

EPO-3

Extreme Winterslaap Interrupter

Final report

14-01-2015

Projectgroep A1



Roy Blokker 4148894  
Martin Geertjes 4324285  
Rens Hamburger 4292936  
Kevin Hill 4287592  
Alex Oudsen 4325494  
Joran Out 4331958  
Elke Salzmann 4311450  
Jeroen van Uffelen 4232690

# SAMENVATTING

Dit is het verslag van "EPO-3" van groep A1. Hierin is te vinden hoe het project is aangepakt en uitgewerkt. Het systeem dat is ontworpen lijkt op de Wake-up Light van het bedrijf Philips. De obstakels die overwonnen moesten worden zijn het ontvangen en verwerken van het DCF-77 signaal, het aansturen van het licht, het aansturen van het geluid en het aansturen van een LCD schermpje. In het verslag is te vinden hoe al deze subsystemen zijn ontworpen en uitgewerkt.

# INHOUDSOPGAVE

<b>Samenvatting</b>	<b>ii</b>	
<b>1</b>	<b>Introductie</b>	<b>1</b>
<b>2</b>	<b>Ontwerp specificatie</b>	<b>2</b>
<b>3</b>	<b>Systeem overzicht en ontwerp</b>	<b>3</b>
<b>4</b>	<b>DCF77</b>	<b>5</b>
4.1	Inleiding . . . . .	5
4.2	Specificaties . . . . .	6
4.2.1	Ingangen . . . . .	6
4.2.2	Uitgangen . . . . .	6
4.2.3	Gedrag . . . . .	6
4.3	Functionaliteit . . . . .	6
4.3.1	FSM diagrammen . . . . .	8
4.3.2	VHDL code . . . . .	8
4.4	Simulatie . . . . .	9
4.5	Rapid prototyping (FPGA implementatie) . . . . .	9
4.6	Resultaten . . . . .	10
<b>5</b>	<b>Main controller</b>	<b>11</b>
5.1	Inleiding . . . . .	11
5.2	Specificaties . . . . .	11
5.2.1	Ingangen . . . . .	11
5.2.2	Uitgangen . . . . .	11
5.2.3	Gedrag . . . . .	12
5.3	Functionaliteit . . . . .	12
5.4	VHDL code . . . . .	13
5.5	Testen . . . . .	13
5.6	Simulatie . . . . .	13
5.7	Resultaten . . . . .	13
5.7.1	Conclusie en discussie . . . . .	15
<b>6</b>	<b>Alarm</b>	<b>16</b>
6.1	Inleiding . . . . .	16
6.2	Specificaties . . . . .	16
6.2.1	Ingangen . . . . .	16
6.2.2	Uitgangen . . . . .	16
6.2.3	Gedrag . . . . .	16
6.3	Functionaliteit . . . . .	17
6.3.1	FSM . . . . .	17
6.3.2	Code . . . . .	18
6.4	Resultaten . . . . .	19
6.5	Testen . . . . .	19
<b>7</b>	<b>LCD controller</b>	<b>20</b>
7.1	LCD Controller . . . . .	20
7.2	Specificaties . . . . .	21
7.2.1	Gedrag . . . . .	22

7.3	Functionaliteit . . . . .	22
7.4	Subsystemen LCD . . . . .	23
7.4.1	VHDL code . . . . .	24
7.5	Simulatie . . . . .	24
7.6	Testen . . . . .	24
7.7	Resultaten . . . . .	24
7.7.1	Conclusie en discussie . . . . .	24
<b>8</b>	<b>Results for total design</b>	<b>25</b>
<b>9</b>	<b>Plan voor het testen van de chip</b>	<b>26</b>
9.1	Logic analyzer . . . . .	26
9.2	Testsignalen . . . . .	26
<b>10</b>	<b>Voortgang van het project</b>	<b>27</b>
10.1	Inleiding . . . . .	27
10.2	Werkverdeling . . . . .	27
10.3	Samenwerking binnen de groep . . . . .	27
10.4	Afspraken en deadlines binnen de groep . . . . .	27
<b>11</b>	<b>Conclusie</b>	<b>28</b>
<b>A</b>	<b>FSM diagrammen (DCF77)</b>	<b>29</b>
<b>B</b>	<b>VHDL code</b>	<b>32</b>
B.1	VHDL beschrijving van het DCF77 blok . . . . .	32
B.2	Testbenches voor het DCF77 blok . . . . .	49
B.3	FPGA hulpbestanden van het DCF77 blok . . . . .	62
B.4	VHDL code controller . . . . .	68
B.4.1	Top level entity . . . . .	68
B.4.2	Behavioural VHDL code controller . . . . .	68
B.4.3	Menu entity . . . . .	68
B.4.4	Behavioural VHDL code menu . . . . .	69
B.4.5	Memory . . . . .	72
B.4.6	Behavioural VHDL memory . . . . .	73
B.4.7	Entity buffer . . . . .	73
B.4.8	Behavioural VHDL buffer . . . . .	73
B.5	Testbenches voor de controller . . . . .	74
B.5.1	VHDL controller . . . . .	74
B.5.2	Testbench VHDL menu . . . . .	75
B.5.3	Testbench VHDL geheugen . . . . .	77
B.5.4	Testbench VHDL buffer . . . . .	78
B.6	Vhdl code van het alarm . . . . .	78
B.6.1	Entity alarm-compare . . . . .	78
B.6.2	Behavioural alarm-compare . . . . .	78
B.6.3	Top entity alarm . . . . .	79
B.6.4	Behavioural alarm . . . . .	79
B.6.5	Entity alarm-counter . . . . .	80
B.6.6	Behavioural alarm-counter . . . . .	80
B.6.7	Entity alarm-pwm . . . . .	81
B.6.8	Behavioural alarm-pwm . . . . .	81
B.7	VHDL code LCD aansturing . . . . .	82
B.7.1	Entity Tijd . . . . .	82
B.7.2	Behavioural Tijd . . . . .	82
B.7.3	Entity Datum . . . . .	86
B.7.4	Behavioural Datum . . . . .	87
B.7.5	Entity DCF LCD . . . . .	91
B.7.6	Behavioural DCF LCD . . . . .	91
B.7.7	Entity Geluid . . . . .	92

B.7.8 Behavioural Geluid . . . . .	92
B.7.9 Entity Licht . . . . .	94
B.7.10 Behavioural Licht . . . . .	94
B.7.11 Entity Menu scherm . . . . .	95
B.7.12 Behavioural menu scherm . . . . .	95
B.7.13 Entity Wektijd . . . . .	97
B.7.14 Behavioural Wektijd . . . . .	97
B.7.15 Entity send control . . . . .	100
B.7.16 Behavioural send control . . . . .	100
B.7.17 Entity send bus . . . . .	102
B.7.18 Behavioural send bus . . . . .	102
B.7.19 Entity send top . . . . .	104
B.7.20 Structural send top . . . . .	104
<b>C Simulatie resultaten DCF</b>	<b>107</b>
<b>D Simulatie resultaten</b>	<b>112</b>
D.1 Behavioral simulatie . . . . .	112
D.2 Synthesize simulatie . . . . .	113
D.3 Extracted simulatie . . . . .	114
D.4 Timing . . . . .	115
<b>E Simulatie resultaten Alarm</b>	<b>116</b>
E.1 Behavioral simulatie . . . . .	116
E.2 Extracted simulatie . . . . .	117
<b>F Simulatie resultaten LCD</b>	<b>118</b>
F.1 Menu . . . . .	118
F.2 Geluid . . . . .	118
F.3 Licht . . . . .	118
F.4 DCF . . . . .	118
<b>Bibliografie</b>	<b>121</b>

# 1

## INTRODUCTIE

Epo 3 staat in het teken van het ontwerpen van een chip. Wat voor product er ontworpen gaan worden ligt aan de projectgroep. Het bedenken van het ontwerp is de eerste stap in het ontwerpproces, bij deze stap moet er al rekening gehouden met de randvoorwaarden die aan het project gesteld worden, zoals het aantal beschikbare transistoren op de chip.

