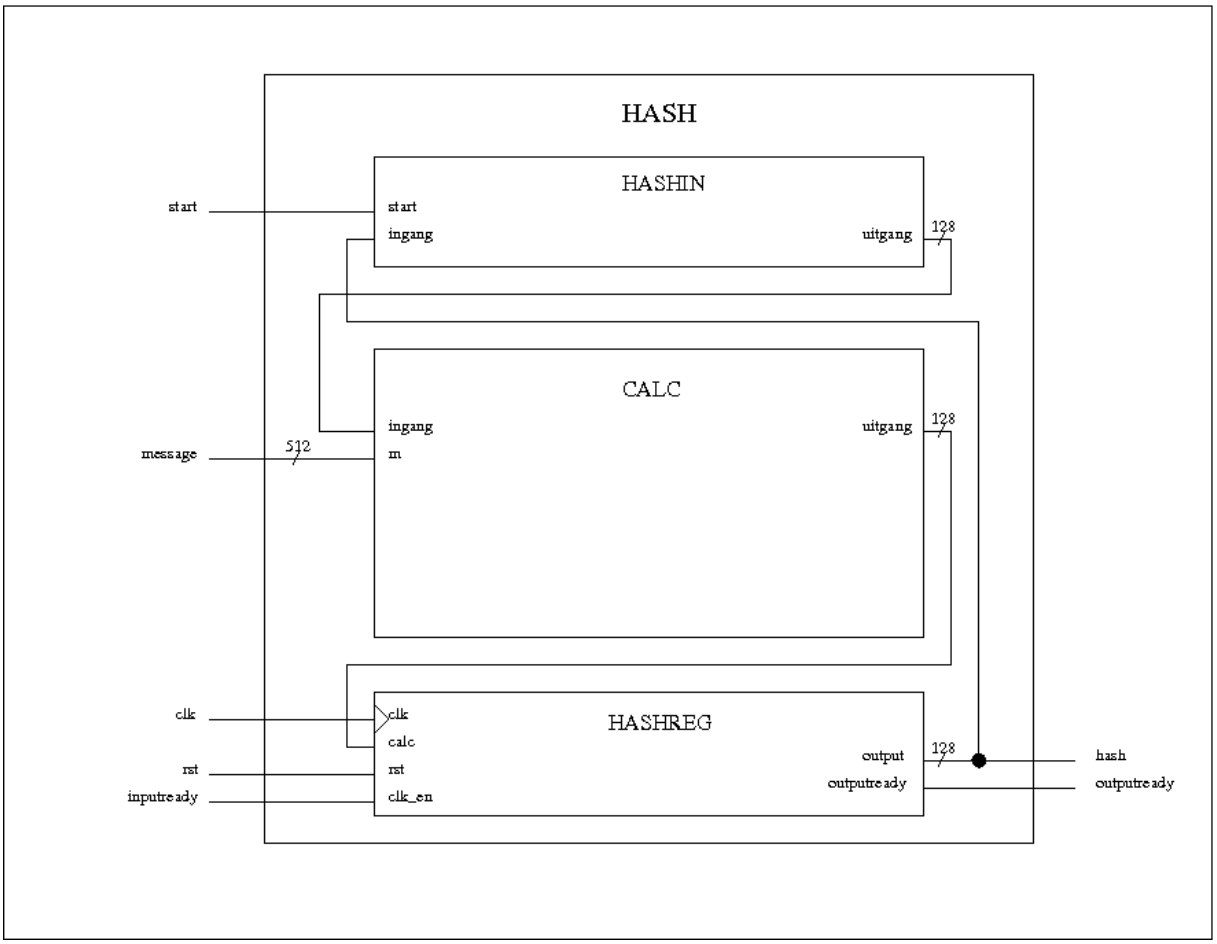
Figuur 3.3: Architectuur van het MD5-bouwblok

3.4.1 Input-bouwblok

Dit bouwblok bestaat uit een register van 512 bits waarin een deel van de boodschap opgeslagen wordt.
Bij iedere stijgende klokflank worden eerst de 32 bits van *Data_in* ingelezen in de 32 meest beduidende bits van dit register. Bij de volgende stijgende klokflank gebeurt er een schuifactie die ervoor zorgt dat de 480 meest beduidende bits 32 plaatsen verschoven worden, zodat deze de 480 minst beduidende bits worden. Hierdoor wordt er ruimte gecreëerd voor de volgende 32 bits die gelijktijdig ingelezen worden.
Als de volledige boodschap van 512 bits ingelezen is wordt het signaal *ready* hoog gemaakt waardoor het volgende bouwblok, namelijk Hash, met de berekening van de hashwaarde kan beginnen.

3.4.2 Hash-bouwblok

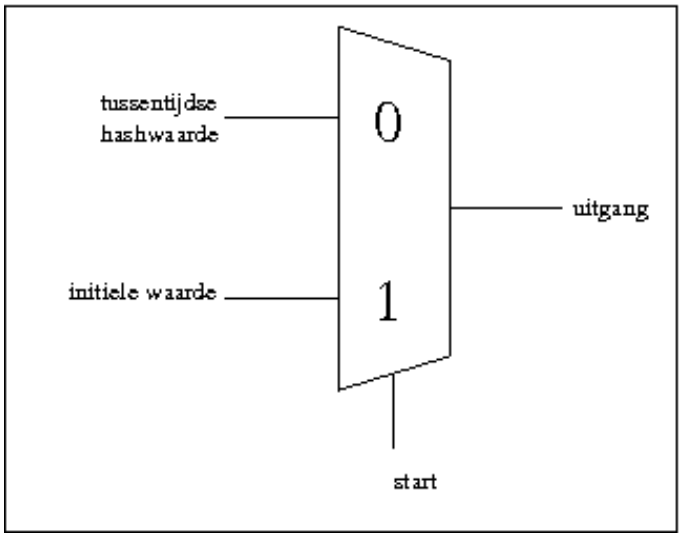
Dit bouwblok is opnieuw opgebouwd uit 3 verschillende bouwblokken, namelijk Hashin, Calc en Hashreg, zoals in Figuur 3.4 weergegeven wordt.



Figuur 3.4: Architectuur van het Hash-bouwblok

3.4.2.1 Hashin-bouwblok

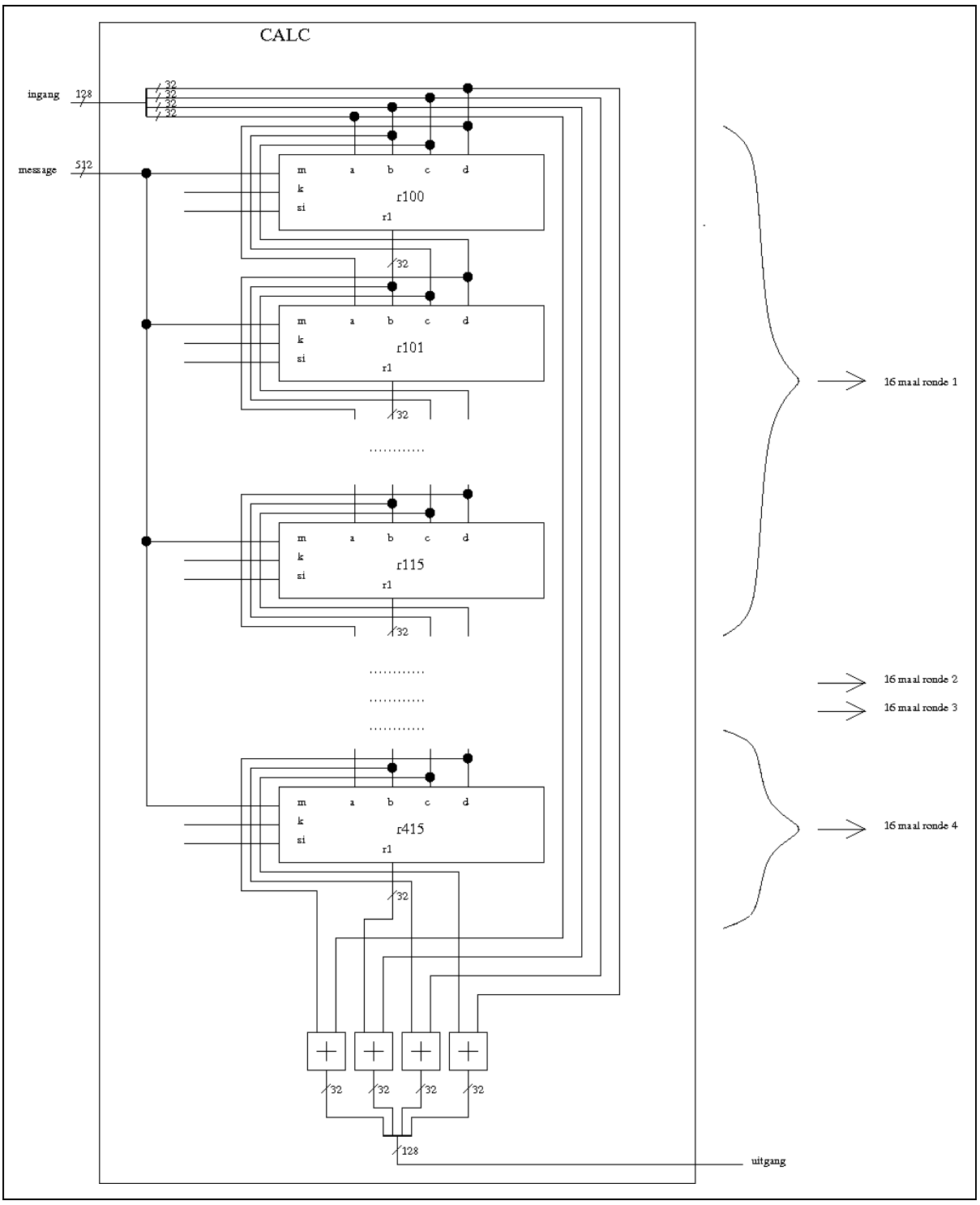
Dit bouwblok bestaat uit een multiplexer. Deze multiplexer gebruikt het ingangssignaal *start* om te beslissen welk ingangssignaal naar de uitgang doorverbonden wordt. Wanneer *start* hoog is, dit is bij het verwerken van de eerste 512 bits, wordt de initiële waarde met de uitgang doorverbonden. Indien *start* laag is wordt de tussentijdse hashwaarde, die aan de ingang aangeboden wordt, met de uitgang doorverbonden.



Figuur 3.5: Architectuur van het Hashin-bouwblok

3.4.2.2 Calc-bouwblok

Dit bouwblok berekent de hashwaarde. Calc is opnieuw opgebouwd uit verschillende bouwblokken. Figuur 3.6 geeft hiervan een schematische voorstelling. In Calc wordt er voor gezorgd dat de verschillende rondes op de juiste manier met elkaar verbonden worden. Dit bouwblok is volledig combinatorisch opgebouwd waardoor de uitgang binnen één klokperiode wordt berekend op basis van de aangelegde ingang. De grootste propagatievertraging binnen Calc zal dan ook de maximale klokfrequentie bepalen.

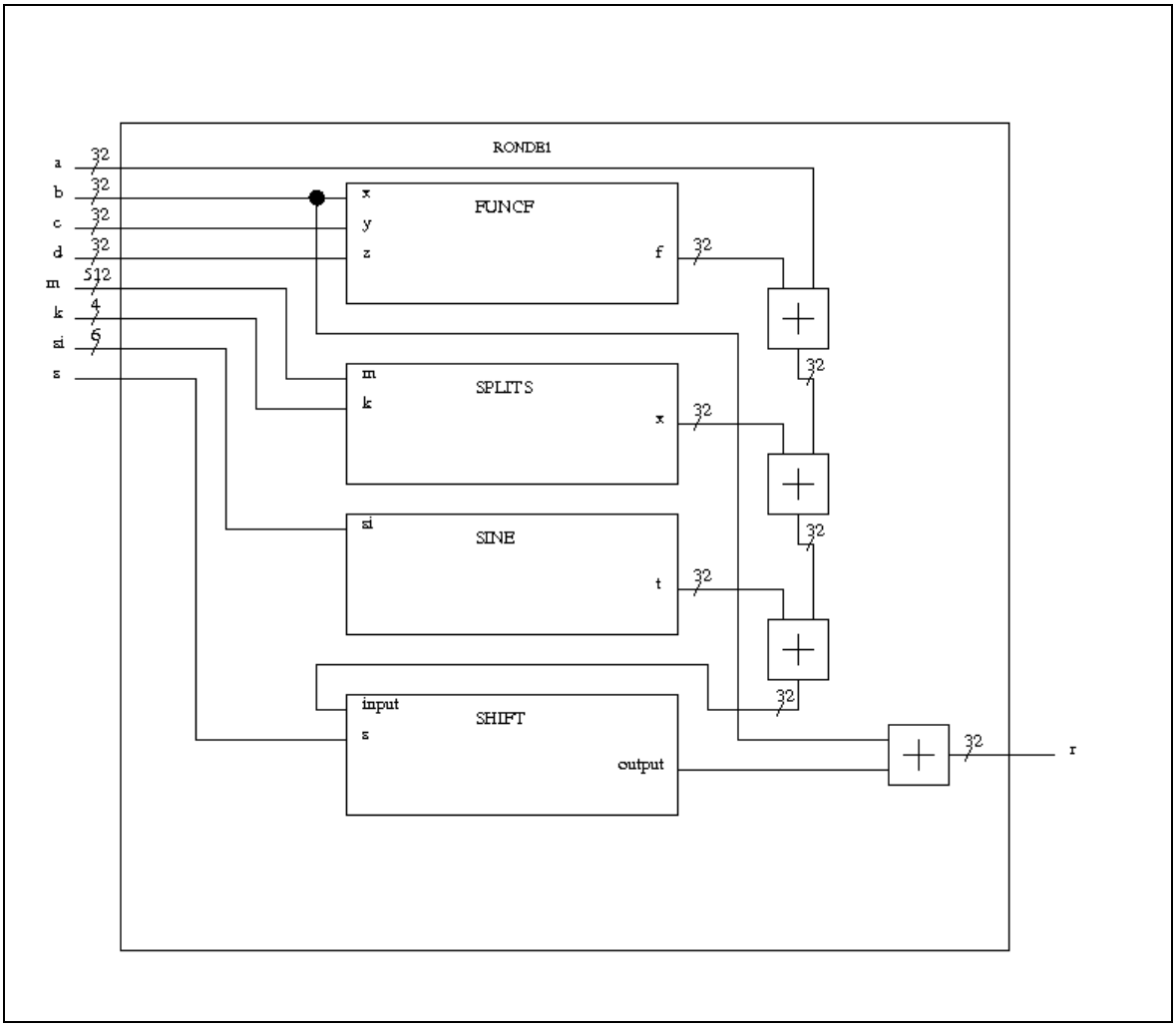


Figuur 3.6: Architectuur van het Calc-bouwblok

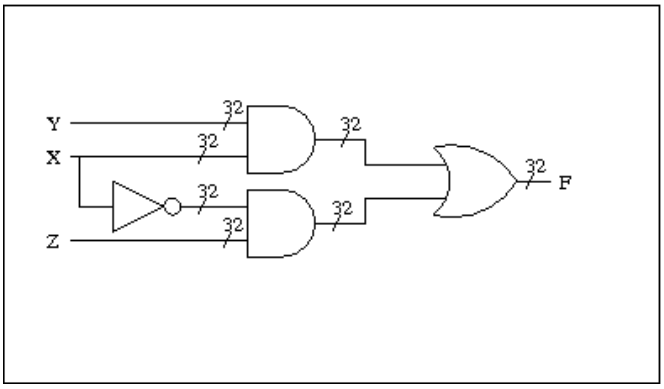
Opbouw van de rondes

Er zijn 64 rondes die allemaal op dezelfde manier opgebouwd zijn. Opnieuw worden verschillende bouwblokken gebruikt om de juiste werking te bekomen. Iedere ronde maakt gebruik van de bouwblokken Sine, Shift, Splits en de functie die in deze ronde gebruikt wordt. Voor de eerste 16 rondes wordt functie F gebruikt, in de volgende 16 rondes wordt gebruik gemaakt van functie G. De volgende 16 rondes gebruiken functie H en in de laatste

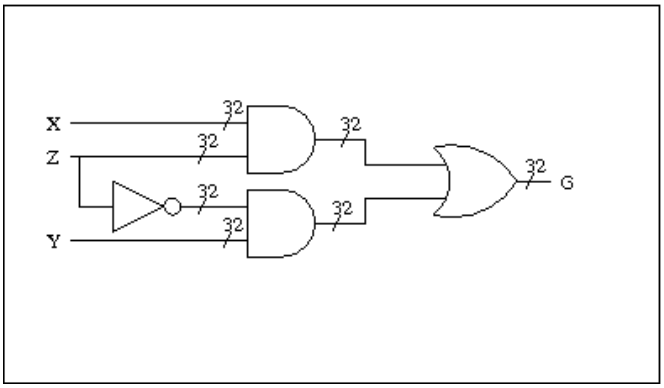
16 rondes wordt functie I gebruikt. In Figuur 3.7 wordt de opbouw van Ronde 1 duidelijk gemaakt.
De functies F, G, H en I maken gebruik van de ingangen *b*, *c* en *d* om een uitgang van 32 bits te berekenen. Splits selecteert de juiste 32 bits van de boodschap a.d.h.v. de ingang *k*. Sine zal de juiste waarde naar buiten sturen m.b.v. de ingang *si*. De uitgangen van deze 3 bouwblokken en *a* worden bij elkaar opgeteld en aan de ingang van Shift aangeboden. Shift zal deze 32 bits cyclisch verschuiven. Het aantal verschuivingen is afhankelijk van *s*. De uitgang van Shift wordt met *b* opgeteld en vormt de uitgang van de ronde.
De opbouw van de functies F, G, H en I worden schematisch weergegeven in respectievelijk Figuur 3.8, Figuur 3.9, Figuur 3.10 en Figuur 3.11.