Er is besloten om een wake-up light te maken. De belangrijkste functie is dat het licht 15 minuten voor de alarmtijd langzaam aan begint te gaan, totdat de lamp op de alarmtijd op volle sterkte brandt. Daarnaast zullen er nog een paar functies toegevoegd worden. Het DCF-signalen zal opgevangen worden voor de actuele datum en tijd, dit zal op een LCD-scherm worden laten zien. Door middel van vijf knoppen kan de wekker bediend worden. De alarmtijd kan ingesteld worden en de gebruiker kan aangeven of het licht en geluid aan moeten gaan als de gebruiker gewekt wil worden. Op de LCD zal ook te zien zijn of er iets aangepast wordt. De ingangs- en uitgangssignalen en het gedrag moeten geformuleerd worden als specificaties.

Er wordt structuur aangebracht in het systeem door het systeem op te delen in een paar grote blokken, deze blokken kunnen dan over de acht projectleden verdeeld worden. Allereerst moeten er van de afzonderlijke subsystemen specificaties opgesteld worden, zodat de blokken op elkaar afgestemd kunnen worden. Vervolgens moet van elk blok één of meer FSM's gemaakt worden waarna er een code geschreven kan worden. De geschreven code moet gesimuleerd en gesyntetiseerd worden. Als aan het eind van het project van het hele systeem een lay-out gemaakt is, kan het systeem op een chip gezet worden.

# 2

## ONTWERP SPECIFICATIE

Voor ons project ontwerpen we een klok, gesynchroniseerd met DCF77, weergeven op lcd met een wake-up alarm. De tijd, die intern wordt bijgehouden, zal worden gesynchroniseerd met een zogenaamd DCF signaal. De wekker zal bediend worden door middel van een menu. Dit menu wordt aangestuurd op basis van 4 knoppen. In dit menu moet de wekkertijd ingesteld worden. Ook moet de wekker en het wekkergeluid aan en uit gezet kunnen worden. Een vijfde knop is de uitknop voor als de wekker gaat en uitgezet moet worden. De visualisatie van dit menu zal op een LCD weergegeven worden. Als men zich niet in het menu bevindt, zal men alle data verdeeld over het scherm zien. Deze data bestaat uit de actuele tijd, de wekkertijd, de datum en de weekdag. Daarnaast zal op het LCD-scherm weergegeven worden of de wekker en het geluid aan staan. Met het knipperen van scheidingstekens tussen uren en minuten zal het passeren van seconden aangegeven worden.

Het systeem zal enkele reandvoorwaarden hebben. Zo zal het een algemene reset moeten bevatten. Als gevolg van het indrukken van een resetknop zullen alle opgeslagen waarden en counters op 'nul' worden gezet. Ook zullen alle signalen 'active high' moeten zijn. De implementatie van het totale ontwerp moet op een chip-oppervlak van circa  $0.4 \text{ cm}^2$ , oftewel 40.000 transistor-paren. Dit in de vorm van twee opeengestapelde bond\_bars, zoals aangeleverd door de TU Delft. De chip beschikt over 32 pins voor i/o poorten. Daar zitten niet de voedingspinnen bij, deze worden apart aangesloten. Voor de FSM's (Finite State Machine) mogen alleen die van het Moore-type gebruikt worden. Als de schakeling geactiveerd wordt moeten alle FSM's in hun begintoestand komen door middel van een reset signaal. Voor de opwekking van het kloksignaal kan gebruik gemaakt worden van een kristal van 6.144 MHz of 32 kHz. Het streven is om zo weinig mogelijk componenten extern te gebruiken. De dissipatie van de chip dient echter ook beperkt te zijn. Dit geeft een compromis voor de maximale stroom die de elektronica mag dissiperen voor de aansturing van externe componenten, zoals LEDs. De voedingsspanning van het IC bedraagt 5 Volt. Het IC wordt gemaakt in een semi-custom CMOS proces. [? ]

Het systeem zal de volgende ingangen hebben:

- DCF-signaal
- 36kHz klok
- Reset-knop
- 4 menu-knoppen
- 1 uit-knop

Onze chip zal over de volgende uitgangen beschikken:

- LED, 1 bit om de led aan te sturen
- Sound, 1 bit om de buzzer aan te sturen
- LCD, een 7 bits vector om het scherm aan te sturen via een microcontroller
- Clk\_out, 1 bit ter aansturing van de microcontroller

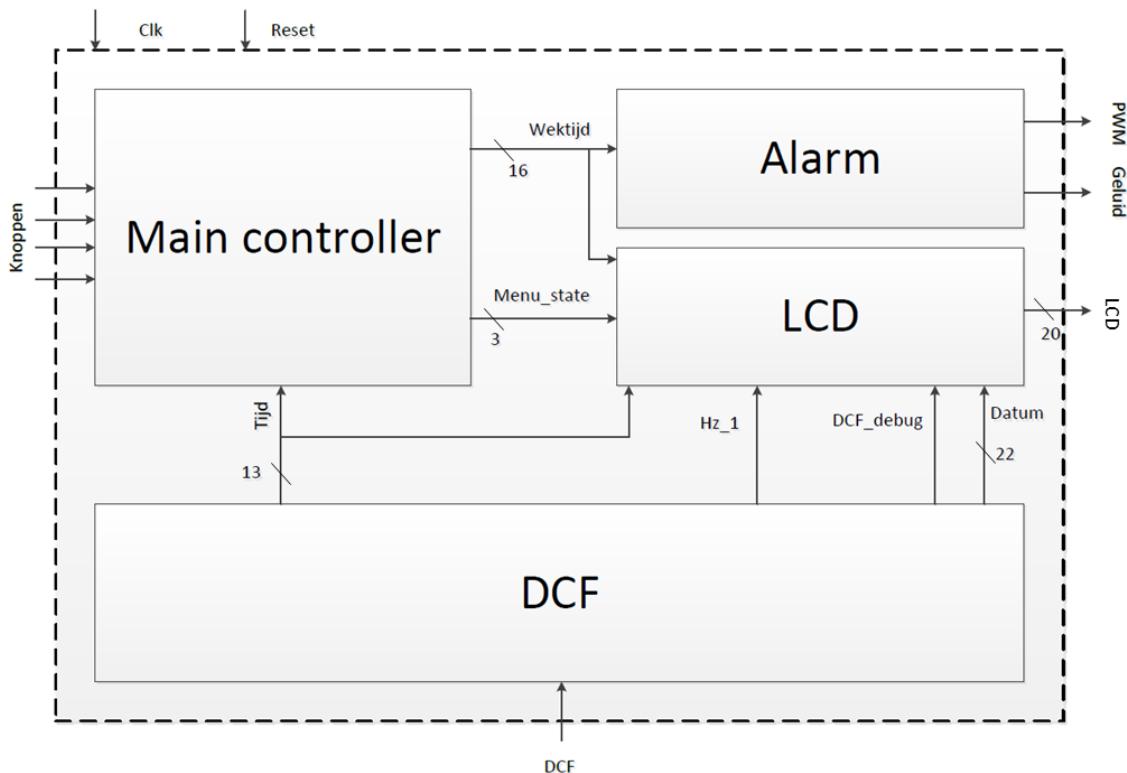
# 3

## SYSTEEM OVERZICHT EN ONTWERP

Het systeem is opgedeeld in vier blokken:

- De DCF controller
- De main controller
- Het alarm
- De LCD controller

In figuur 3.1 is te zien welke ingangs- en uitgangssignalen het systeem in en uit gaan en hoe de blokken elkaar aansturen.