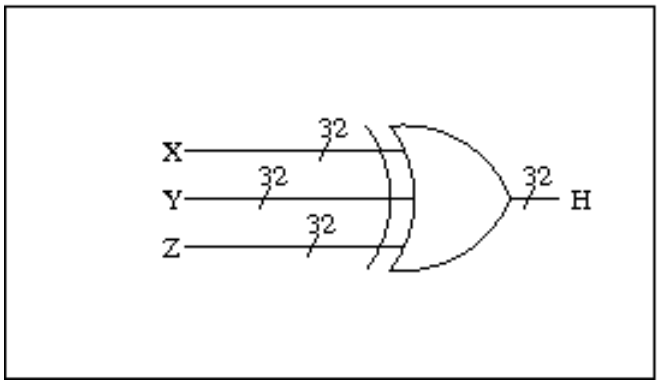
Figuur 3.7: Archtiectuur van Ronde 1



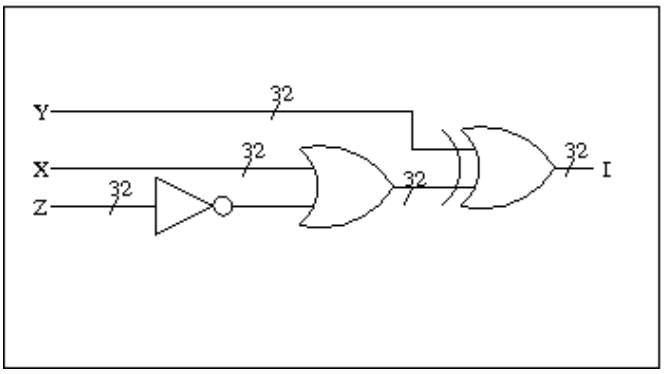
Figuur 3.8: Functie F



Figuur 3.9: Functie G



Figuur 3.10: Functie H



Figuur 3.11: Functie I

3.4.2.3 Hashreg-bouwblok

Dit bouwblok slaat de hashwaarde die in Calc berekend wordt op in een register van 128 bits en genereert het uitgangssignaal *outputready*.

3.4.3 Output-bouwblok

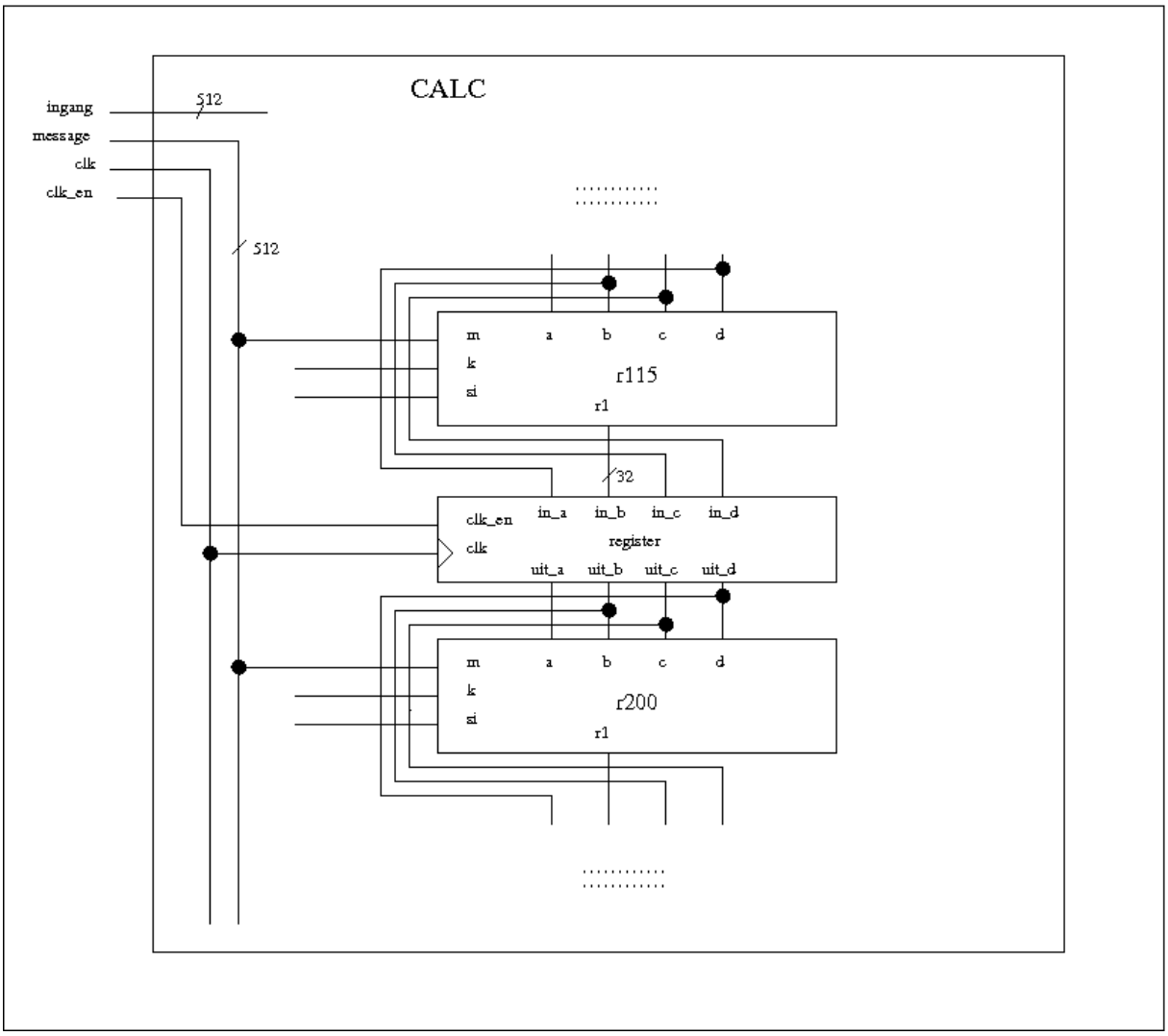
Dit bouwblok bestaat uit een register van 128 bits waarin de hashwaarde wordt opgeslagen. Wanneer *shift_data_out* hoog wordt zal de hashwaarde in 4 stukken van 32 bits op de uitgangspinnen gezet worden. De 32 meest beduidende bits worden eerst uitgestuurd.

3.5 Maximale werkingsfrequentie

Niet enkel de goede werking van het algoritme is van belang maar ook de maximale klokfrequentie waarop de implementatie kan werken. De maximale frequentie is afhankelijk van de grootste propagatievertraging tussen twee geheugenelementen in de implementatie, ook wel het kritisch pad genoemd. Door het kritisch pad te zoeken en dit op te splitsen door één of meerdere registers toe te voegen zal de grootste propagatievertraging kleiner worden, waardoor er met een hogere klokfrequentie gewerkt kan worden. Hier staat tegenover dat het ontwerp een grotere oppervlakte zal gebruiken omdat er registers toegevoegd worden. Ook zullen er meerdere klokperiodes nodig zijn om de hashwaarde te berekenen. De totale tijd die nodig is om een hashwaarde te berekenen blijft echter ongeveer gelijk. Het toevoegen van registers heeft meestal tot doel om pipelining toe te voegen aan het systeem. In ons geval wordt dit echter gedaan met het oog op het inbedden van de implementatie in een groter systeem dat op een hogere frequentie werkt. Aangezien de tussentijdse hashwaarde opgeslagen moet worden is het in deze implementatie onmogelijk om pipelining toe te passen.
In ons ontwerp is het bouwblok Calc volledig combinatorisch opgebouwd, terwijl de meeste andere bouwblokken sequentiële processen zijn met een klein kritisch pad. Dit wil zeggen dat het pad met de grootste propagatievertraging zich in het Calc-bouwblok bevindt.
De eerste versie van het MD5-algoritme is een implementatie zonder registers in het Calc-bouwblok, zoals hierboven besproken werd. Omdat het kritisch pad bij deze implementatie heel groot is zullen we dit proberen te verkleinen.

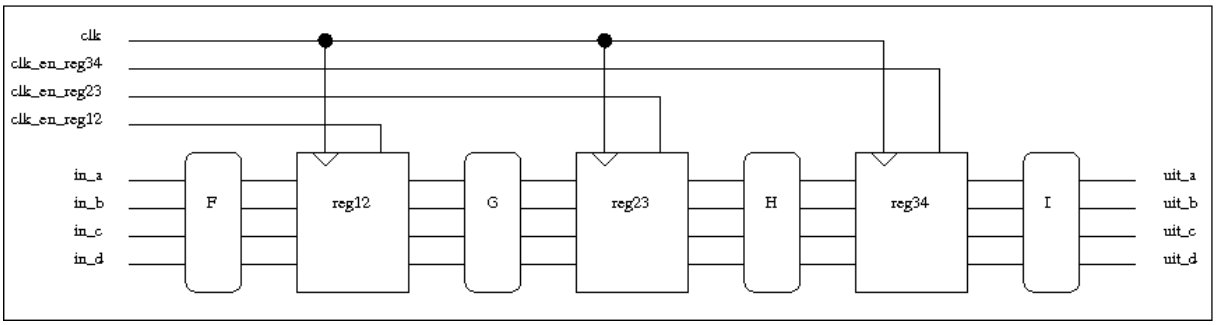
3.5.1 MD5 versie 2 (MD5v2)

Door het pad met de grootste propagatievertraging op te splitsen in twee gelijke stukken, met behulp van een register, zal het kritisch pad ongeveer halveren. Zo ontstaan er twee paden die opnieuw opgesplitst kunnen worden om de propagatievertraging verder te verkleinen.
Bij de tweede versie van de implementatie van het MD5-algoritme werden er drie registers bijgevoegd. Het Calc-bouwblok bestaat uit 64 rondes die per 16 rondes een andere logische functie gebruiken (F,G,H en I). Deze 64 rondes hebben we opgesplitst in 4 delen, na 16 rondes werd er een register toegevoegd. In Figuur 3.12 wordt de plaats van register reg12 duidelijk gemaakt. Deze implementatie kan met een klokfrequentie werken die ongeveer 4 maal zo hoog is als de klokfrequentie van de eerste versie. Dit kan omdat het combinatorisch gedeelte in 4 delen opgesplitst wordt.

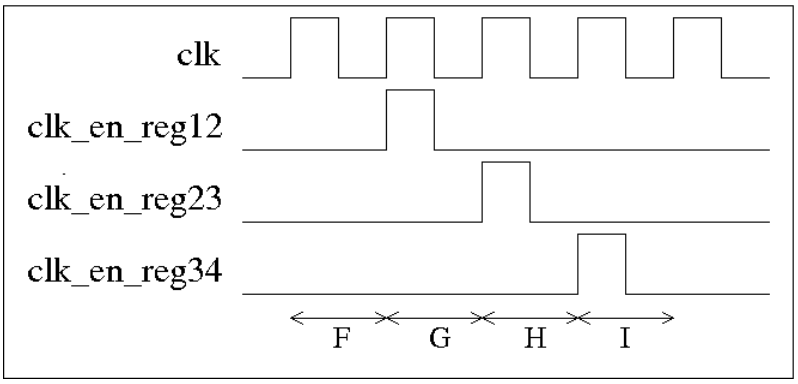


Figuur 3.12: Plaats van register reg12

Het is belangrijk dat ieder register op het juiste moment zijn ingang opslaat. Elk register heeft een *clk_en*-ingang. Enkel wanneer dit signaal hoog is en er een stijgende klokflank is zal de ingang opgeslagen worden. Elk register krijgt een ander *clk_en*-signaal. De *clk_en*-signalen voor de verschillende registers worden in het Hash-bouwblok aangemaakt. Het eerste register, dat na 16 rondes data opslaat, gebruikt *clk_en_reg12*. Het volgende register dat 16 rondes later data opslaat, gebruikt *clk_en_reg23*. Het derde register slaat weer 16 rondes later de data op, en zal hiervoor *clk_en_reg_34* gebruiken. Dit wordt weergegeven in Figuur 3.13. De timing van de signalen wordt duidelijk gemaakt in Figuur 3.14.



Figuur 3.13: Plaats van de registers binnen de tweede versie van MD5



Figuur 3.14: Timing van de clk_en-signalen

3.5.2 MD5 versie 3 (MD5v3)

Omdat de maximale klokfrequentie die de tweede versie van de implementatie van het MD5-algoritme kan behalen mogelijk nog niet hoog genoeg is voor sommige applicaties, zullen we meer registers toevoegen om het combinatorisch gedeelte verder op te splitsen in kleinere stukken. Zo zal men de grootste propagatievertraging verder verlagen. Om de 4 rondes staat er een register. Elk register wordt weer aangestuurd met een *clk_en*-signaal. Deze signalen worden aangemaakt binnen het Hash-bouwblok.

3.5.3 MD5 versie 4 (MD5v4)

In de laatste implementatie van MD5, namelijk versie 4, bevindt er zich na elke ronde een register. Deze implementatie zal dan ook met de hoogste klokfrequentie kunnen werken. Ook hier worden de nodige *clk_en*-signalen binnen het Hash-bouwblok aangemaakt.

4 Het Whirlpoolalgoritme

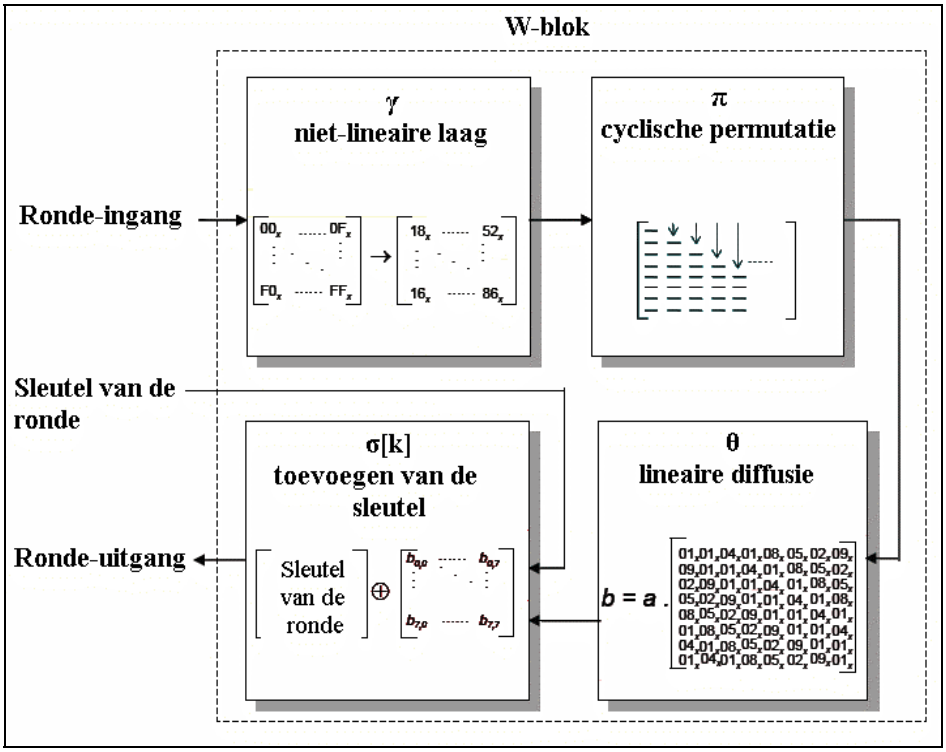
In dit hoofdstuk zal het Whirlpoolalgoritme in detail besproken worden.