Figuur 3.1: Blokdiagram van het gehele systeem

In de DCF controller wordt het DCF signaal opgevangen en omgezet naar een bitvector met weekdag, datum en tijd. Daarnaast wordt er een kloksignaal van 1 Hz gegenereerd. Mocht het DCF-signaal tijdelijk niet goed opgevangen kunnen worden, kan een intern register de tijd door blijven geven en op de LCD wordt aangegeven of het

DCF signaal opgevangen wordt. Dit register wordt dan weer gesynchroniseerd als het signaal weer opgevangen wordt.

De main controller bestuurt het hele systeem. De alarmtijd kan ingesteld worden en de alarmtijd wordt met de actuele tijd vergeleken, zodat het alarmblok weet wanneer het alarm aan moet gaan. Met knoppen kan het menu bediend worden.

In het alarmblok wordt eerst de vijftien minuten van de wekkertijd afgetrokken. De ingestelde tijd is namelijk de tijd waarop het geluid aan moet gaan, de lamp moet al een kwartier eerder beginnen met branden. Daarnaast zorgt het alarm ervoor dat een PWM-signaal gegenereerd wordt wat naar een LED gaat.

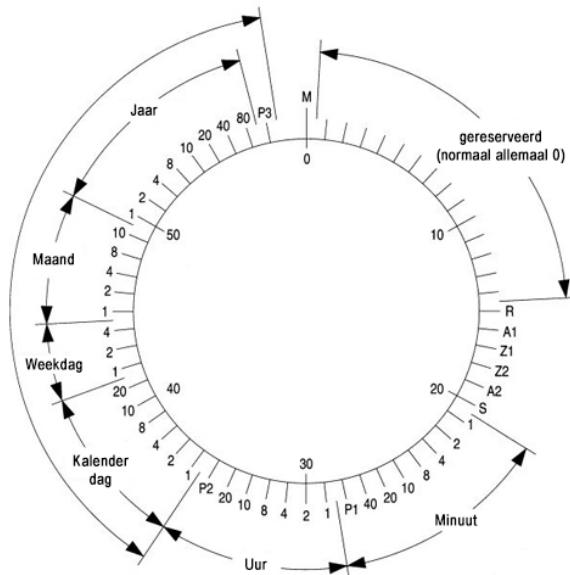
De LCD controller zorgt dat de datum, tijd, ingestelde alarmtijd en de veranderingen in het menu op de LCD zichtbaar zijn. Er wordt een LCD scherm gebruikt waar de pixels afzonderlijk van elkaar aangestuurd worden. Tussen de chip en het scherm zit nog een microcontroller, waarin de karakters zijn opgeslagen, dit zou namelijk te groot zijn om op de chip te regelen.

# 4

## DCF77

### 4.1. INLEIDING

In dit onderdeel, genaamd DCF77, wordt de basis van de wekker gelegd, door verschillende belangrijke datasignalen aan te maken, welke nodig zijn om de rest van de wekker goed te laten functioneren. Een eis die is gesteld aan de eigenschappen van deze klok, is dat deze gesynchroniseerd wordt met het zogenaamde DCF77 signaal. Dit is een signaal dat vanuit Duitsland wordt verzonden en bestaat uit korte (100 ms) en lange (200 ms) pulsen, welke respectievelijk nullen en enen coderen. Iedere seconde, behalve de laatste van iedere minuut, wordt er een puls verzonden. De bits die door deze pulsen worden gecodeerd, bevatten allerlei informatie, zoals de actuele datum en tijd op de eerstvolgende minuut. Een deel van de informatie die op deze manier wordt verzonden, zal worden gebruikt voor het aansturen van de wekker. Om gebruik te kunnen maken van de informatie die met het DCF77 signaal wordt verzonden, is het echter wel nodig te weten welke bit uit de reeks van 59 stuks welke informatie codeert. In figuur 4.1 is te zien welke informatie door elk van de 59 bits wordt gecodeerd. Een versie in tabelvorm is te vinden op Wikipedia [1]. De wekker zal gebruik gaan maken van bits 21 t/m 58. In de afbeelding worden binnen deze selectie bits 28, 35 en 58 respectievelijk P1, P2 en P3 genoemd. Deze bits zijn zogenaamde parity-bits, welke de ontvanger van het DCF77 signaal in staat stelt om tot op zekere hoogte te controleren of de ontvangen bitreeks correct is. In het DCF77 signaal wordt gebruik gemaakt van even parity. Dit betekent dat, wanneer zich in de bits die bij een zekere parity bit horen een even aantal logische enen bevindt, de parity bit een logische 0 zal zijn [2]. Het DCF77 blok converteert een gedigitaliseerde versie van het DCF77 signaal naar een tijdreferentie, waarna deze een autonome klok synchroniseert, welke zich ook binnen het DCF77 blok bevindt.



Figuur 4.1: Codering van het dcf-signaal [2]

## 4.2. SPECIFICATIES

In deze sectie worden de in- en uitgangen van de DCF-controller overzichtelijk weergeven. Doordat dit onderdeel aan het begin staat van het totale systeem, bevat dit blok enkel standaard ingangen en een ingang van buitenaf met het DCF77 signaal. De uitgangen uren en minuten worden doorgestuurd naar de main-controller. De clk van 1 Hz zal in verschillende onderdelen worden gebruikt, zowel binnen als buiten het DCF77 blok. De datum en het signaal dcf\_led zullen rechtstreeks op het LCD scherm worden weergegeven. Enkele signalen zijn in BCD (Binary Coded Decimal). Meer informatie over BCD is te vinden op de website van Technology UK [3].

### 4.2.1. INGANGEN

Dit onderdeel maakt gebruik van de volgende ingangen:

- De 32 kHz systeemklok, een standaard input.
- Het 'active high' resetsignaal, een standaard input.
- Het gedigitaliseerde DCF77 signaal, bestaande uit korte en lange pulsen.

### 4.2.2. UITGANGEN

Dit onderdeel genereert de volgende uitgangen:

- Een kloksignaal met een frequentie van 1 Hz, welke gebruikt kan worden om secondes te tellen.
- Het debug signaal dcf\_led, wat een seconde lang hoog is na ontvangst van een puls van het DCF77 signaal.
- Uren; de uren van de huidige tijd in een BCD vector van 6 bits.
- Minuten; de minuten van de huidige tijd in een BCD vector van 7 bits.
- Weekdag; de dag van de week, binair gecodeerd met maandag als 001.
- Dag; de dag van de maand in een BCD vector van 6 bits.
- Maand; het nummer van de maand in een BCD vector van 5 bits.
- Jaar; de laatste twee cijfers van het jaartal in een BCD vector van 8 bits.
- Date\_ready; een signaal dat aangeeft dat de datum gereed is voor verder gebruik.

### 4.2.3. GEDRAG

Het DCF77 blok heeft als belangrijkste gedragsfunctie om de huidige tijd en datum door te geven. Om deze informatie zo precies mogelijk te houden, dient deze zo vaak mogelijk te worden gesynchroniseerd met het extern gedigitaliseerde DCF77 ingangssignaal. Idealiter zou er dus iedere minuut met het DCF77 signaal worden gesynchroniseerd. Pas als via de parity bits is gebleken dat het de ontvangen tijd en datum plausibel zijn, wordt dit echter gedaan. Anders blijft de datum onveranderd en wordt de tijd bijgehouden met een interne klok. Zo wordt voorkomen dat andere onderdelen op de chip tijdelijk een compleet verkeerd signaal krijgen doorgestuurd, indien het DCF-signaal signaal niet of slecht wordt ontvangen.

Naast deze belangrijkste functie, heeft het DCF77 blok ook nog enige kleinere taken. Zo dient het 1 Hz signaal dat afkomstig is uit een interne klokdelner en wordt gebruikt voor de interne klok ook beschikbaar te worden gesteld voor gebruik in andere blokken. Ook dient een debug signaal dcf\_led gegenereerd te worden, dat na iedere ontvangen bit uit het DCF77 signaal een seconde lang hoog is. Ten slotte dienen alle subblokken, inclusief registers, van de gehele module bij een 'active high' reset gereset te worden.

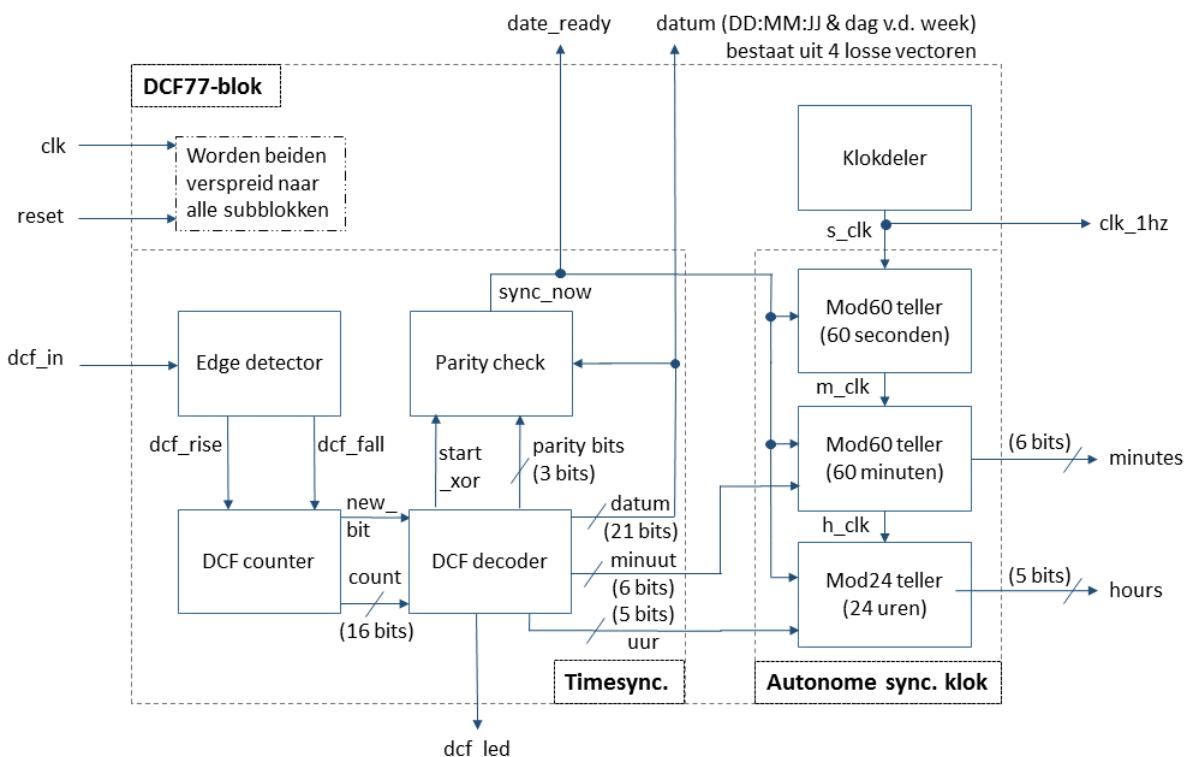
## 4.3. FUNCTIONALITEIT

Het DCF blok dient volgens de specificaties een tijd- en dagstempel te genereren uit het gedigitaliseerde DCF77 signaal dat aan het blok wordt aangeboden. Bovendien dient het blok de tijd zelf bij te houden wanneer het DCF77 signaal niet of niet goed wordt ontvangen. Het is vrij ondoenlijk om deze volledige functionaliteit in één keer te implementeren in VHDL. Bovendien zou dit een groot, lomp blok opleveren in de layout, welke vervolgens lastig op de chip te plaatsen zal zijn. Daarom wordt het DCF blok opgedeeld in kleinere subblokken, totdat deze wel in één keer geïmplementeerd kunnen worden. Een top-level beschrijving knoopt vervolgens de kleinere subblokken weer aan elkaar tot een groot blok. Naast het voordeel dat dit het ontwerpen vergemakkelijkt, geeft dit ook een gemakkelijker op de chip te plaatsen ontwerp, omdat ook de layout op dezelfde hiërarchische manier gegenereerd zal worden.

In fig. 4.2 is te zien hoe het DCF blok is verdeeld in subblokken. Het DCF77 signaal wordt allereerst aangeboden aan het subblok edge detector, welke dit signaal vervolgens opsplitst in twee afzonderlijke signalen. Het eerste signaal geeft een puls wanneer er een rising edge plaatsvindt op het DCF77 signaal en het tweede signaal geeft een puls wanneer er een falling edge plaatsvindt op het DCF77 signaal. Beide signalen worden doorgevoerd naar de DCF counter, welke het tijdsverschil tussen de rising en falling edges telt. De tellerwaarde wordt na iedere ontvangen puls beschikbaar gesteld aan de daadwerkelijke DCF decoder door het signaal new\_bit hoog te maken.

De DCF decoder bepaald vervolgens wat de bitreeks is die in één minuut van het DCF77 signaal gecodeerd is en genereert hieruit de signalen voor de tijd en datum. Ook de parity bits worden apart naar buiten gevoerd, zodat deze kunnen worden gebruikt in de parity check. Naast deze signalen genereert de decoder ook het debug signaal dcf\_led, dat aangeeft of het DCF77 signaal goed wordt ontvangen. Ten slotte genereert de decoder een signaal "start" dat aangeeft wanneer een volledige minuut is gedecodeerd. Dit laatste signaal gaat, samen met de tijd-, datum- en parity bits naar het subblok parity check. Hier wordt gecontroleerd of het aantal enen (even of oneven) klopt met wat het parity bit aangeeft. Het parity bit is namelijk alleen 0 wanneer er een even aantal bits bij hoort. Het controleren op een even of oneven aantal enen gebeurt door middel van een herhaalde xor operatie. Het subblok parity check genereert vervolgens een sync\_now signaal dat aangeeft dat de controle is voltooid en succesvol was. Dit signaal wordt ook naar buiten gevoerd om aan te geven dat de datum aan de uitgang van de decoder gebruikt kan worden.

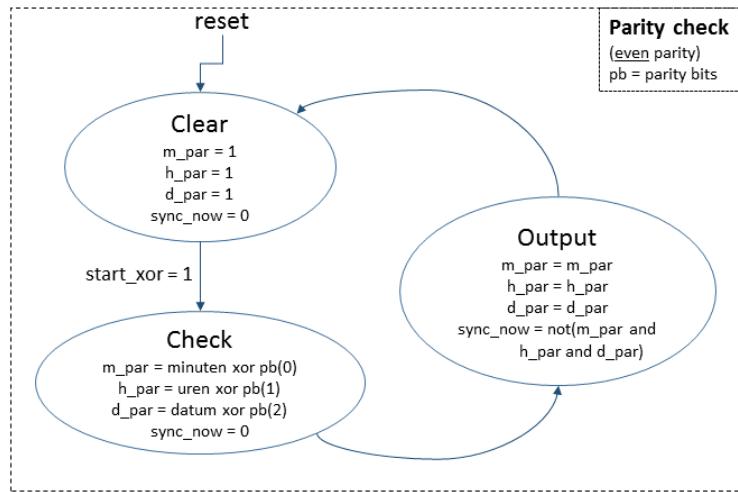
Het sync\_now signaal komt vervolgens aan bij de autonome synchroniseerbare klok. Dit subblok bestaat vervolgens zelf weer uit twee mod60 tellers en een mod24 teller, waarmee autonomoos de tijd kan worden bijgehouden, mocht het DCF77 signaal onverhoopt wegvalLEN. Wanneer sync\_now hoog wordt, worden deze tellers gesynchroniseerd met de tijd uit het DCF77 signaal. Voor het tellen wordt gebruik gemaakt van het uitgangssignaal van een ander subblok, namelijk de klokdeler. Dit subblok genereert het 1 Hz kloksignaal uit de 32 kHz systeemklok, wat wordt gebruikt voor het tellen van seconden en bovendien naar andere blokken buiten het DCF blok wordt doorgevoerd.