4.1 Ontstaan

Het Whirlpoolalgoritme werd ontwikkeld door Vincent Rijmen en Paula S. L. M. Barreto. Van het Whirlpoolalgoritme bestaan drie versies waarvan wij de derde versie zullen implementeren.
Dit algoritme werd in 2004 in de ISO-standaard opgenomen (ISO/IEC 10118-3:2004).[10]
Het Whirlpoolalgoritme is genoemd naar een melkweg in Canes Venatici.[11]

4.2 Het algoritme

Het Whirlpoolalgoritme berekent van een boodschap met maximum lengte 2^{256} een hashwaarde van 512 bits. Het hart van het algoritme zit in het W-blok, dat opgesplitst kan worden in 4 blokken, namelijk de blokken γ of de niet-lineaire laag (S-blok), π of de cyclische permutatie, θ of de lineaire diffusie en $\sigma[k]$ of het toevoegen van de sleutel, zoals in Figuur 4.1 weergegeven wordt. Binnen het algoritme wordt gebruik gemaakt van een 8*8-matrix, ieder element is een byte.

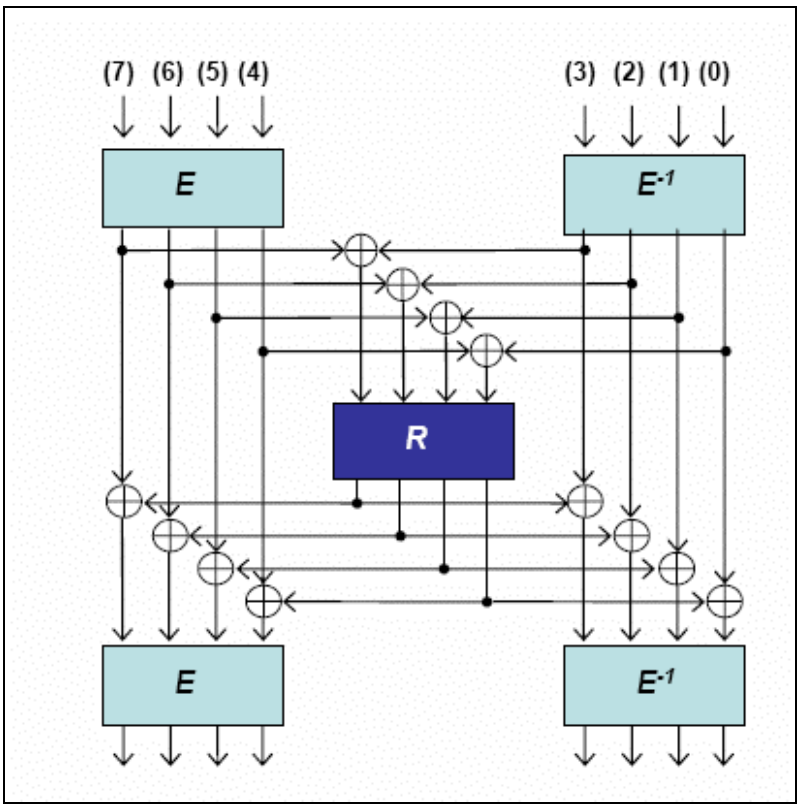


Figuur 4.1: Het W-blok

[10] INTERNATIONAL ORGANIZATION OF STANDARDIZATION, *ISO/IEC 10118-3:2004*, online, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=39876&ICS1=35&ICS2=40&ICS3=, 1 december 2005.
[11] BARRETO, S. L. M. P., *The Whirlpool Hash Function*, online, http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html, 1 december 2005.

4.2.1 γ, de niet-lineaire laag

De niet-lineaire laag bestaat uit de toepassing van het niet-lineaire substitutieblok: het S-blok. Dit S-blok wordt schematisch weergegeven in Figuur 4.2.



Figuur 4.2: Het S-blok[12]

u	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
***E*(*u*)**	**1**	**B**	**9**	**C**	**D**	**6**	**F**	**3**	**E**	**8**	**7**	**4**	**A**	**2**	**5**	**0**

Tabel 4.1: Omzettingstabel van het E-miniblok[11]

u	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
***E*$^{-1}$(*u*)**	**F**	**0**	**D**	**7**	**B**	**E**	**5**	**A**	**9**	**2**	**C**	**1**	**3**	**4**	**8**	**6**

Tabel 4.2: Omzettingstabel van het E^{-1}-miniblok[11]

u	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
R(u)	**7**	**C**	**B**	**D**	**E**	**4**	**9**	**F**	**6**	**3**	**8**	**A**	**2**	**5**	**1**	**0**

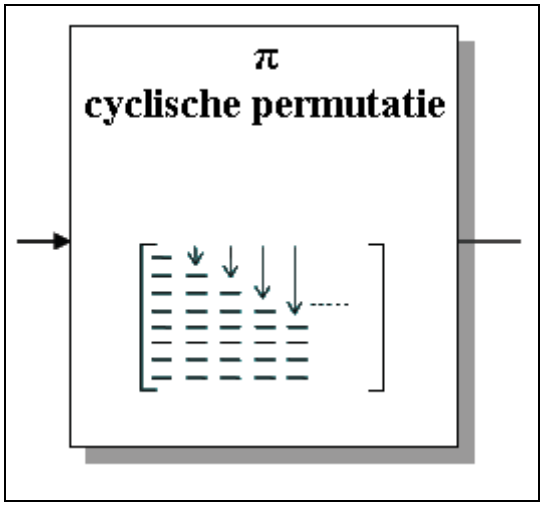
Tabel 4.3: Omzettingstabel van het R-miniblok[11]

[12] LANO, J, *Hash function workshop slides*, online, www.ecrypt.eu.org/stvl/hfw/McLoone.pdf, 1 december 2005.
[11] BARRETO, S. L. M. P., The Whirlpool Hash Function, online, http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html, 1 december 2005.

Het S-blok krijgt als ingang 8 bits en geeft als uitgang een niet-lineaire gesubstitueerde waarde van de ingang die ook weer 8 bits bedraagt. De minst beduidende bit wordt helemaal rechts aan de ingang aangeboden (0), de meest beduidende helemaal links (7). Deze bits worden aan de hand van de tabel omgezet.
Stel dat men de binaire waarde 0011 1000 (wat hexadecimaal overeenkomt met 38) aan de ingang legt. 0011 wordt aangeboden aan het E-miniblok, terwijl 1000 aangeboden wordt aan het E^{-1}-miniblok. Uit het E-miniblok en E^{-1}-miniblok krijgen we respectievelijk de waardes 1100 en 1001 (hexadecimaal C en 9). Dit volgt uit Tabel 4.1 en Tabel 4.2. De uitgangen van het E- en E^{-1}-miniblok worden aan de ingangen van exclusieve ofpoorten aangeboden, zoals Figuur 4.2 aangeeft. De uitgangen van deze exclusieve ofpoorten worden verbonden met de ingangen van het R-miniblok. Als men met vorig voorbeeld verder rekent, gaat men 0011 en 1000 aanleggen aan de exclusieve ofpoorten, wat als uitgangen 1011 geeft (hexadecimaal B). Uit het R-miniblok krijgen we dan de waarde 1010 (hexadecimaal A), dit volgt uit Tabel 4.3. Deze uitgangen van het R-miniblok moeten aan de ingangen van exclusieve ofpoorten aangelegd worden, samen met de uitgangen van het bovenste E- en E^{-1}-miniblok. De uitgangen van deze exclusieve ofpoorten dienen als ingangen voor het onderste E- en E^{-1}-miniblok, respectievelijk 0110 en 0011. Wanneer we ook dit weer omzetten, krijgen we uiteindelijk de niet-lineaire gesubstitueerde waarde van de ingang, 0011 1000, namelijk 1111 0111 (hexadecimaal F7).
Dit S-blok wordt toegepast op elke waarde uit de 8*8-matrix van de ronde-ingang.

4.2.2 π, de cyclische permutatie

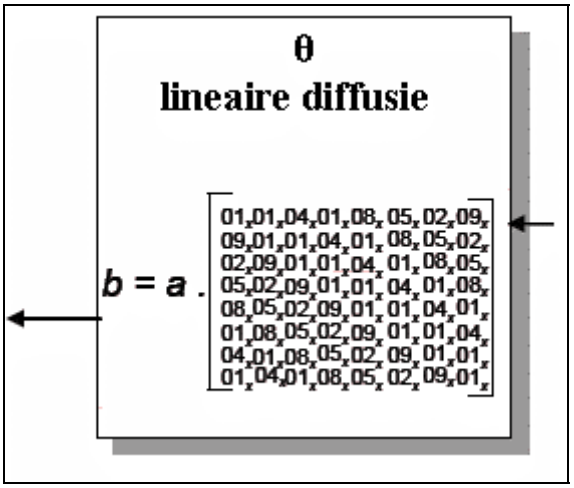
De cyclische permutatie heeft als ingang de uitgang van het γ-blok. De cyclische permutatie gaat op elke kolom van de 8*8-matrix een circulaire schuifactie uitvoeren. Het aantal keer dat deze kolom naar beneden geschoven wordt is afhankelijk van de kolom: kolom j wordt j keer naar beneden geschoven. Dit betekent dat de eerste kolom met index 0 niet verschoven wordt, en dat de laatste kolom met index 7 met 7 plaatsen verschoven wordt. Dit wordt weergegeven in Figuur 4.3.



Figuur 4.3: Het π-blok

4.2.3 θ, de lineaire diffusie

De uitgang van het π-blok vormt hier de ingang. In dit blok wordt de ingangsmatrix vermenigvuldigd met een matrix met een vaste waarde, zoals weergegeven in Figuur 4.4. Deze vermenigvuldiging gebeurt in een eindig veld. Alle getallen van beide matrices zijn elementen van het eindig veld. Het eindresultaat is opnieuw een 8*8-matrix, waarvan alle waardes terug binnen het eindig veld liggen.
Een eindig veld kent de optelling, aftrekking, vermenigvuldiging en deling (niet door nul).[13] In tegenstelling tot de wiskunde die wij in het dagelijks leven gebruiken, is de verzameling van elementen in een eindig veld gesloten. Dit wil zeggen dat het aantal elementen van een eindig veld eindig is, en dat een bewerking op twee elementen terug een element geeft uit dit eindig veld.
Een eindig veld bestaat alleen met als orde een priemgetal, of een priemgetal tot een macht verheven.[14] De bewerkingen zullen uitgevoerd worden binnen het eindig veld met 16 lichamen (4 bits). Dit wordt als volgt geschreven: $GF(2^8)$. Wanneer men beide lichamen vermenigvuldigt krijgt men een element dat niet tot het eindig veld behoort. Om dit resultaat terug binnen het eindig veld te laten behoren moet men het resultaat omzetten aan de hand van de irreduceerbare veelterm $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$. Alle waardes die groter zijn dan x^7 moeten omgezet worden aan de hand van deze veelterm.
Uit de som van de veeltermen $x^8 + x^4 + x^3 + x^2 + 1 = 0$ en $x^4 + x^3 + x^2 + 1 = x^4 + x^3 + x^2 + 1$ kan men de veelterm $x^8 = x^4 + x^3 + x^2 + 1$ halen. Door waarden die groter zijn dan x^7 te vervangen aan de hand van $x^8 = x^4 + x^3 + x^2 + 1$, bekomt men het resultaat, dat opnieuw binnen het eindig veld ligt.

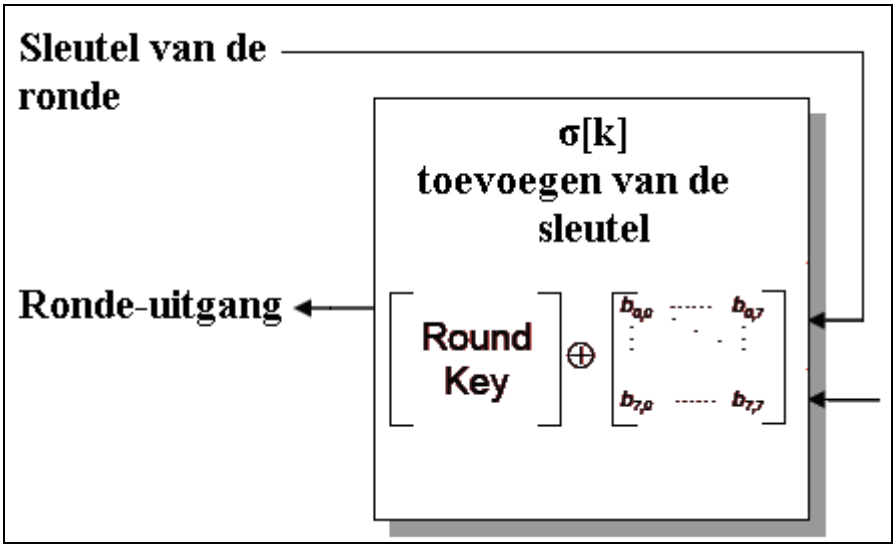


Figuur 4.4: Het θ-blok

4.2.4 σ[k], het toevoegen van de sleutel

Het σ[k]-blok heeft 2 ingangen. Eén ingang is de uitgang van het θ-blok. De andere ingang is de sleutel van de ronde (dit is ook een 8*8-matrix). Beide ingangen worden bit per bit aangeboden aan de ingang van een exclusieve ofpoort. De uitgang van deze exclusieve ofpoort wordt dan op de overeenkomstige plaats in de matrix geplaatst die de uitkomst weergeeft. Dit bouwblok wordt weergegeven in Figuur 4.5.

[13] WIKIPEDIA, *Eindig lichaam*, online, http://nl.wikipedia.org/wiki/Eindig_lichaam, 1 maart 2006.
[14] WIKIPEDIA, *Finite field*, online, http://en.wikipedia.org/wiki/Finite_Field, 1 maart 2006.



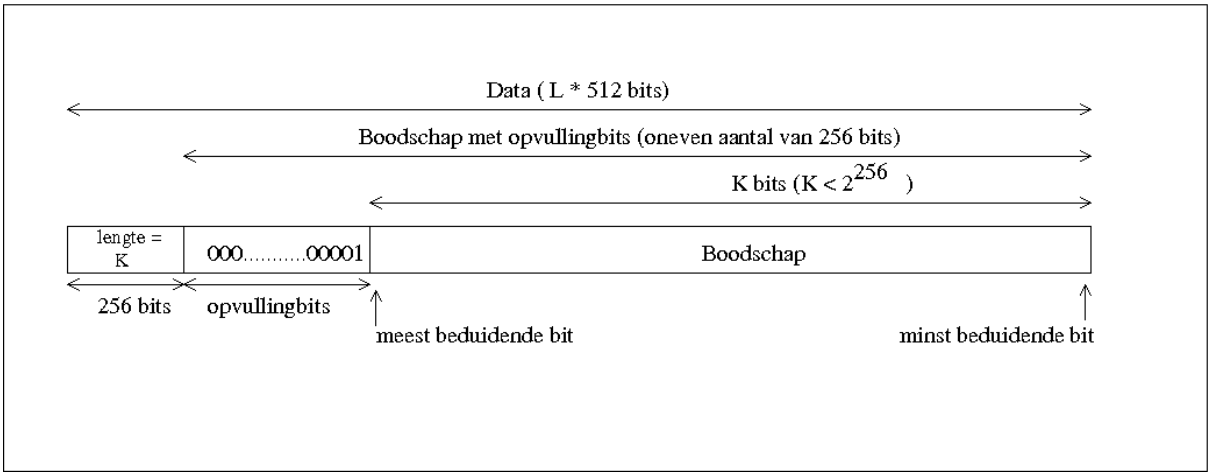
Figuur 4.5: Het σ[k]-blok

Deze 4 blokken, γ, π, θ en σ[k], vormen samen het W-blok.

4.3 Het berekenen van de hashwaarde

4.3.1 Toevoegen van opvullingbits

De boodschap mag een maximale lengte van 2^{256} bits hebben. Aan deze boodschap worden aan de kant van de meest beduidende bit opvullingbits toegevoegd. De minst beduidende bit van de opvullingbits is een 1, al de andere opvullingbits zijn 0'en. Er worden zo weinig mogelijk 0'en toegevoegd, zodat de lengte van de boodschap met opvullingbits nu een oneven veelvoud van 256 bits bedraagt. Vervolgens voegen we aan deze boodschap met opvullingbits de lengte toe als een 256 bitsgetal. De meest beduidende bit van de lengte is ook de meest beduidende bit van de data. Deze lengte geeft de lengte van de boodschap in bits weer (de boodschap zonder opvullingbits). De data zijn nu een veelvoud van 512 bits. Voor het berekenen van de hashwaarde met het Whirlpoolalgoritme worden de data opgesplitst en verwerkt per 512 bits. Dit wordt weergegeven in Figuur 4.6.