Figuur 4.2: Verdeling van het DCF blok in subblokken

### 4.3.1. FSM DIAGRAMMEN

Het opdelen van het grote DCF77 blok in subblokken is bijzonder nuttig, maar nog niet voldoende om direct een implementatie in VHDL te kunnen maken. Daarom wordt voor alle subblokken op het laagste abstractieniveau een FSM diagram gemaakt, waarin globaal wordt aangegeven wat er wanneer dient te gebeuren binnen elk van deze blokken. Er zijn in totaal acht verschillende subblokken te ontwerpen. Dit betekent dat er ook acht verschillende FSM diagrammen dienen te worden getekend. Bij wijze van voorbeeld wordt hier het FSM diagram van de parity\_check besproken. Alle acht FSM diagrammen zijn terug te vinden in appendix A als fig. A.1 t/m fig. A.8.



Figuur 4.3: FSM diagram van het subblok parity check

In fig. 4.3 is het FSM diagram van de parity check nogmaals weergegeven. Zoals te zien is, begint deze na een reset altijd in de clear state. In tegenstelling tot wat je zou verwachten, worden in deze state de drie bits die het resultaat van de drie parity checks bevatten, niet op een logische 0, maar op een logische 1 gezet. Dit is zo, omdat in het DCF77 signaal gebruik wordt gemaakt van even parity. Dit betekent dat een parity bit een logische 0 bevat, wanneer de bijbehorende bitreeks een even aantal logische enen bevat. Een xor operatie met deze bitreeks + de bijbehorende parity bit zal dus 0 opleveren wanneer de parity bit correct is. In de reset ga je er echter juist vanuit dat de parity nog niet correct is. Wanneer dit door de decoder wordt aangegeven met het signaal start\_xor, zal de state van het subblok parity\_check naar check gaan.

In de state check worden vervolgens, door middel van herhalde xor operaties, de daadwerkelijke parity checks uitgevoerd. Één check controleert de minuten, een tweede controleert de uren en de derde controleert de volledige datum (inclusief dag van de week). Onafhankelijk van het resultaat van deze drie parity checks, zal het subblok vervolgens gelijk naar de state output gaan.

In de state output worden dan de resultaten van de drie parity checks bij elkaar genomen. Dit gebeurt, om zo een strengere controle op de correctheid van de ontvangen bits te verkrijgen. Bovendien is op deze manier slechts één signaal nodig dat aangeeft of de ontvangen tijd en datum correct zijn. Alleen als alle parity checks kloppen, zal het uitgangssignaal sync\_now hoog worden gemaakt. Hiermee worden vervolgens de tijd en datum vrijgegeven voor verder gebruik in de wekker. Ten slotte zal het subblok direct na het uitzenden van een puls op het sync\_now signaal weer teruggaan naar de state clear, zodat deze klaar staat om de volgende minuut weer de parity checks uit te voeren. De cirkel is nu rond.

### 4.3.2. VHDL CODE

Met behulp van de FSM diagrammen kan er behavioural VHDL worden geschreven, welke vervolgens met behulp van structural VHDL aan elkaar kan worden gemaakt om zo uiteindelijk het hele DCF77 blok te vormen. Al deze behavioural en structural VHDL beschrijvingen zijn voorzien van enig commentaar en opgenomen in appendix B.1. Vanuit de top-level beschrijving in appendix B.1.11 is, eventueel met behulp van fig. 4.2, gemakkelijk terug te vinden hoe alle andere VHDL beschrijvingen binnen het blok samenwerken.

## 4.4. SIMULATIE

Om de correcte functionaliteit te toetsen aan de verwachtingen, wordt de VHDL beschrijving van het DCF77 blok veelvuldig gesimuleerd. Hiervoor dienen echter wel testbenches geschreven te worden, welke het te testen onderdeel voorzien van testsignalen. Deze testbenches zijn voor ieder apart onderdeel geschreven in twee varianten, ook voor de structural beschrijvingen. In de eerste variant wordt er vanuit gegaan dat de systeemklok, net als in het uiteindelijke product, een signaal van 32 kHz is. Op deze variant van de testbenches wordt in dit verslag verder niet ingegaan, omdat deze te lang zijn. Dit levert later in het proces problemen op tijdens het simuleren op transistorniveau. Daarom is er ook een tweede variant van alle testbenches geschreven, waarin gebruik wordt gemaakt van geschaalde ingangssignalen. Op deze manier is er een veel kortere simulatietijd nodig, terwijl toch uit de resultaten opgemaakt kan worden of het geteste onderdeel naar verwachting werkt. Deze testbenches zijn opgenomen in appendix B.2 als appendix B.2.1 t/m appendix B.2.11

In *Modelsim* worden vervolgens alle onderdelen van het DCF77 blok getest op behavioural niveau met behulp van deze testbenches. Zodra de resultaten van deze simulaties overeenkomen met de verwachte resultaten, worden alle VHDL beschrijvingen in het programma *GoWithTheFlow* gesynthetiseerd. De gesynthetiseerde VHDL kan vervolgens met dezelfde testbenches opnieuw in *Modelsim* worden gesimuleerd. Als alles goed is, zullen de resultaten er precies hetzelfde uitzien. In dat geval wordt er van alle subblokken een layout gegenereerd door de programma's *Madonna & Trout*. Uit deze layout kan vervolgens binnen *GoWithTheFlow* weer VHDL worden geëxtraheerd. Deze VHDL kan vervolgens opnieuw met dezelfde testbenches in *Modelsim* worden gestopt ter controle. Ook de resultaten van deze simulaties zullen als alles klopt precies hetzelfde zijn als die van de behavioural simulaties. Binnen *GoWithTheFlow* is het echter ook mogelijk om aan de hand van een testbench de gegenereerde layout direct, zonder extractie van VHDL, te simuleren. Dit heet ook wel de switch-level simulatie, omdat deze het ontwerp op transistorniveau simuleert. Ten slotte is het binnen *GoWithTheFlow* mogelijk om de resultaten van de switch-level simulatie te laten vergelijken met die van de behavioural simulatie. Als ze met elkaar overeen komen, mag deze vergelijking slechts één enkele error opleveren; namelijk direct aan het begin van de simulatie, wanneer er nog geïnitialiseerd wordt.

## 4.5. RAPID PROTOTYPING (FPGA IMPLEMENTATIE)

Wanneer de testbenches eenmaal geschreven zijn, is het teseten van de geschreven VHDL beschrijvingen door middel van simuleren vrij snel voor elkaar. De resultaten van deze simulaties geven een aardige indicatie van de kans dat een ontwerp ook op de Sea-of-gates chip zal gaan werken. Er is echter nog een manier van testen, welke de resultaten van de simulaties kan ondersteunen. Dit is het plaatsen van het ontwerp op een FPGA development board, zoals het *Altera DE1* bord, waarna met behulp bijvoorbeeld de schakelaars, knoppen en ledjes van het development board het ontwerp realtime kan worden getest. Door ook op deze manier te testen, kunnen mogelijk andere problemen naar voren komen, welke in de simulaties niet worden opgemerkt.

In het specifieke geval van het DCF77 blok, zijn voor het testen op het FPGA bord nog enige extra VHDL beschrijvingen nodig. Deze extra beschrijvingen zijn voor onderstaande functies van belang en zijn opgenomen in appendix B.3 als appendix B.3.1 t/m appendix B.3.7.

- Het 50 MHz interne kloksignaal van het FPGA bord delen tot de 32 kHz van het ontwerp. [gen32khz]
- Een DCF signaal genereren wat iedere minuut opnieuw wordt herhaald. [dcf\_gen] m.b.v. [gen10hz]
- Het wel of niet aansluiten van het gegenereerde DCF signaal op het ontwerp. [dcf\_switch]
- Vanwege het beperkte aantal ledjes op het FPGA bord; schakelen tussen dag, maand en jaar. [switch]
- Om het uitgangssignaal date\_ready af te kunnen lezen op een ledje, wordt dit signaal gebufferd. [buff]
- De huidige tijd op het seven-segment display van het FPGA bord weergeven. [sevenseg]

Uiteindelijk is met behulp van deze extra bestanden het volgende waar te nemen op het FPGA development board: Na de reset knop te hebben ingedrukt, zal op het seven-segment display te zien zijn dat de tijd is gereset naar 00:00. Ook de datum, welke kan worden weergegeven op de ledjes van het bord, is gereset naar enkel nullen. Door de daarvoor aangewezen schakelaar om te halen, wordt het gegenereerde DCF signaal aangesloten op het ontwerp en zal de tijd na een initialisatieperiode van één tot twee minuten gesynchroniseerd worden naar de in het signaal gecodeerde waarde. Ook zal het ledje dat is toegewezen aan date\_ready nu gaan branden, als teken dat de datum nu beschikbaar is voor weergave op de ledjes. Wanneer de schakelaar van het DCF signaal weer wordt teruggezet, zal de tijd verder lopen op de interne klok. Op het seven-segment display is dit te zien aan het feit dat de weergegeven tijd iedere minuut met een minuut wordt opgehoogd.

## 4.6. RESULTATEN

Van alle simulaties die zijn gedaan, zijn de resultaten gecontroleerd en opgeslagen. In appendix C is van ieder subblok van het DCF77 blok ter illustratie van de functionaliteit het resultaat van de behavioural simulaties op schaal weergegeven. Van het DCF77 blok als geheel is bovendien ook het resultaat van de switch-level simulatie opgenomen. Wanneer deze resultaten worden vergeleken met de eisen die aan het DCF77 blok gesteld waren, kan er geconcludeerd worden dat de ontworpen VHDL beschrijving voldoet aan deze eisen. Deze conclusie wordt bovendien onderbouwd door het feit dat het blok ook als prototype op het FPGA bord naar verwachting functioneert. Of deze conclusie terecht is, zal pas duidelijk worden wanneer de daadwerkelijke Sea-of-gates chip gefabriceerd is.

# 5

## MAIN CONTROLLER

### 5.1. INLEIDING

De main controller bevat de interface van de wekker. Deze zorgt er voor dat een wekker ingesteld kan worden, aangepast kan worden en uitgezet kan worden. Belangrijk aan elke interface is dat deze gebruiksvriendelijk is. Dit kan onder andere bereikt worden door een optimum voor het aantal knoppen te bepalen. Te veel knoppen, en de gebruiker weet niet welke knop wat doet, te weinig knoppen, en de gebruiker moet navigeren door een nodeloos ingewikkeld menu.

Daarnaast is er nog een beperkende factor: het aantal pinnen op de chip.

Dit alles bijeengenomen leverde een zo gebruiksvriendelijk mogelijke interface als er gebruik wordt gemaakt van 4 knoppen. Daarnaast is er nog een knop die slechts gebruikt wordt om een afgaand alarm uit te zetten.

De controller stuurt een hoop dingen aan, en van te voren was al geanticipeerd dat dit hierdoor een van de grootste onderdelen op de chip zou kunnen worden.

### 5.2. SPECIFICATIES

De controller moet aan een aantal eisen voldoen zodat andere componenten overweg kunnen met de uitgangssignalen van de controller. De tijd moet gecodeerd worden volgens het DCB codering. Om door te geven of het alarm of onderdelen aan of uit zijn wordt gebruik gemaakt van positieve logica. Dit houdt in als er een 1 als uitgangsignaal is dan staat het onderdeel aan. Het reset gebeurt bij een hoog signaal en synchroon met de klok.

#### 5.2.1. INGANGEN

- Klok, dit is een standaard input;
- Reset, ook dit is een standaard input;
- Knoppen, dit zijn de 4 knoppen die (nadat ze gebufferd zijn) onderdeel zijn van de interface.
  - knoppen[0] = menu
  - knoppen[1] = set
  - knoppen[2] = up
  - knoppen[3] = down

#### 5.2.2. UITGANGEN

- Wekker, dit is een vector van 16 lang. In dit signaal zijn alle instellingen te vinden van hoelaat de wekker afgaat tot of het alarm aan of uit is.
- Menu, een vector van 3 lang hierin is gecodeerd welke state de gebruiker is zodat de lcd de output eraan kan aanpassen.

In tabel 5.1 staat wat voor informatie te vinden is in de uitgangen van de controller.

Tabel 5.1: Uitgangen van de controller

Uitgang	Informatie over wat in de uitgang te vinden is
wekker	De huidige info over de wekker instellingen uit geheugen wekker[6 down to 0] daarin staan de minuten wekker[12 down to 7] daarin staan de uren wekker[13] geluid bit wekker[14] led bit wekker[15] wekker bit (Of de wekker überhaupt aan is of niet)
menu	Deze geeft door aan de in welke state we zitten aan de lcd module 000 : Het normale scherm weergeven met alarm en wekkertijd weergave states : (Rust, wekker toggle) 001 : Uren aanpassen states : (uren set, uren min/plus) 010 : Minuten aanpassen states : (minuten set, minuten min/plus) 011 : Led aanpassen states : (Led, Led toggle) 100 : Geluid aanpassen states : (Geluid, Geluid toggle) 101 : De menu is geopend states : (Wekkertijd)

### 5.2.3. GEDRAG

Om te beginnen moet de tijd waarop de wekker af gaat ingesteld kunnen worden. Dit wordt gedaan door eerst de huidige wekkertijd weer te geven, vervolgens het uur waarop gewekt moet worden te wijzigen en daarna de minuten. Hierna wordt de huidige tijd weer weergegeven.

Daarnaast is een vereiste dat de alarm in het geheel of gedeeltelijk uitgezet kan worden. Afhankelijk van een instelling moet het niks gebeuren of een combinatie van het afgaan van de led en geluid.

Dit alles moet gebruiksvriendelijk mogelijk zijn.

## 5.3. FUNCTIONALITEIT

Om de controller goed hiërarchisch te kunnen ontwerpen is deze opgedeeld in 3 componenten. Een buffer voor de vier knoppen, geheugenelement en het menu zelf. In fig. 5.1 is te vinden hoe de blok diagram van de controller eruit ziet.

Voor het menu is een grote FSM die in fig. 5.2 staat de gemaakte fsm en in tabel 5.2 staan de uitgangen per state gespecificeerd.