Figuur 4.6: Het toevoegen van opvullingbits (Whirlpoolalgoritme)

4.3.2 Verwerken van een datablok van 512 bits

Een datablok van 512 bits, wordt aangeboden aan de ingang van een exclusieve ofpoort samen met de vorige tussentijdse hashwaarde. Bij het verwerken van de eerste 512 bits is er nog geen tussentijdse hashwaarde, en gebruikt men een matrix bestaande uit allemaal 0'en. De uitgang van deze exclusieve ofpoort wordt opgeslagen in een 8*8-matrix, waarbij elke waarde van de matrix een byte is. Deze matrix is de ingang van het W-blok. Per blok van 512 bits van de data wordt het W-blok 10 keer uitgevoerd. Bij de eerste ronde is de ronde-ingang van het γ-blok de matrix met de 512 bits waarvan we de hashwaarde gaan berekenen. Bij de volgende 9 rondes is deze ingang gelijk aan de uitgang van het σ[k]-blok van de vorige ronde.
Na 10 rondes wordt de tussentijdse hashwaarde berekend. Deze is gelijk aan de uitgang van een exclusieve ofpoort met drie ingangen, namelijk het blok van 512 bits van de data, de tussentijdse hashwaarde van de vorige ronde en de uitgang van het σ[k]-blok van de laatste ronde. De tussentijdse hashwaarde van de laatste 512 bits vormt ook de uiteindelijke hashwaarde van de hele boodschap.

4.3.3 Genereren van de sleutel

Binnen Sleutel wordt ook het W-blok gebruikt om de sleutel te genereren. Als ronde-ingang krijgt het W-blok binnen Sleutel de tussentijdse hashwaarde van de vorige 512 bits. Indien men de eerste 512 bits verwerkt bestaat er nog geen tussentijdse hashwaarde en krijgt het W-blok allemaal 0'en aan de ronde-ingang aangeboden. De sleutel van de ronde van het W-blok is een vaste waarde, deze ingang is afhankelijk van de ronde. De onderste 7 rijen van deze matrix zijn steeds 0; enkel de eerste rij bevat data verschillend van 0. Deze data worden gegenereerd met het S-blok. De ingang van het S-blok is:

[8(r-1)+j] met r: rondenummer ($1 \leq r \leq 10$)
j: kolomnummer ($0 \leq j \leq 7$)

Als men bijvoorbeeld naar de eerste ronde kijkt zal het element op de eerste rij en derde kolom, berekend worden met het S-blok met als ingang 8(1-1)+2 = 2. Wanneer we 02 (de hexadecimale waarde van 2) aanleggen aan de ingang van het S-blok krijgen we als uitgang C6 (hexadecimaal). De matrix van de eerste ronde ziet er uit zoals in Figuur 4.7.

```
18 23 C6 E8 87 B8 01 4F
00 00 00    …  …  …    00


00 00 00    …  …  …    00
```

Figuur 4.7: Sleutelingang van de eerste ronde voor het W-blok binnen het sleutelblok

5 De implementatie van het Whirlpoolalgoritme

Dit hoofdstuk beschrijft de implementatie van het Whirlpoolalgoritme.

5.1 De ingangssignalen

Bij de implementatie worden volgende ingangssignalen gebruikt:

- *clk*: dit is de klokfrequentie waarop de gehele implementatie zal werken.
- *data_in*: via deze 32 lijnen wordt de boodschap ingelezen.
- *shift_data_in*: dit signaal duidt aan dat er 512 nieuwe bits ingelezen kunnen worden.
- *rst*: als dit signaal hoog wordt, worden alle registers naar hun beginwaarde (0) gezet.
- *start*: met behulp van dit signaal wordt aangeduid of een boodschap uit meerdere blokken van 512 bits bestaat.
- *shift_data_out*: dit signaal geeft aan dat de hashwaarde naar buiten gestuurd kan worden.

5.2 De uitgangssignalen

Bij de implementatie worden volgende uitgangssignalen gebruikt:

- *data_out*: via deze 32 lijnen wordt de hashwaarde naar buiten gestuurd.
- *outputready*: dit signaal geeft aan dat de hashwaarde berekend is.

5.3 Afspraken

Om een goede vergelijking tussen beide algoritmes te kunnen maken hebben we er ook hier voor gekozen om het toevoegen van de opvullingsbits door software te laten uitvoeren. Hierdoor zullen de data steeds een veelvoud van 512 bits bedragen. De software zal de data per 512 bits aanbieden aan de ingang van het hardwareblok. Ook zal deze software ervoor zorgen dat de andere ingangssignalen op het juiste tijdstip worden aangeboden en dat de uitgangssignalen op het juiste tijdstip worden uitgelezen zodat er geen informatie verloren kan gaan.

5.3.1 Het inlezen van de boodschap

Om de boodschap in te lezen worden volgende signalen gebruikt: *data_in*, *shift_data_in*, *start* en *outputready*. Via *data_in* zal de boodschap ingelezen worden. Aangezien een boodschap 512 bits telt en er voor *data_in* 32 bits voorzien zijn zal het inlezen van een boodschap 16 klokperiodes in beslag nemen.

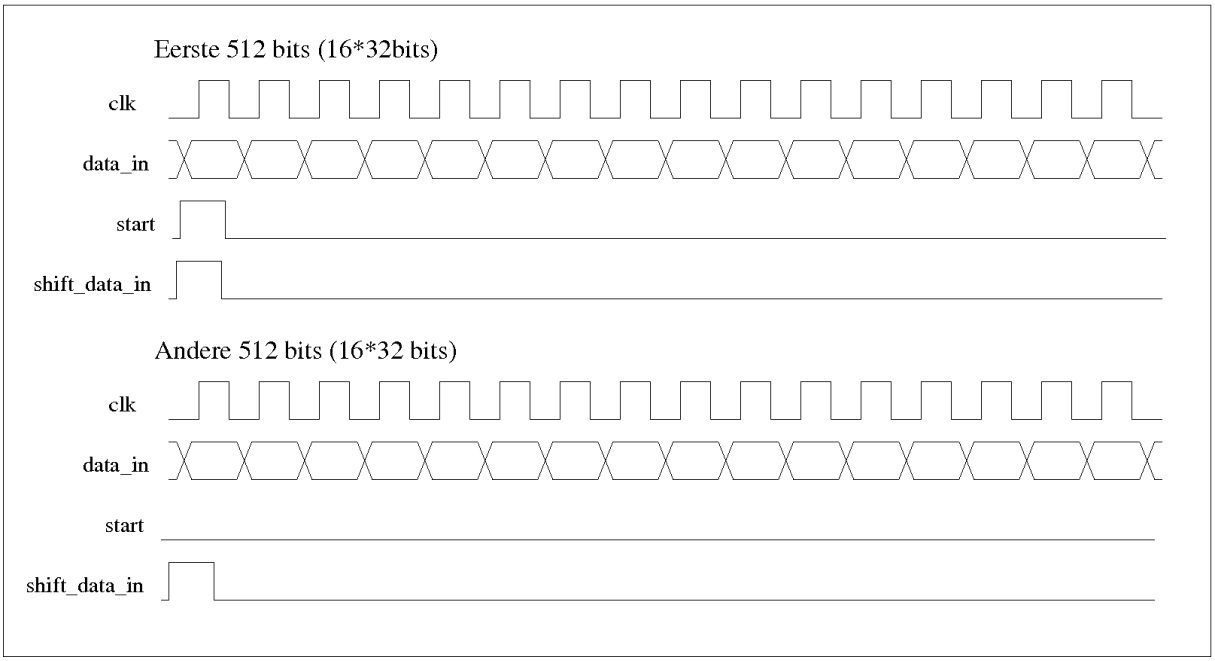
5.3.1.1 Eerste 512 bits inlezen

Op het moment dat *shift_data_in* en *start* hoog worden moeten de 32 minst beduidende bits van de boodschap aan *data_in* aangeboden worden. Op de stijgende klokflank zullen, terwijl *shift_data_in* en *start* hoog zijn, deze 32 minst beduidende bits van de boodschap

ingelezen en opgeslagen worden in een register. Na het inlezen van de eerste 32 bits worden de ingangen *shift_data_in* en *start* terug laag gemaakt. Hierna zal bij iedere stijgende klokflank opnieuw *data_in* ingelezen worden, waar iedere keer de 32 volgende bits van de boodschap aangeboden worden. Bij de 16^{de} stijgende klokflank worden de 32 meest beduidende bits van de boodschap ingelezen. Hierna zijn 512 bits ingelezen, en kan de hashwaarde van deze 512 bits berekend worden. Het bovenste deel van Figuur 5.1 geeft dit weer.

5.3.1.2 Volgende 512 bitsblokken inlezen

Een eventueel volgend deel van de totale boodschap kan pas ingelezen worden op het moment dat het uitgangssignaal *outputready* hoog geworden is. Dit geeft aan dat de vorige 512 bits verwerkt zijn. Het inlezen gebeurt op dezelfde manier als het inlezen van de eerste 512 bits maar het ingangssignaal *start* zal laag blijven. Het onderste deel van Figuur 5.1 geeft dit weer.

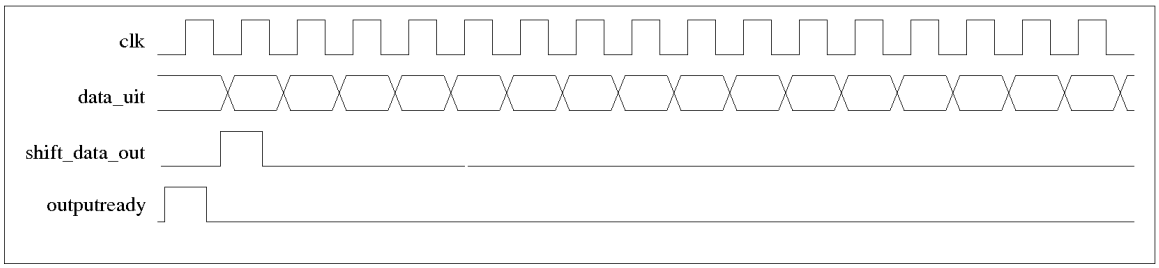


Figuur 5.1: Timing van het inlezen

5.3.2 Het uitlezen van de hashwaarde

Om de hashwaarde uit te lezen worden volgende signalen gebruikt: *data_out*, *shift_data_out* en *outputready*. Via *data_out* zal de hashwaarde uitgestuurd worden. Aangezien een hashwaarde 512 bits groot is en er voor *data_out* 32 bits voorzien zijn zal het uitlezen van de boodschap 16 klokperiodes in beslag nemen. Op het moment dat de hashwaarde klaar staat om uitgelezen te worden zal het uitgangssignaal *outputready* gedurende 1 klokperiode hoog worden. Hierna zal de software het ingangssignaal *shift_data_out* gedurende 1 klokperiode hoog maken, wat aanduidt dat de software klaar is om de hashwaarde uit te lezen. Dit wordt weergegeven in Figuur 5.2. De 32 minst beduidende bits van de hashwaarde worden aan de uitgang aangeboden bij de stijgende klokflank terwijl *shift_data_out* hoog is. Hierna worden bij iedere stijgende klokflank de volgende 32 bits

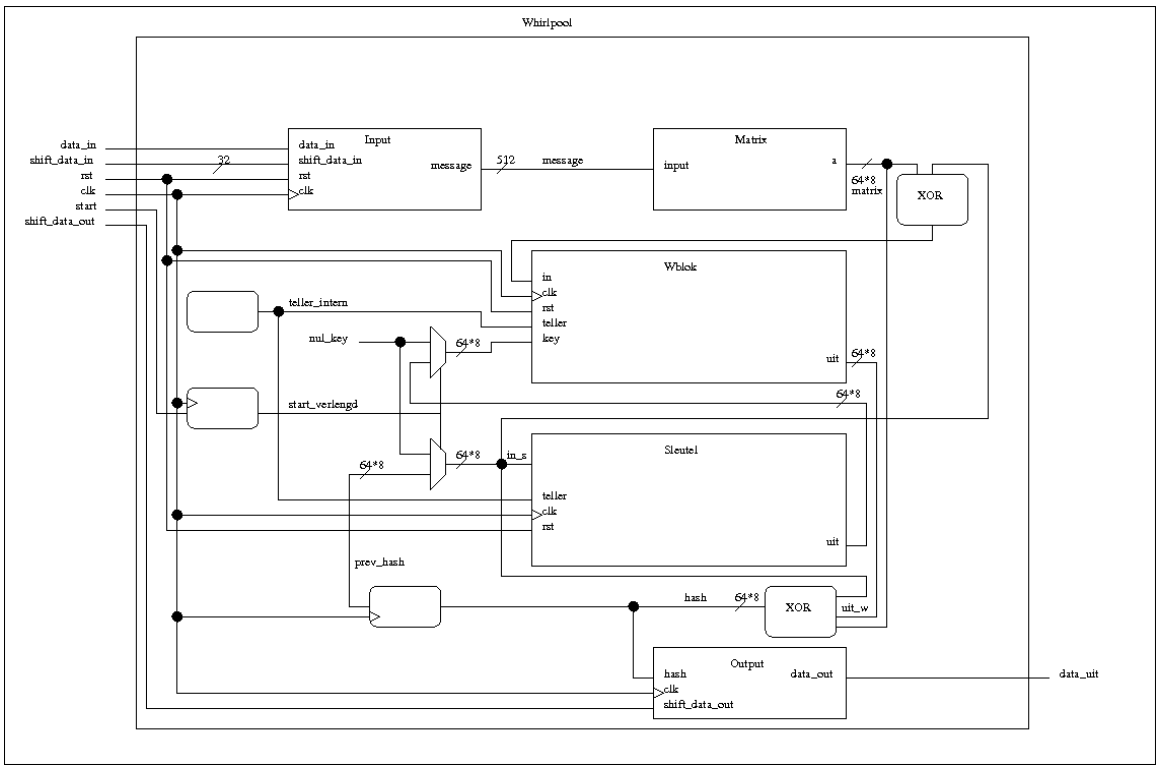
aangeboden. Bij de 16de stijgende klokflank is de waarde aan de uitgang gelijk aan de 32 meest beduidende bits.



Figuur 5.2: Timing van het uitlezen

5.4 Whirlpool-bouwblok

Alle ingangssignalen worden aangelegd aan de ingang van het Whirlpool-bouwblok en na verwerking zal Whirlpool de uitgangssignalen aansturen. Whirlpool is opgebouwd uit 5 bouwblokken, namelijk Input, Matrix, Wblok, Sleutel en Output, zoals Figuur 5.3 aangeeft. Whirlpool zorgt ervoor dat elk bouwblok de juiste ingangssignalen krijgt. Deze ingangssignalen kunnen zowel controlesignalen zijn als signalen die van andere bouwblokken binnen Whirlpool komen. Nadat Input 512 bits heeft ingelezen, worden deze aangeboden aan Matrix. Hierna wordt deze uitgang samen met de vorige tussentijdse hashwaarde (dit is ook de ingang van Sleutel) aangeboden aan de ingang van een exclusieve ofpoort. De uitgang van deze exclusieve ofpoort is, samen met de uitgang van Sleutel, verbonden met Wblok. Na het berekenen van de hashwaarde wordt deze aangelegd aan de ingang van Output.



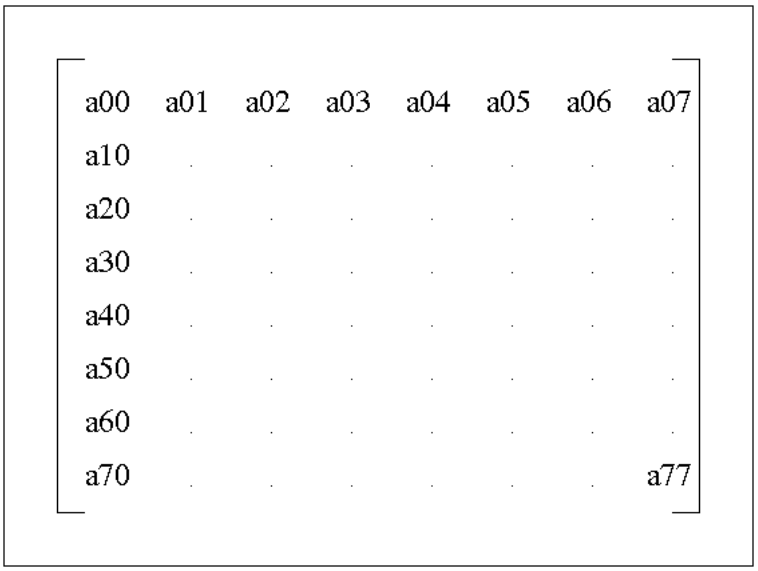
Figuur 5.3: Architectuur van het Whirlpoolbouwblok

5.4.1 Input-bouwblok

Dit bouwblok werd ook voor de implementatie van het MD5-algoritme gebruikt en werd reeds besproken in "3.4.1 Input-bouwblok". Voor de volledigheid wordt dit bouwblok hier nogmaals kort besproken.
Dit bouwblok bestaat uit een register van 512 bits waarin de boodschap opgeslagen wordt. Bij iedere stijgende klokflank worden eerst de 32 bits van *Data_in* ingelezen in de 32 meest beduidende bits van dit register. Bij de volgende stijgende klokflank gebeurt er een schuifactie die ervoor zorgt dat de 480 meest beduidende bits 32 plaatsen verschoven worden. Hierdoor wordt er ruimte gecreëerd voor de volgende 32 bits die gelijktijdig ingelezen worden.
Als de volledige boodschap van 512 bits ingelezen is wordt het signaal *ready* hoog gemaakt. Aan de hand van dit *ready*-signaal worden verschillende tellers gesynchroniseerd die nodig zijn voor de goede werking van het algoritme.