Rust	enable = '0' wekker=wekdata menu= "000"
Wekker toggle	enable = '1' wekker[12 down to 0]=wekdata[12 down to 0] wekker[13]= niet wekdata[13] menu = "000"
Wekkertijd	enable ='0' wekker=wekdata menu = "000"
Led	enable ='0' wekker=wekdata menu = "011"
Led toggle	enable ='1' wekker[11 down to 0]=wekdata[11 down to 0] wekker[12] = niet wekdata[12] wekker[13] = wekdata[13] menu = "011"
Geluid	enable ='0' wekker=wekdata menu = "100"

Geluid toggle	enable = '1' wekker[10 down to 0]=wekdata[10 down to 0] wekker[11] = niet wekdata[11] wekker[13 downto 12] = wekdata[13 downto 12] menu = "100"
Uren set	enable = '0' wekker=wekdata menu = "001"
Uren plus	enable = '1' wekker=wekdata+1 menu = "001"
Uren min	enable = '1' wekker=wekdata-1 menu = "001"
Minuten set	enable = '0' wekker=wekdata menu = "010"
Minuten plus	enable = '1' wekker=wekdata+1 menu = "010"
Minuten min	enable = '1' wekker=wekdata-1 menu = "010"

Tabel 5.2: Uitgangen binnen de state van de controller

## 5.4. VHDL CODE

De code voor de controller van de wekker is te vinden in appendix [B.4](#). Voor de overzicht en het modular opbouwen is de code in vier blokken geschreven.

- De top entity met de port map. Deze is te vinden in appendices [B.4.1](#) en [B.4.2](#).
- Het menu, hierin zit de echte logica verwerkt. Deze is te vinden in appendices [B.4.3](#) en [B.4.4](#).
- Het gebruikte geheugen element voor de opslag van 16 bits, te vinden in appendices [B.4.5](#) en [B.4.6](#).
- De gebruikte buffer is te vinden in appendices [B.4.7](#) en [B.4.8](#). De buffer regelt het ingangssignaal, en zorgt ervoor dat er maar 1 klokperiode lang een hoog signaal gelezen word.

Voor het testen van de code zijn er testbenches gemaakt welke te vind zijn in appendices [B.5.1](#) tot [B.5.4](#).

## 5.5. TESTEN

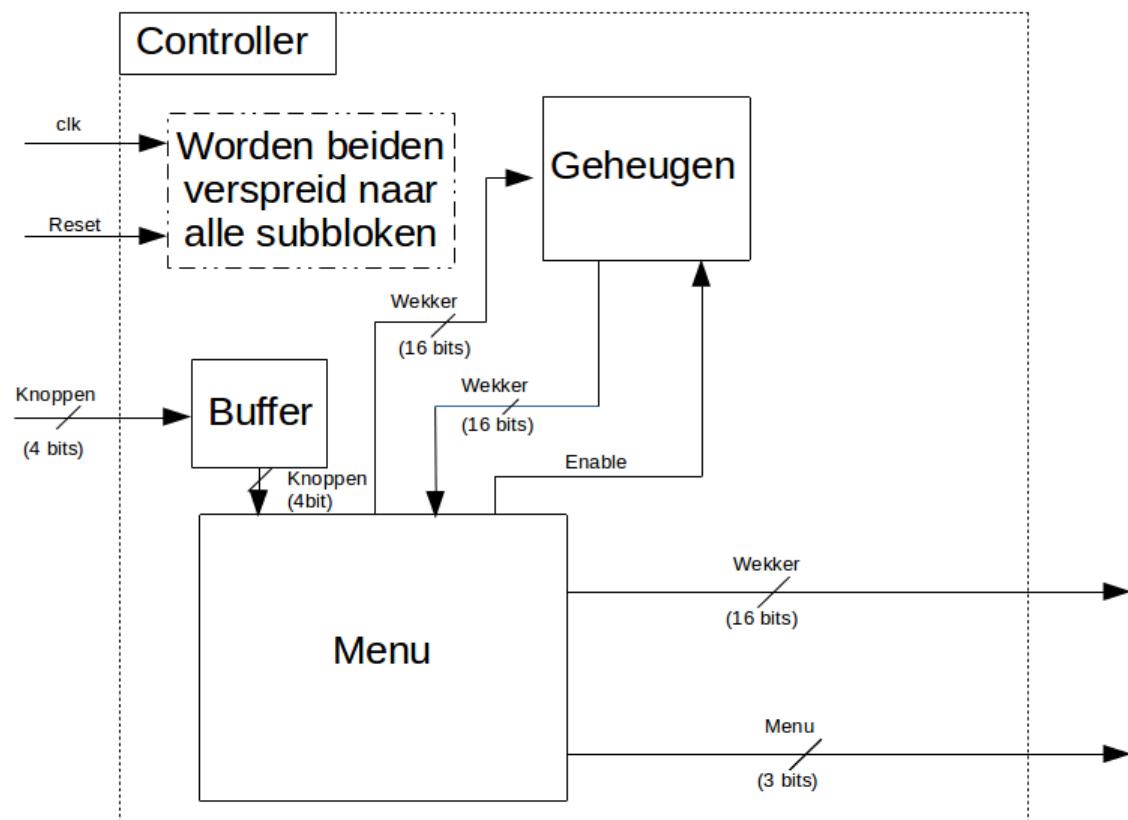
Om zeker te zijn dat alles goed werkt worden er vier verschillende testen uitgevoerd. De belangrijkste zijn de behavioural en switch level simulatie. Om zeker te zijn dat alles goed werkt wordt er ook nog een test gedaan door de code te synthetiseren en te extracten uit de schakeling. Bij een test op behavioural niveau wordt er gekeken of de code werkt zoals verwacht. Na een geslaagd resultaat kan de code worden gesynthetiseerd, en deze gesynthetiseerde code worden gesimuleerd. Als er geen fouten optreden kan het ontwerp gemaakt worden, daarna geëxtraheerd en nogmaals getest worden. De testen worden uitgevoerd met behulp van *Modelsim*. De laatste stap bevat de switch level simulatie deze is als bijlage ingeleverd bij het document.