5.4.2 Matrix-bouwblok

Binnen Matrix wordt een boodschap van 512 bits omgezet in een 8*8-matrix. Ieder element van deze matrix bestaat uit een byte (8 bits). Deze omzetting is niet expliciet nodig voor de goede werking van het algoritme, maar zorgt ervoor dat de implementatie duidelijker en eenvoudiger wordt. De matrix bevat alle elementen van a00 tot a77, waarbij het eerste getal de rij en het tweede getal de kolom aanduidt. Zie Figuur 5.4.

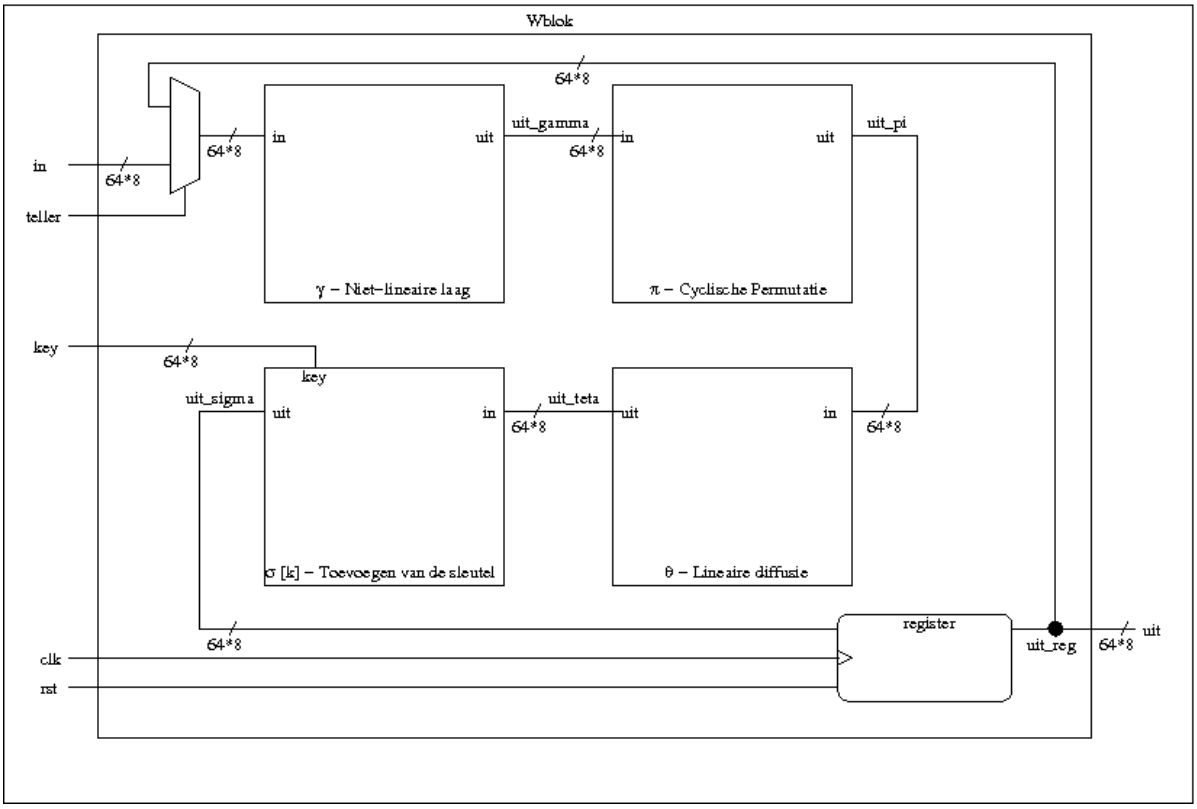


Figuur 5.4: Matrix met zijn elementen

5.4.3 Wblok-bouwblok

Dit bouwblok bevat volgende bouwblokken: Gamma, Pi, Teta en Sigma. Per 512 bits die verwerkt worden zal het Wblok 10 keer doorlopen worden. De multiplexer zorgt ervoor dat Gamma de juiste ingang krijgt. De eerste ronde zijn dit de 512 bits die Wblok als ingang krijgt, de andere rondes krijgt Gamma de uitgang van Sigma. Om ervoor te zorgen dat de 10 rondes niet als één combinatorisch geheel geïmplementeerd worden, maar dat dezelfde hardware 10 keer doorlopen wordt, werd er een register tussen de uitgang van Sigma en de

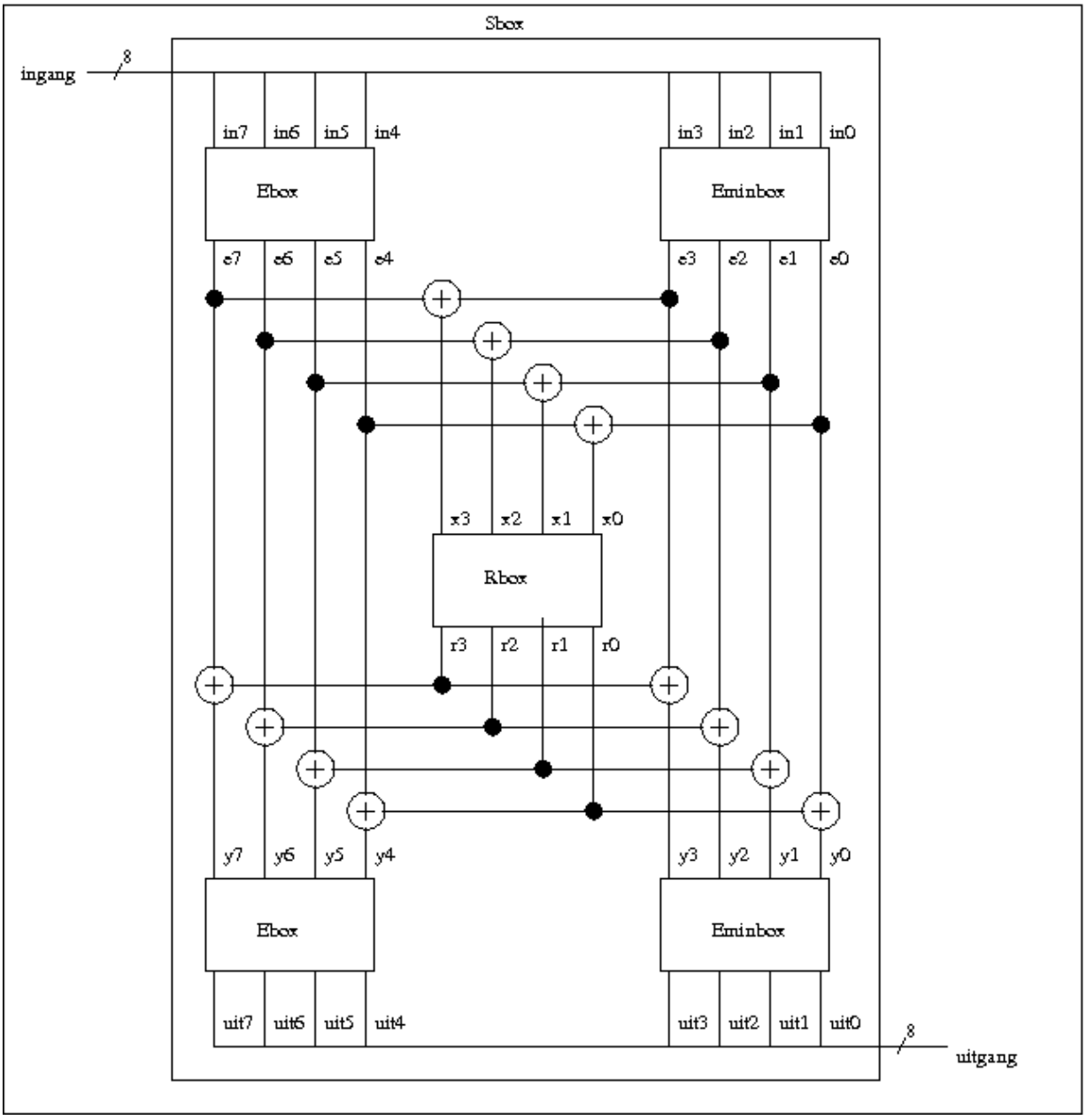
ingang van Gamma geplaatst. Dit zorgt ervoor dat de implementatie beduidend kleiner is. De opbouw van het Wblok wordt weergegeven in Figuur 5.5.



Figuur 5.5: Architectuur van het Wblokbouwblok

5.4.3.1 Gamma-bouwblok

Dit bouwblok wordt ook wel Sbox genoemd en is opgebouwd uit 3 bouwblokken, namelijk Ebox, Eminbox en Rbox, zoals weergegeven wordt in Figuur 5.6.
De 4 minst beduidende bits van de ingang van Sbox worden aangeboden aan een Eminbox, met in0 als minst beduidende bit. De 4 meest beduidende bits van de ingang worden aangeboden aan een Ebox, met in7 als meest beduidende bit. Deze uitgangen worden bit per bit aangeboden aan de ingang van een exclusieve ofpoort, namelijk e7 en e3, e6 en e2, e5 en e1, e4 en e0. De uitgangen van deze exclusieve ofpoorten, x3, x2, x1 en x0 worden aan de Rbox aangeboden. De uitgangen van deze Rbox worden weer aangeboden aan een exclusieve ofpoort, samen met de signalen e7 tot en met e0, zoals in Figuur 5.6 weergegeven wordt. De uitgangen van deze exclusieve ofpoorten, y7 tot en met y4, worden aangeboden aan een Ebox en y3 tot en met y0 aan een Eminbox. De uitgang van Ebox en Eminbox vormen de uitgang, met uit7 als meest beduidende bit en uit0 als minst beduidende bit.



Figuur 5.6: Architectuur van Sbox

5.4.3.2 Pi-bouwblok

Dit bouwblok heeft als ingang de uitgang van Gamma en zorgt ervoor dat de elementen uit de ingangsmatrix op de juiste manier verschoven worden. Dit wil zeggen dat kolom j met j plaatsen verschoven wordt, zoals beschreven in "4.2.2 π, de cyclische permutatie".

5.4.3.3 Teta-bouwblok

In dit bouwblok worden 2 matrices met elkaar vermenigvuldigd, namelijk de ingangsmatrix en een constante matrix waarbij de ingangsmatrix de uitgang van Pi is.
Wanneer 2 matrices, a en c, met elkaar vermenigvuldigd worden krijgt men een matrix d als

uitkomst, zoals in Figuur 5.7 voorgesteld is. De waardes van deze matrix worden als volgt berekend:

d00 = (a00 * c00) + (a01 * c10) + (a02 * c20)
d10 = (a10 * c00) + (a11 * c10) + (a12 * c20)
d01 = (a00 * c01) + (a01 * c11) + (a02 * c21)
…

$$\begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \\ a20 & a21 & a22 \end{bmatrix} \bullet \begin{bmatrix} c00 & c01 & c02 \\ c10 & c11 & c12 \\ c20 & c21 & c22 \end{bmatrix} = \begin{bmatrix} d00 & d01 & d02 \\ d10 & d11 & d12 \\ d20 & d21 & d22 \end{bmatrix}$$

Figuur 5.7: Vermenigvuldigen van matrices

Aangezien in de implementatie de tweede matrix een constante is, kan deze vermenigvuldiging uitgevoerd worden door voor elke kolom een bouwblok te gebruiken waarin de waardes van de constanten vastliggen. De 8 bouwblokken die hiervoor geïmplementeerd werden zijn Mult0, Mult1, Mult2, Mult3, Mult4, Mult5, Mult6 en Mult7. Mult0 is de bewerking met de eerste kolom, Mult7 is de bewerking met de laatste kolom. Binnen deze bouwblokken wordt gebruik gemaakt van Verm. Dit bouwblok zorgt ervoor dat de vermenigvuldiging in een eindig veld gebeurt. In Mult0 tot en met Mult7 gebeurt de optelling van de elementen. Omdat de overdrachtsbit van deze optelling niet van belang is kunnen we deze optelling uitvoeren aan de hand van exclusieve ofpoorten.
Elk element van de matrix is een byte, die we in x kunnen voorstellen. Bij voorbeeld, de byte $a_7a_6a_5a_4a_3a_2a_1a_0$, met a_7 de meestbeduidende bit en a_0 de minstbeduidende bit, kan men voorstellen door $a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$.

$$C = \begin{bmatrix} 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x \\ 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x \\ 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x \\ 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x \\ 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x \\ 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x \\ 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x \\ 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x \end{bmatrix}$$

Figuur 5.8: Constante matrix

Wanneer we 09_x binair schrijven, krijgen we 00001001, wat in x overeenkomt met $x^3 + 1$. Deze regel kunnen we toepassen op alle elementen uit de constante matrix, die in Figuur 5.8 weergegeven wordt. Dit resulteert in Tabel 5.1.

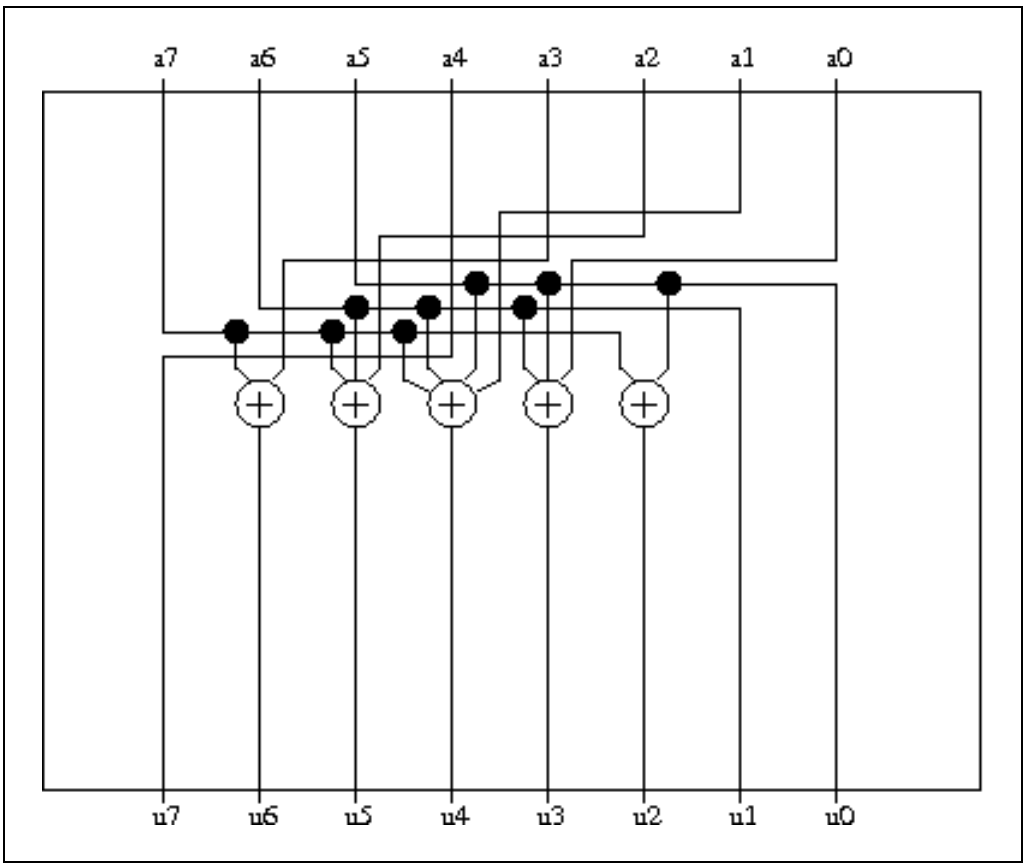
Element uit C	Overeenkomstige waarde in x
01_x	1
02_x	x
04_x	x^2
05_x	$x^2 + 1$
08_x	x^3
09_x	$x^3 + 1$

Tabel 5.1: Vermenigvuldigingswaardes

Wanneer men een element a vermenigvuldigt met 1, zal het product gelijk zijn aan het element a. De elementen 05_x en 09_x kunnen opgesplitst worden in respectievelijk $04_x + 01_x$ en $08_x + 01_x$. Afhankelijk van het ingangssignaal *factor*, zal in Verm de vermenigvuldiging met x, x^2 of x^3 uitgevoerd worden. De uitgangen van Verm worden, binnen Mult0 tot en met Mult7, aangeboden aan de ingangen van een exclusieve ofpoort, samen met de elementen die met 1 vermenigvuldigd moeten worden. Zo bekomt men het resultaat van de vermenigvuldiging binnen het eindige veld.

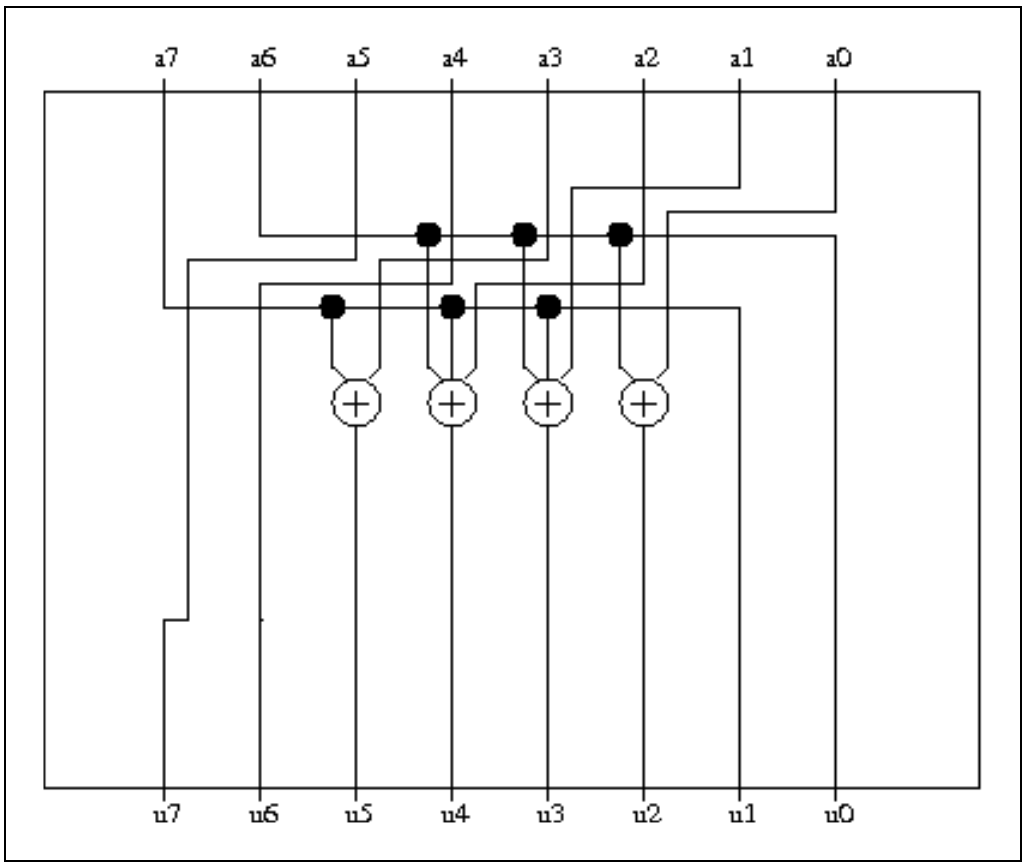
Werking Verm

Bij een vermenigvuldiging met x^3 gaan we als volgt te werk.
$a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ stelt de factor voor die vermenigvuldigd moet worden met x^3.
Dit geeft:
$(a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0) * x^3$
$= a_7x^{10} + a_6x^9 + a_5x^8 + a_4x^7 + a_3x^6 + a_2x^5 + a_1x^4 + a_0x^3$
De elementen $a_7x^{10} + a_6x^9 + a_5x^8$ behoren niet tot het eindig veld waarin we werken, en moeten dus omgerekend worden naar een equivalent binnen het eindig veld. Dit doen we met behulp van de irreduceerbare veelterm $x^8 + x^4 + x^3 + x^2 + 1$. Dit komt overeen met $x^8 = x^4 + x^3 + x^2 + 1$. Zo kunnen we a_7x^{10} vervangen door $a_7x^6 + a_7x^5 + a_7x^4 + a_7x^2$. Dit kunnen we ook doen voor a_6x^9 en a_5x^8, wat respectievelijk $a_6x^5 + a_6x^4 + a_6x^3 + a_6x$ en $a_5x^4 + a_5x^3 + a_5x^2 + a_5$ geeft. Wanneer we al deze termen samenvoegen krijgen we:
$a_7x^{10} + a_6x^9 + a_5x^8 + a_4x^7 + a_3x^6 + a_2x^5 + a_1x^4 + a_0x^3$
$= a_7x^6 + a_7x^5 + a_7x^4 + a_7x^2 + a_6x^5 + a_6x^4 + a_6x^3 + a_6x + a_5x^4 + a_5x^3 + a_5x^2 + a_5 + a_4x^7 + a_3x^6 + a_2x^5 + a_1x^4 + a_0x^3$
$= a_4x^7 + (a_3 + a_7)*x^6 + (a_2 + a_7 + a_6)*x^5 + (a_1 + a_7 + a_6 + a_5)*x^4 + (a_0 + a_6 + a_5)*x^3 + (a_7 + a_5)*x^2 + a_6x + a_5.$
In hardware worden deze optellingen gerealiseerd met behulp van exclusieve ofpoorten, omdat we de overdrachtsbits niet nodig hebben. Dit wordt weergegeven in Figuur 5.9.

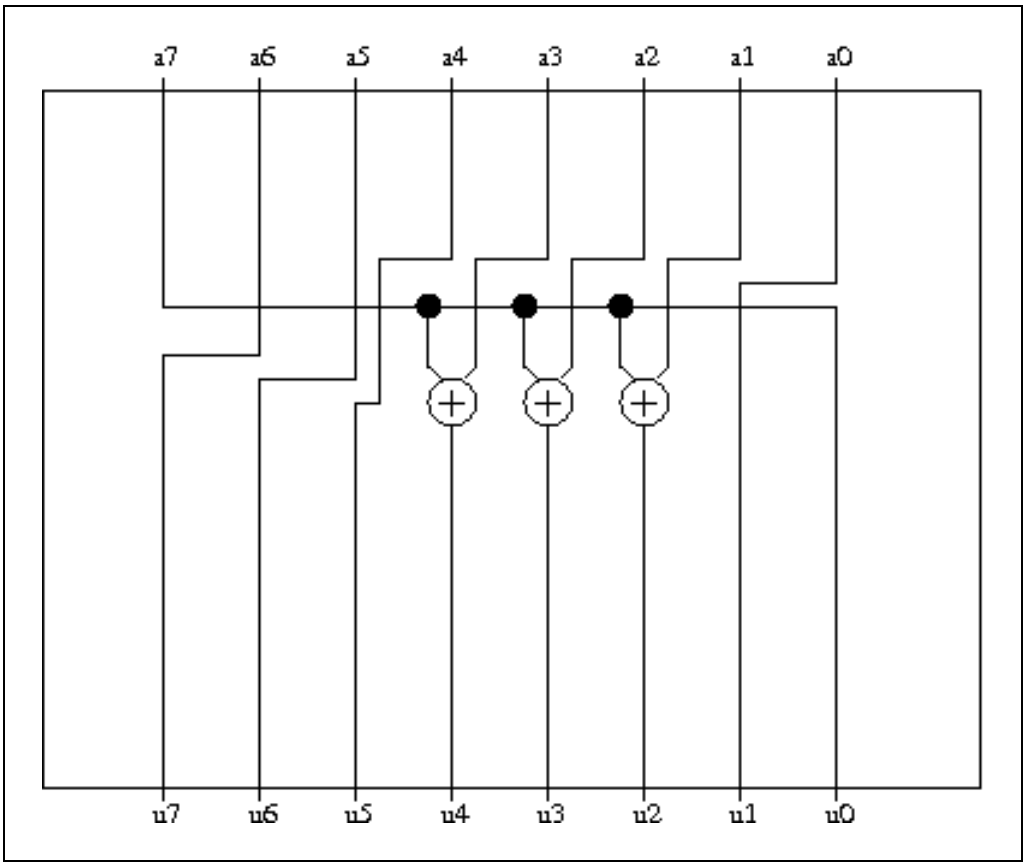


Figuur 5.9: Vermenigvuldigen met x³

Voor de vermenigvuldiging met x² en x gaan we op dezelfde manier te werk. Dit wordt weergegeven in respectievelijk Figuur 5.10 en Figuur 5.11.



Figuur 5.10: Vermenigvuldigen met x^2



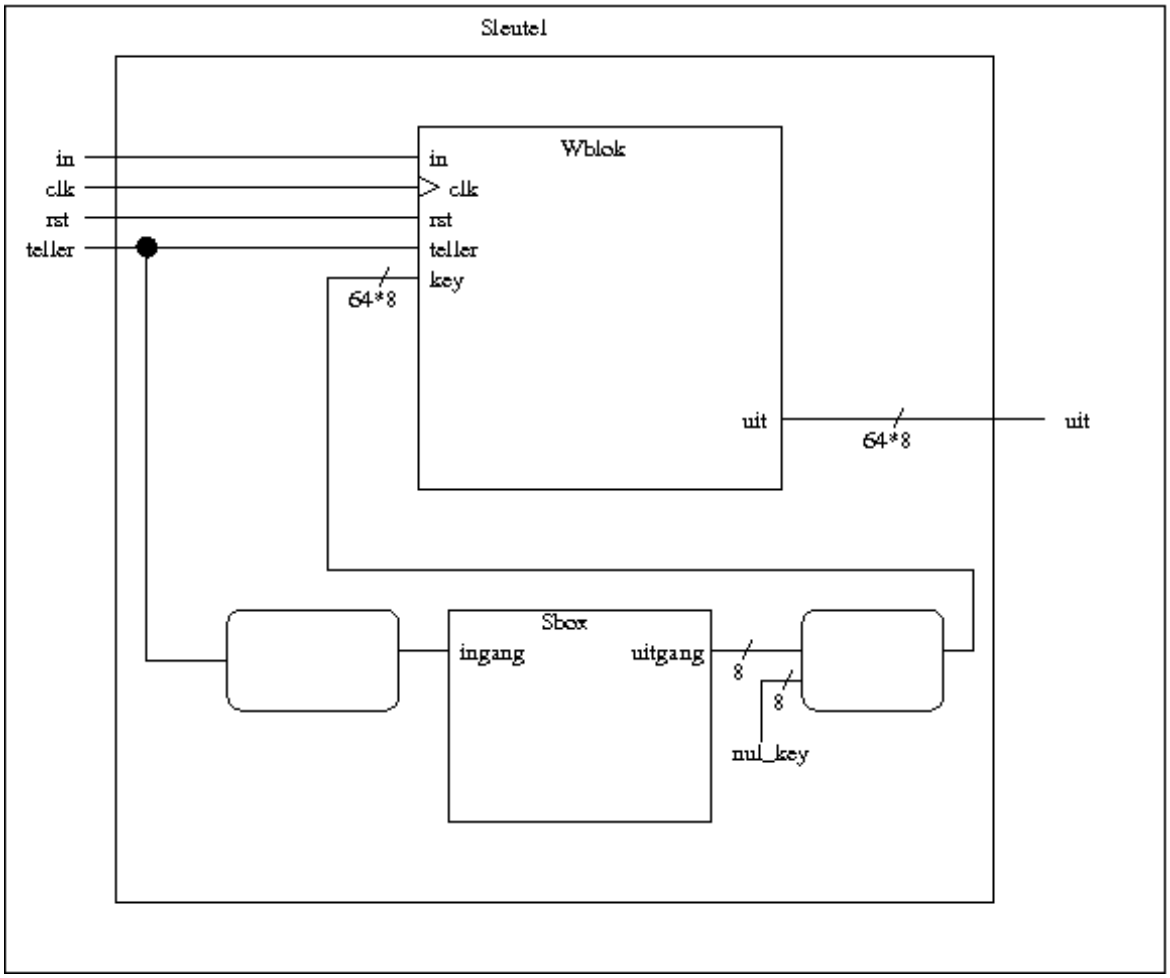
Figuur 5.11: Vermenigvuldigen met x

5.4.3.4 Sigma-bouwblok

Dit bouwblok heeft 2 ingangen, namelijk de uitgang van Teta en de uitgang van Sleutel. De overeenkomstige elementen uit beide matrices worden bit per bit aangeboden aan de ingang van een exclusieve ofpoort.
Deze vier bouwblokken, Gamma, Pi, Teta en Sigma, vormen samen het W-blok.

5.4.4 Sleutel-bouwblok

Dit bouwblok genereert de sleutel die door Sigma gebruikt wordt en is opgebouwd uit Wblok en Sbox, zoals in Figuur 5.12 weergegeven wordt. De ingang van het Wblok is bij het verwerken van de eerste 512 bits gelijk aan een matrix met allemaal 0'en. Bij het verwerken van de volgende 512 bits zal deze ingang gelijk zijn aan de tussentijdse hashwaarde van de vorige 512 bits.
Sbox genereert de sleutelingang van het Wblok binnen Sleutel. Deze sleutelingang bestaat uit een matrix, waarvan de eerste rij data bevat, die afhankelijk is van de ronde. De andere rijen bevatten allemaal 0'en.
Afhankelijk van de ronde, krijgt Sbox een ingang aangeboden. De uitgang van Sbox bepaalt de data van de eerste rij van de matrix van de sleutelingang.



Figuur 5.12: Architectuur van het Sleutelbouwblok

5.4.5 Output-bouwblok

Dit bouwblok is gebaseerd op Output van de implementatie van het MD5-algoritme. In tegenstelling tot het MD5-algoritme wordt er bij het Whirlpoolalgoritme een hashwaarde berekend van 512 bits in plaats van 128 bits bij het MD5-algoritme.
Dit bouwblok bestaat uit een register van 512 bits waarin de hashwaarde opgeslagen wordt. Wanneer *shift_data_out* hoog wordt zullen er gedurende 16 klokpulsen 32 bits van de hashwaarde op de uitgangspinnen gezet worden, te beginnen bij de 32 meest beduidende bits. Na 16 klokpulsen is de hashwaarde van 512 bits volledig uitgestuurd.

5.5 Whirlpool met RAM

Als alternatieve implementatie van het Whirlpool-algoritme werd er een deel van de implementatie vervangen door RAM. Deze RAM was reeds aanwezig op de FPGA. Door deze implementatie in RAM wordt er minder oppervlakte van de herconfigureerbare bouwblokken op de FPGA gebruikt.
Aangezien er voor het S-blok ook een tabel ter beschikking is die voor alle ingangen de uitgang weergeeft, kan S-blok ook eenvoudig in RAM geïmplementeerd worden. Deze tabel wordt in Figuur 5.13 weergegeven.

	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0c_x$	$0d_x$	$0E_x$	$0F_x$
00_x	18_x	23_x	$c6_x$	$E8_x$	87_x	$B8_x$	01_x	$4F_x$	36_x	$A6_x$	$d2_x$	$F5_x$	79_x	$6F_x$	91_x	52_x
10_x	60_x	Bc_x	$9B_x$	$8E_x$	$A3_x$	$0c_x$	$7B_x$	35_x	$1d_x$	$E0_x$	$d7_x$	$c2_x$	$2E_x$	$4B_x$	FE_x	57_x
20_x	15_x	77_x	37_x	$E5_x$	$9F_x$	$F0_x$	$4A_x$	dA_x	58_x	$c9_x$	29_x	$0A_x$	$B1_x$	$A0_x$	$6B_x$	85_x
30_x	Bd_x	$5d_x$	10_x	$F4_x$	cB_x	$3E_x$	05_x	67_x	$E4_x$	27_x	41_x	$8B_x$	$A7_x$	$7d_x$	95_x	$d8_x$
40_x	FB_x	EE_x	$7c_x$	66_x	dd_x	17_x	47_x	$9E_x$	cA_x	$2d_x$	BF_x	07_x	Ad_x	$5A_x$	83_x	33_x
50_x	63_x	02_x	AA_x	71_x	$c8_x$	19_x	49_x	$d9_x$	$F2_x$	$E3_x$	$5B_x$	88_x	$9A_x$	26_x	32_x	$B0_x$
60_x	$E9_x$	$0F_x$	$d5_x$	80_x	BE_x	cd_x	34_x	48_x	FF_x	$7A_x$	90_x	$5F_x$	20_x	68_x	$1A_x$	AE_x
70_x	$B4_x$	54_x	93_x	22_x	64_x	$F1_x$	73_x	12_x	40_x	08_x	$c3_x$	Ec_x	dB_x	$A1_x$	$8d_x$	$3d_x$
80_x	97_x	00_x	cF_x	$2B_x$	76_x	82_x	$d6_x$	$1B_x$	$B5_x$	AF_x	$6A_x$	50_x	45_x	$F3_x$	30_x	EF_x
90_x	$3F_x$	55_x	$A2_x$	EA_x	65_x	BA_x	$2F_x$	$c0_x$	dE_x	$1c_x$	Fd_x	$4d_x$	92_x	75_x	06_x	$8A_x$
$A0_x$	$B2_x$	$E6_x$	$0E_x$	$1F_x$	62_x	$d4_x$	$A8_x$	96_x	$F9_x$	$c5_x$	25_x	59_x	84_x	72_x	39_x	$4c_x$
$B0_x$	$5E_x$	78_x	38_x	$8c_x$	$d1_x$	$A5_x$	$E2_x$	61_x	$B3_x$	21_x	$9c_x$	$1E_x$	43_x	$c7_x$	Fc_x	04_x
$c0_x$	51_x	99_x	$6d_x$	$0d_x$	FA_x	dF_x	$7E_x$	24_x	$3B_x$	AB_x	cE_x	11_x	$8F_x$	$4E_x$	$B7_x$	EB_x
$d0_x$	$3c_x$	81_x	94_x	$F7_x$	$B9_x$	13_x	$2c_x$	$d3_x$	$E7_x$	$6E_x$	$c4_x$	03_x	56_x	44_x	$7F_x$	$A9_x$
$E0_x$	$2A_x$	BB_x	$c1_x$	53_x	dc_x	$0B_x$	$9d_x$	$6c_x$	31_x	74_x	$F6_x$	46_x	Ac_x	89_x	14_x	$E1_x$
$F0_x$	16_x	$3A_x$	69_x	09_x	70_x	$B6_x$	$d0_x$	Ed_x	cc_x	42_x	98_x	$A4_x$	28_x	$5c_x$	$F8_x$	86_x

Figuur 5.13: Waardes van S-blok

Overal waar Sbox vroeger gebruikt werd, wordt nu gebruik gemaakt van RAM, namelijk in Wblok en in Sleutel. We hebben hiervoor het bouwblok RAMB16_S9 geïnstantieerd. Deze RAM heeft een grootte van 2k * (8 + 1) = 18k (1k = 1 kilobit = 1024 bits). 2k wordt echter gebruikt voor de pariteitsbit, zodat er nog 16k over is om data in op te slaan. Dit is meer dan voldoende voor onze implementatie, aangezien we maar 2k geheugen nodig hebben.
Aangezien deze tabel constante waardes bevat, zullen we enkel de RAM uitlezen. De tabel wordt bij het programmeren van de FPGA in de RAM geladen.

6 Vergelijking van de implementaties

In dit hoofdstuk zullen de verschillende implementaties met elkaar vergeleken worden wat klokfrequentie, oppervlakte en verwerkingssnelheid betreft. Eerst worden de verschillende versies van het MD5-algoritme tegen elkaar afgewogen. Daarna worden de verschillende versies van het Whirlpoolalgoritme met elkaar vergeleken. Tenslotte zullen de verschillende implementaties van beide algoritmes tegen elkaar afgewogen worden.

6.1 De FPGA

Voor de implementaties van alle algoritmes werd er gekozen voor een FPGA van Xilinx, namelijk de Virtex-II Pro XC2VP100 met snelheidsgraad 6 en als verpakking FF1696. Deze FPGA bevat 44096 slices, 444 blokken RAM van 18 kilobit en 1696 pinnen.
Het implementeren van de algoritmes gebeurde met het softwarepakket Xilinx ISE webpack 7.1i. Al deze implementaties werden op dezelfde computer uitgevoerd.

6.2 Aantal poorten op ASIC

Met het softwarepakket Design Vision van Synopsis werd er berekend hoe groot enkele van de implementaties op ASIC zouden zijn indien men een 0,25µm CMOS-technologie gebruikt. Deze grootte kan ook omgezet worden naar het aantal equivalente poorten dat wordt gebruikt op de ASIC, door de oppervlakte te delen door de grootte van 1 geïnverteerde enpoort (namelijk 23,8 µm²). Als instelfrequentie werd dezelfde frequentie gebruikt die de implementatie op de FPGA kon halen. De implementaties op ASIC kunnen sneller werken, maar de analyse werd beperkt tot het vergelijken van de oppervlakte zonder daarbij de maximale frequentie na te streven.

6.3 MD5

6.3.1 Klokfrequentie

Zoals reeds vermeld, werden er verschillende implementaties van het MD5-algoritme gemaakt. Omdat de eerste implementatie enkel aan een zeer lage frequentie (namelijk lager dan 2 MHz) kan werken werd er een tweede versie geïmplementeerd. Bij deze versie werd om de 16 rondes een register geplaatst. Deze implementatie is sneller, met een maximale klokfrequentie van iets meer dan 7 MHz. Een derde versie bevat om de 4 rondes een register. Deze versie kan werken met een maximale klokfrequentie van ongeveer 26 MHz. Hierna werd een vierde versie geïmplementeerd waarbij er na elke ronde een register geplaatst werd. Deze kan werken met een klokfrequentie die net kleiner is dan 66 MHz. Afhankelijk van de applicatie kan achteraf gekozen worden voor een van deze vier implementaties. De juiste waardes worden weergegeven in Tabel 6.1.

	MD5v1	MD5v2	MD5v3	MD5v4
Minimale periode (ns)	560	141,5	38	15,2
Maximale frequentie (MHz)	1,786	7,067	26,316	65,789
Oppervlakte (aantal slices)	5987	6137	5858	9953
Aantal slices gebruikt (%)	13,58	13,92	13,28	22,57
RAM (# blokken van 16 bit)	-	-	-	-
RAM gebruikt (%)	-	-	-	-
# klokpulsen nodig om 512 bits te berekenen	22	25	39	88
tijd nodig om 512 bits te berekenen (ns)	12320	3537,5	1482	1337,6
# klokpulsen nodig om 1024 bits te berekenen	40	46	74	172
tijd nodig om 1024 bits te berekenen (ns)	22400	6509	2812	2614,4
AT 512 (# slices * ns)	73759840,0	21709637,5	8681556,0	13313132,8
AT 1024 (# slices * ns)	134108800,0	39945733,0	16472696,0	26021123,2
Oppervlakte in een ASIC (µm²)	-	3121945,2	3414375,4	5065600,3
Aantal poorten in een ASIC	-	131174,17	143461,15	212840,35

Tabel 6.1: Vergelijking van MD5

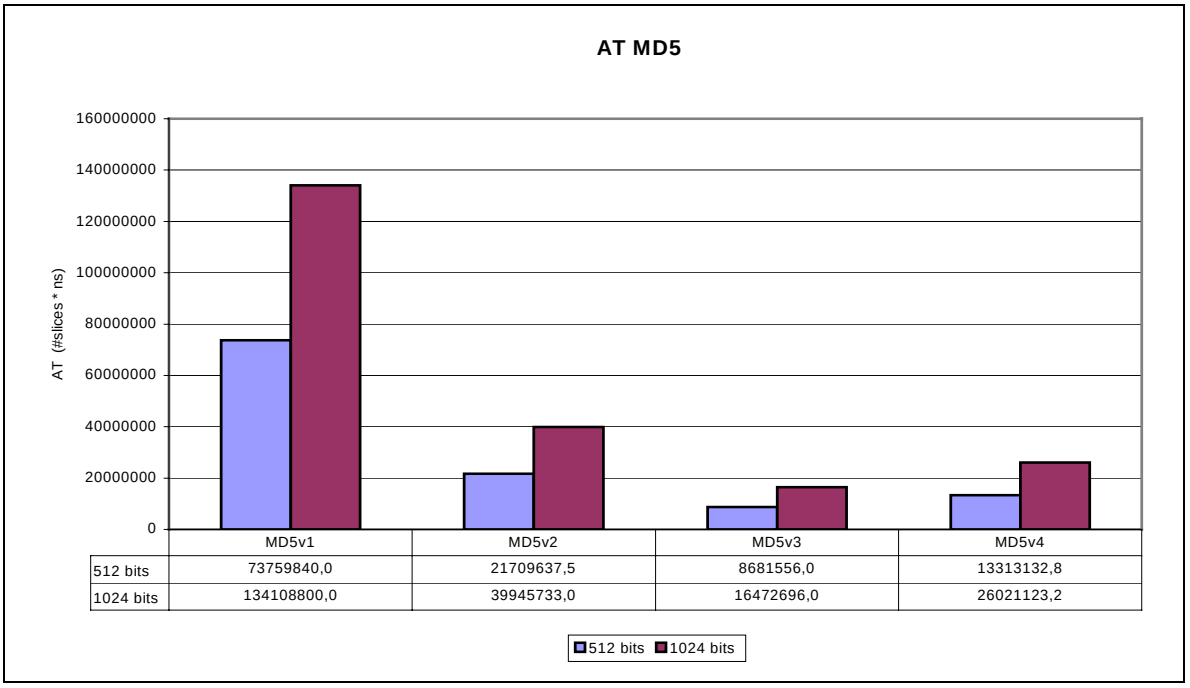
6.3.2 Oppervlakte

Wanneer de implementatie aangepast wordt door het toevoegen van registers, zodat deze met een hogere klokfrequentie kan werken, zal de gebruikte oppervlakte stijgen. Dit kan men zien in Tabel 6.1.
Wanneer men het aantal slices dat gebruikt wordt op de FPGA vergelijkt, ziet men echter dat MD5v3 een uitzondering is, want men zou verwachten dat deze meer slices zou gebruiken dan MD5v2 en minder dan MD5v4. Dit is echter niet het geval, want deze implementatie gebruikt de minste slices van alle implementaties. Dit is waarschijnlijk te verklaren doordat het plaatsen en routen van de hardware beter geoptimaliseerd kon worden.
Indien men het aantal poorten van de implementaties op een ASIC bekijkt, ziet men dat het aantal poorten stijgt naarmate er meer registers werden toegevoegd. Hier ziet men dat MD5v3 wel groter is dan MD5v2 en kleiner dan MD5v4 zoals verwacht werd. Voor MD5v1 werd deze berekening niet gedaan aangezien deze implementatie waarschijnlijk toch onbruikbaar is door de lage klokfrequentie.

6.3.3 Verwerkingssnelheid

Voor het inlezen van 512 bits zijn 16 klokperiodes nodig. Voor het uitsturen van de hashwaarde van 128 bits zijn 4 klokperiodes nodig. Dit geldt voor de vier versies van MD5. Afhankelijk van het aantal toegevoegde registers zijn er nog een aantal extra klokperiodes nodig voor het berekenen van de hashwaarde. Deze waardes kunnen teruggevonden worden in Tabel 6.1. Bij MD5v1 en MD5v2 is de tijd die nodig is om de hashwaarde van een

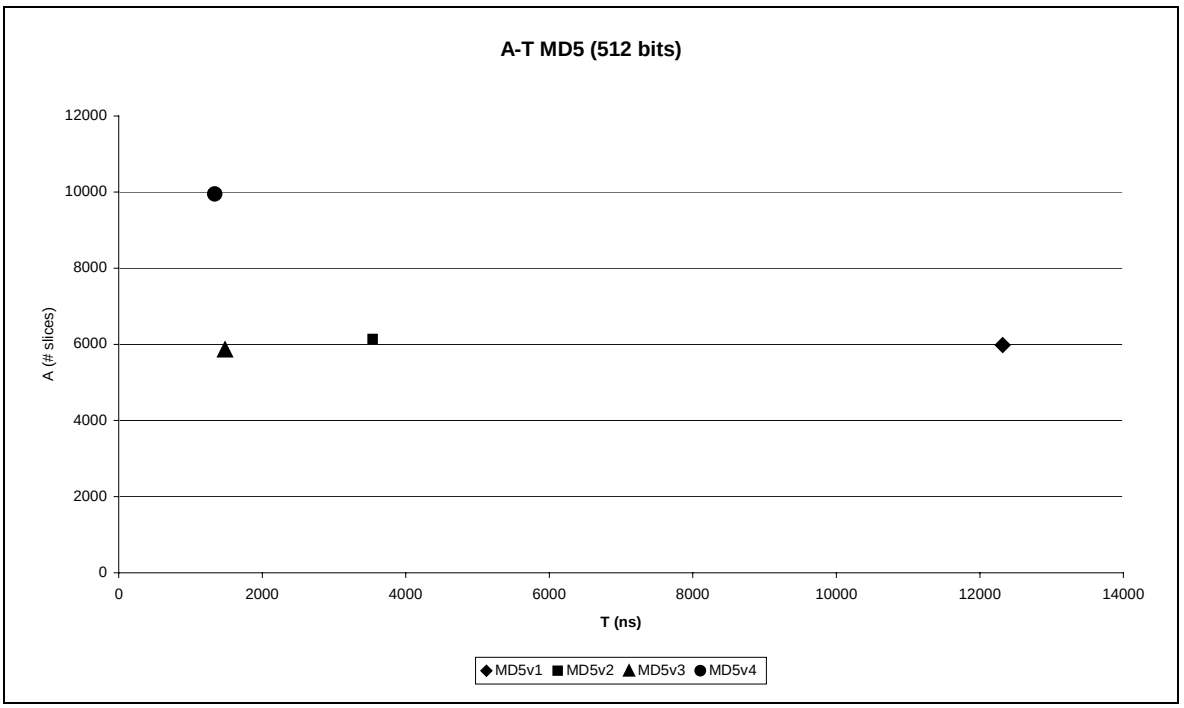
boodschap van 512 bits te berekenen veel groter dan bij MD5v3 en MD5v4. Dit komt omdat de maximale klokfrequentie bij deze implementaties laag is. Deze klokfrequentie zal echter ook gebruikt worden voor het inlezen en uitsturen van de data, zodat de totale tijd nodig voor het inlezen, verwerken en uitsturen van de data veel groter zal zijn bij de eerste twee implementaties dan bij de laatste twee.
Om een vergelijking te kunnen maken waarin zowel oppervlakte als snelheid een invloed hebben, wordt dikwijls gebruik gemaakt van een meetsleutel die rekening houdt met beide waarden. Deze meetsleutel wordt het AT-product genoemd en is gelijk aan het product van de oppervlakte en de vertraging. De oppervlakte is het aantal slices dat op FPGA gebruikt wordt en de vertraging is de tijd die nodig is om een hashwaarde van een boodschap te berekenen. Uit deze waardes kan men duidelijk afleiden dat MD5v3 de beste prestaties levert, aangezien de routing van MD5v3 beter was dan men zou verwachten. Ook MD5v4 levert goede prestaties, zoals men kan zien in Grafiek 6.1.



	MD5v1	MD5v2	MD5v3	MD5v4
512 bits	73759840,0	21709637,5	8681556,0	13313132,8
1024 bits	134108800,0	39945733,0	16472696,0	26021123,2

Grafiek 6.1: AT van MD5

In Grafiek 6.2 wordt de oppervlakte op FPGA uitgezet tegen de benodigde tijd om een hashwaarde te berekenen bij een boodschap van 512 bits. Hoe dichter de waarde bij de oorsprong ligt hoe beter de implementatie is op het vlak van snelheid en oppervlakte.
Uit deze grafiek blijkt dat MD5v3 en MD5v4 beiden goede implementaties zijn en afhankelijk van de gestelde eisen kan men een keuze maken tussen beide implementaties.



Grafiek 6.2: A-T van MD5 voor 512 bits

6.4 Whirlpool

Zoals eerder vermeld zijn er twee verschillende implementaties van het Whirlpoolalgoritme. In de eerste implementatie werd het S-blok in logica uitgevoerd, in de tweede implementatie in RAM.

6.4.1 Klokfrequentie

In Tabel 6.2 ziet men dat de maximale frequentie van beide implementaties niet veel van elkaar verschilt. De implementatie waarbij RAM wordt gebruikt kan met een iets hogere klokfrequentie werken.

6.4.2 Oppervlakte

In Tabel 6.2 kan men duidelijk zien dat het aantal slices dat gebruikt wordt voor de versie zonder RAM veel groter is dan het aantal slices dat nodig is voor de implementatie met RAM. Aangezien de RAM-blokken standaard op deze FPGA aanwezig zijn, zorgt het gebruik ervan voor een efficiëntere benutting van de mogelijkheden die de FPGA biedt. Omdat het niet mogelijk is RAM te implementeren op ASIC met de bibliotheek die voorhanden was, werd enkel voor Whirlpool zonder RAM een implementatie op ASIC geëvalueerd.

	Whirlpool zonder RAM	Whirlpool met RAM
Minimale periode (ns)	9	8,7
Maximale frequentie (MHz)	111,111	114,943
Oppervlakte (aantal slices)	8289	5702
Aantal slices gebruikt (%)	18,8	12,93
RAM (# blokken van 16 bit)	-	136
RAM gebruikt (%)	-	30,63
# klokpulsen nodig om 512 bits te berekenen	43	44
tijd nodig om 512 bits te berekenen (ns)	387	382,8
# klokpulsen nodig om 1024 bits te berekenen	70	72
tijd nodig om 1024 bits te berekenen (ns)	630	626,4
AT 512 (# slices * ns)	3207843,0	2182725,6
AT 1024 (# slices * ns)	5222070,0	3571732,8
Oppervlakte in een ASIC (µm²)	3355038,7	-
Aantal poorten in een ASIC	140968,01	-

Tabel 6.2: Vergelijking van Whirlpool

6.4.3 Verwerkingssnelheid

Zowel voor het inlezen van 512 bits, als het uitsturen van de hashwaarde van 512 bits zijn er 16 klokperiodes nodig. De tijd die nodig is voor het berekenen van een hashwaarde is ongeveer gelijk voor beide implementaties.

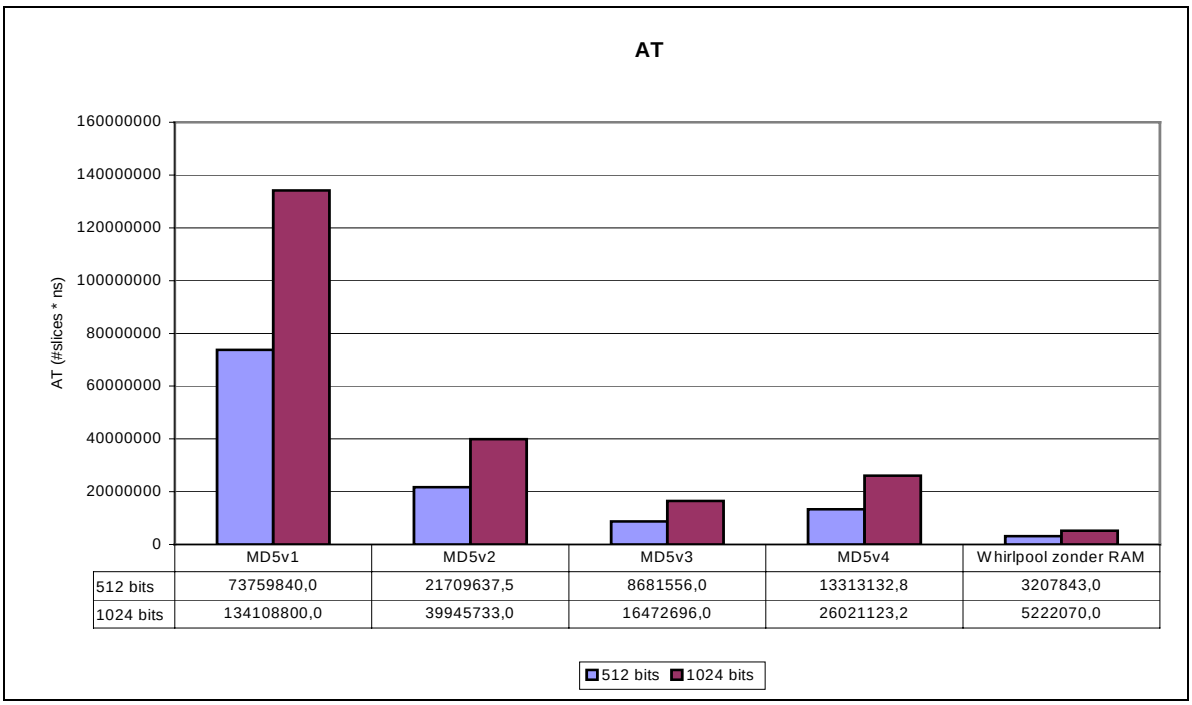
6.5 MD5 tegenover Whirlpool

Zoals Tabel 6.3 duidelijk weergeeft, kunnen de implementaties van Whirlpool met een hogere klokfrequentie werken dan de implementaties van het MD5-algoritme. De implementatie van het MD5-algoritme die met de hoogste klokfrequentie kan werken, namelijk MD5v4, gebruikt meer slices van de FPGA dan beide implementaties van het Whirlpoolalgoritme, en werkt toch op een lagere klokfrequentie. Ondanks het feit dat Whirlpool 16 klokpulsen nodig heeft om de hashwaarde uit te sturen, in tegenstelling tot MD5 dat slechts 4 klokpulsen nodig heeft, is de tijd die nodig is om de hardware te doorlopen voor de implementatie van het Whirlpoolalgoritme beduidend kleiner dan voor de implementaties van het MD5-algoritme.

	MD5v1	MD5v2	MD5v3	MD5v4	Whirlpool zonder RAM	Whirlpool met RAM
Minimale periode (ns)	560	141,5	38	15,2	9	8,7
Maximale frequentie (MHz)	1,786	7,067	26,316	65,789	111,111	114,943
Oppervlakte (aantal slices)	5987	6137	5858	9953	8289	5702
Aantal slices gebruikt (%)	13,58	13,92	13,28	22,57	18,8	12,93
RAM (# blokken van 16 bit)	-	-	-	-	-	136
RAM gebruikt (%)	-	-	-	-	-	30,63
# klokpulsen nodig om 512 bits te berekenen	22	25	39	88	43	44
tijd nodig om 512 bits te berekenen (ns)	12320	3537,5	1482	1337,6	387	382,8
# klokpulsen nodig om 1024 bits te berekenen	40	46	74	172	70	72
tijd nodig om 1024 bits te berekenen (ns)	22400	6509	2812	2614,4	630	626,4
AT 512 (# slices * ns)	73759840,0	21709637,5	8681556,0	13313132,8	3207843,0	2182725,6
AT 1024 (# slices * ns)	134108800,0	39945733,0	16472696,0	26021123,2	5222070,0	3571732,8
Oppervlakte in een ASIC (µm²)	-	3121945,2	3414375,4	5065600,3	3355038,7	-
Aantal poorten in een ASIC	-	131174,17	143461,15	212840,35	140968,01	-

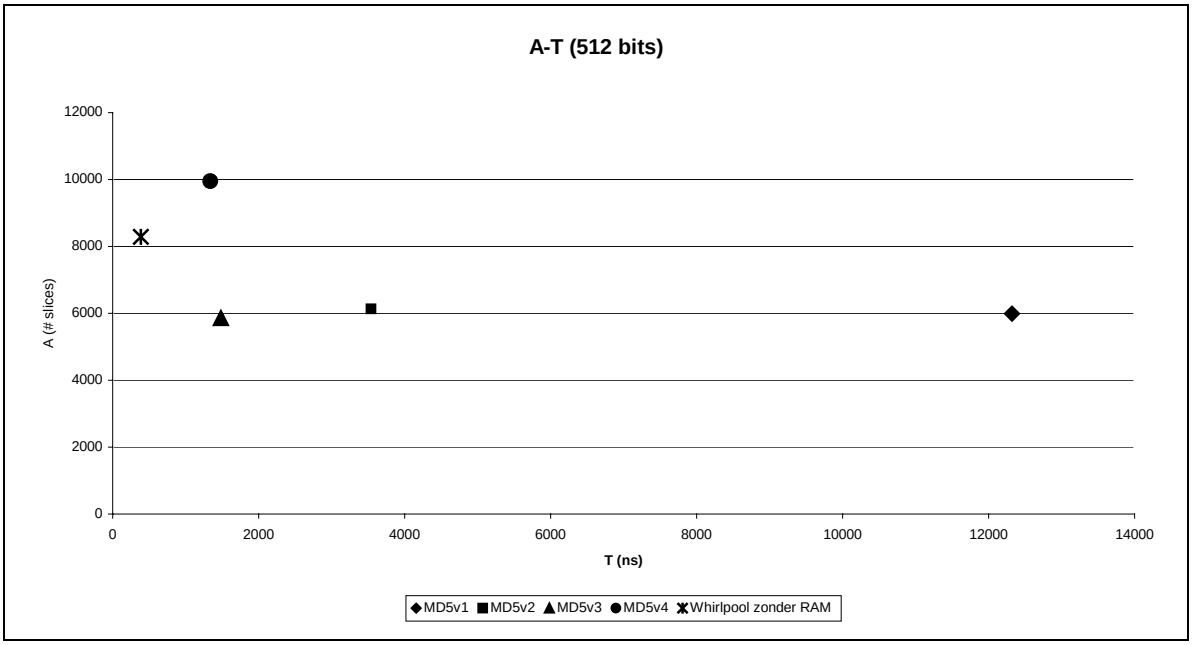
Tabel 6.3: Vergelijking van MD5 en Whirlpool

De AT van elke implementatie werd uitgezet in Grafiek 6.3. Hieruit blijkt duidelijk dat de implementatie van Whirlpool beter is dan de implementaties van MD5. Enkel Whirlpool met RAM werd niet opgenomen in de grafiek omdat deze een vertekend beeld zou geven aangezien enkel de oppervlakte en niet de gebruikte RAM in rekening gebracht zou worden. Ook in de volgende grafieken zal Whirlpool met RAM om deze reden niet opgenomen worden. Aangezien Whirlpool met RAM ongeveer dezelfde snelheid haalt als Whirlpool zonder RAM is deze implementatie ook goed bruikbaar.



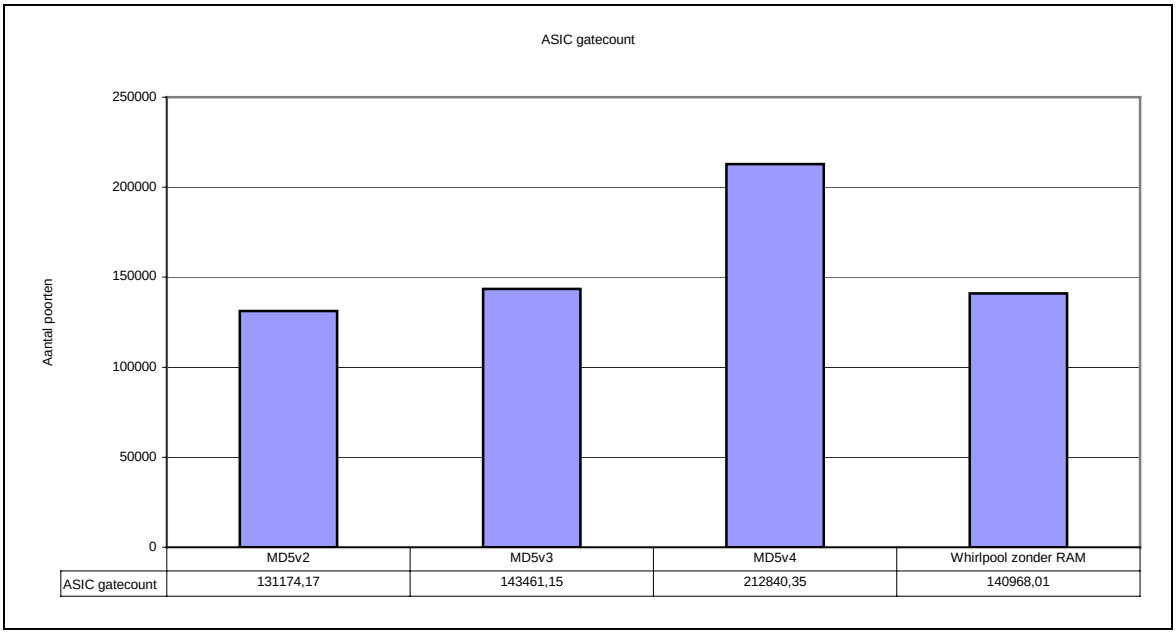
Grafiek 6.3: AT van MD5 en Whirlpool

In Grafiek 6.4 wordt de gebruikte oppervlakte op FPGA uitgezet tegenover de tijd nodig om een hashwaarde te berekenen van een boodschap van 512 bits. Hieruit kan afgeleid worden dat Whirlpool zonder RAM sneller is dan MD5v3 en MD5v4. Deze implementatie gebruikt meer oppervlakte van de FPGA dan MD5v3 maar minder dan MD5v4.



Grafiek 6.4: A-T van MD5 en Whirlpool voor 512 bits

In Grafiek 6.5 wordt de ASIC gatecount van de verschillende implementaties weergegeven. Indien men de algoritmes in een ASIC zou implementeren zou Whirlpool de minste poorten nodig hebben en dus het kleinst zijn. Aangezien deze implementaties gebeurden met een instelfrequentie gelijk aan de frequentie die gehaald werd op FPGA zal Whirlpool ook sneller zijn dan de snelste implementatie van het MD5-algoritme.



Grafiek 6.5: ASIC gatecount

We kunnen dus vaststellen dat de implementatie van Whirlpool op een XC2VP100 FPGA kleiner en sneller is dan de snelste implementatie van MD5, namelijk MD5v4, op dezelfde FPGA. Hiermee tonen we aan dat Whirlpool mogelijk een goed alternatief biedt voor de veelgebruikte MD5 standaard. Belangrijk hierbij is dat er op het MD5-algoritme reeds succesvolle aanvallen zijn uitgevoerd, terwijl nog niet aangetoond werd dat het Whirlpoolalgoritme onveilig is.[15]

[15] X. Wang en H. Yu, *How to Break MD5 and Other Hash Functions*, in Advances in Cryptology: Proceedings of EUROCRYPT'05, Lecture Notes in Computer Science, R. Cramer, Ed., no. 3494, Springer-Verlag, pp. 104-113.

7 Besluit

In deze masterproef werden zowel het MD5- als het Whirlpoolalgoritme bestudeerd en geïmplementeerd. Uit de verkregen resultaten kan men afleiden dat het Whirlpoolalgoritme een waardig alternatief biedt voor het veelgebruikte MD5-algoritme.
Er moet wel opgemerkt worden dat de implementatie van MD5 in het eerste deel van dit eindwerk werd uitgevoerd, terwijl Whirlpool als tweede werd geïmplementeerd. Men kan dus aannemen dat de VHDL-code voor de implementaties van het Whirlpoolalgoritme op een meer efficiënte manier geschreven is dan de code voor de implementaties van het MD5-algoritme. Indien de implementatie van het MD5-algoritme nu opnieuw uitgevoerd zou worden met de huidige kennis, zou deze implementatie waarschijnlijk kleiner en sneller zijn dan de versie waarover hier gerapporteerd wordt.
Na het implementeren en vergelijken van beide algoritmes, zou een logisch vervolg de vergelijking met andere implementaties zijn. Zowel andere implementaties van het MD5- en Whirlpoolalgoritme, als implementaties van andere hashfuncties zouden hiervoor bruikbare vergelijkingswaardes kunnen opleveren.
Ook het implementeren en uitvoeren van aanvallen op de implementaties zou waardevolle resultaten aan het licht kunnen brengen.

Bibliografie

ADVAMEG, INC., *RFC1321*, online, http://www.faqs.org/rfcs/rfc1321.html, 1 december 2005.

BARRETO, S. L. M. P., *The Whirlpool Hash Function*, online, http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html, 1 december 2005.

CYPHER RESEARCH LABORATORIES PTY LTD, *History of cryptography*, online, http://www.cypher.com.au/crypto_history.htm, 1 december 2005.

INTERNATIONAL ORGANIZATION OF STANDARDIZATION, *ISO/IEC 10118-3:2004*, online, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=39876&ICS1=35&ICS2=40&ICS3=, 1 december 2005.

KATHOLIEKE UNIVERSITEIT LEUVEN, *Computer Security and Industrial Cryptography*, online, http://www.esat.kuleuven.be/cosic/, 1 december 2005.

LANO, J, *Hash function workshop slides*, online, www.ecrypt.eu.org/stvl/hfw/McLoone.pdf, 1 december 2005.

MENEZES, A. J., *Handbook of applied cryptography*, online, http://www.cacr.math.uwaterloo.ca/hac/, 1 december 2005.

OPEN FORTRESS, *Toolbox der Cryptografie*, online, http://openfortress.nl/doc/essay/CryptoToolbox/index.nl.html, 1 december 2005.

Stallings, W., *Cryptography and Network Security*, *Principles and Practices*, derde druk, Prentice Hall, Upper Saddle River, 2003.

WIKIPEDIA, *Application-specific integrated circuit*, online, http://en.wikipedia.org/wiki/Asic, 1 december 2005.

WIKIPEDIA, *CPLD*, online, http://en.wikipedia.org/wiki/CPLD, 1 december 2005.

WIKIPEDIA, *Eindig lichaam*, online, http://nl.wikipedia.org/wiki/Eindig_lichaam, 1 maart 2006.

WIKIPEDIA, *Field-programmable gate array*, online, http://en.wikipedia.org/wiki/FPGA, 1 december 2005.

WIKIPEDIA, *Finite field*, online, http://en.wikipedia.org/wiki/Finite_Field, 1 maart 2006.

X. Wang en H. Yu, *How to Break MD5 and Other Hash Functions*, in Advances in Cryptology: Proceedings of EUROCRYPT'05, Lecture Notes in Computer Science, R. Cramer, Ed., no. 3494, Springer-Verlag, pp. 104-113.

Bijlages

In bijlage vindt u een cd, die de volgende mappen en bestanden bevat:

- in de map asicgatecount_rapporten vindt u de 4 rapporten van de implementaties op ASIC.
- in de map implementatie_rapporten vindt u de 6 rapporten van de implementaties op FPGA.
- in de map VHDL_code vindt u opnieuw verschillende mappen die al de VHDL-files, testbenches en een samenvatting van een bepaalde implementatie bevatten.
- ook vindt u het pdf-bestand van deze masterproef op de cd.