## 5.6. SIMULATIE

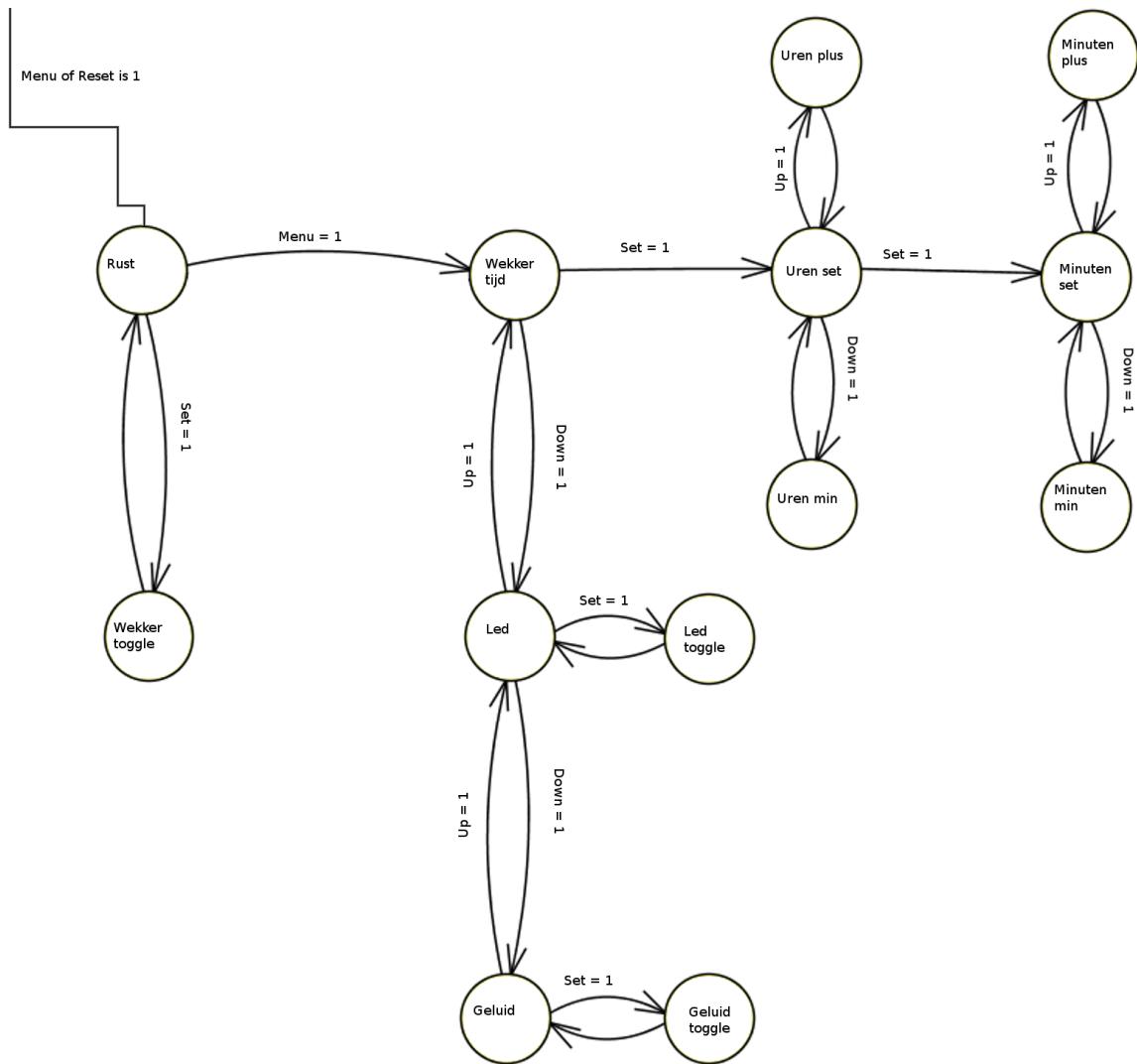
De resultaten van de simulatie staan in appendix [D](#). De testbench is te lang om in een keer weer te geven daarom is deze op geknipt in vier stukken. De testbench die gemaakt is voor de simulatie staat in appendix [B.5.1](#)

## 5.7. RESULTATEN

Van fig. [D.1](#) tot en met fig. [D.12](#) is te zien dat iedere simulatie tot hetzelfde resultaat leid en daarmee succesvol is. De minimale klokperiode kan afgelezen worden aan de hand van fig. [D.13](#). Hieruit is op te maken dat deze 60ns is.



Figuur 5.1: Blok diagramma van de controller



Figuur 5.2: FSM diagramma van de menu

### 5.7.1. CONCLUSIE EN DISCUSSIE

De controller werkt op alle gesimuleerde niveau's naar verwachting. Voor een correcte werking heeft de controller een klok periode die volgens de simulaties ten minste 60ns moet bedragen om gliches te voorkomen. De controller maakt gebruik van 8606 transitoren waarvan er voor de daadwerkelijke logica slechts 2965 worden gebruikt. Vlak nadat het inputbuffer gemaakt was kwam men er achter dat in plaats van een buffer ook de *rising\_edge* functie gebruikt had kunnen worden. Hier is niet voor gekozen omdat het dan niet altijd even duidelijk is wat er gebeurd als er twee knoppen tegelijk ingedrukt worden.

# 6

## ALARM

### 6.1. INLEIDING

In de alarm module wordt een led aangestuurd, die 15 minuten voor de ingestelde tijd in de main controller begint met branden en steeds feller wordt naarmate de tijd verstrijkt. Als de huidige tijd gelijk is aan de ingestelde tijd brandt de led op z'n felst en gaat er een geluid af, totdat er een knop wordt ingedrukt.

### 6.2. SPECIFICATIES

#### 6.2.1. INGANGEN

- Klok, standaard input.
- Reset, standaard input.
- Tijd-uur, huidige tijd in uren.
- Tijd-minuut, huidige tijd in minuten.
- Wekker-uur, uur ingesteld in de main controller.
- Wekker-min, minuten ingesteld in de main controller.
- Sec, seconde signaal gegenereerd in de DCF controller.
- Knop, alarm uitschakelen.

#### 6.2.2. UITGANGEN

- PWM-signaal, signaal om de led aan te sturen.
- Geluid, signaal om een geluid af te laten gaan.

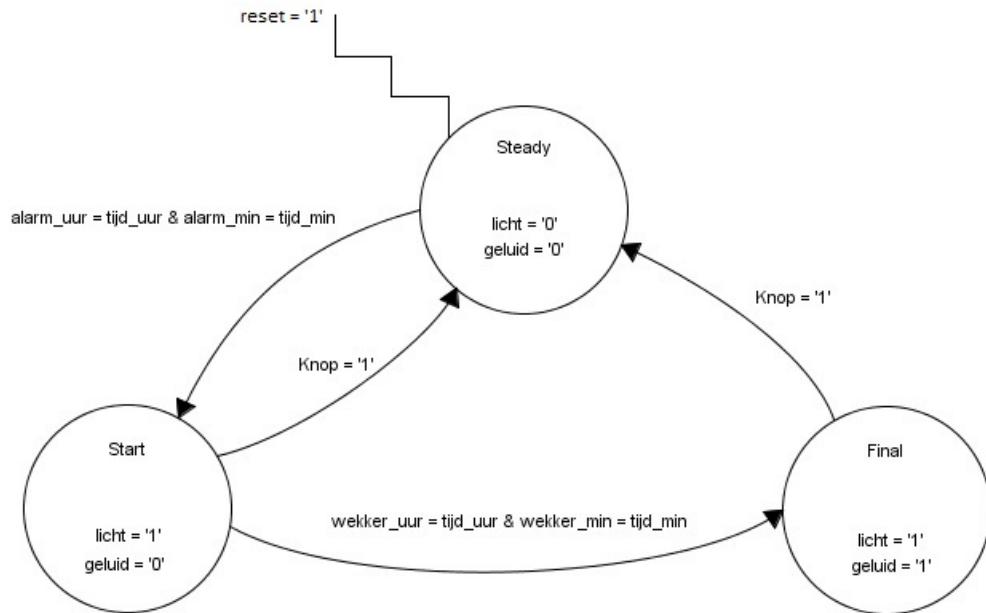
#### 6.2.3. GEDRAG

Het alarm moet een bepaalde tijd voordat de wekker is ingesteld aangaan, nu gekozen voor 15 minuten. Er wordt 15 minuten van de ingestelde tijd afgetrokken. Zodra die tijd gelijk is aan de huidige tijd komt er een signaal (licht) aan bij het gedeelte wat voor een pwm signaal zorgt. In dat gedeelte wordt een pwm signaal gegenereerd dat elke 15 seconde breder wordt. Dit wordt gedaan door in een counter 15 seconde te tellen. Elke 15 seconde wordt de variable "length" kleiner. Deze begon op 64 en wordt vergeleken met een andere counter die elke klokflank telt, tot 64. Als de counter groter of gelijk is aan "length" dan is het pwm-signaal hoog. Als 15 minuten zijn verstreken na het aangaan van de led, dus de ingestelde tijd is gelijk aan de huidige tijd, brandt de led op z'n felst. Ook zal dan een "geluid"signaal naar '1' gaan. Dit blijft zo totdat de knop wordt ingedrukt of alles wordt gereset.

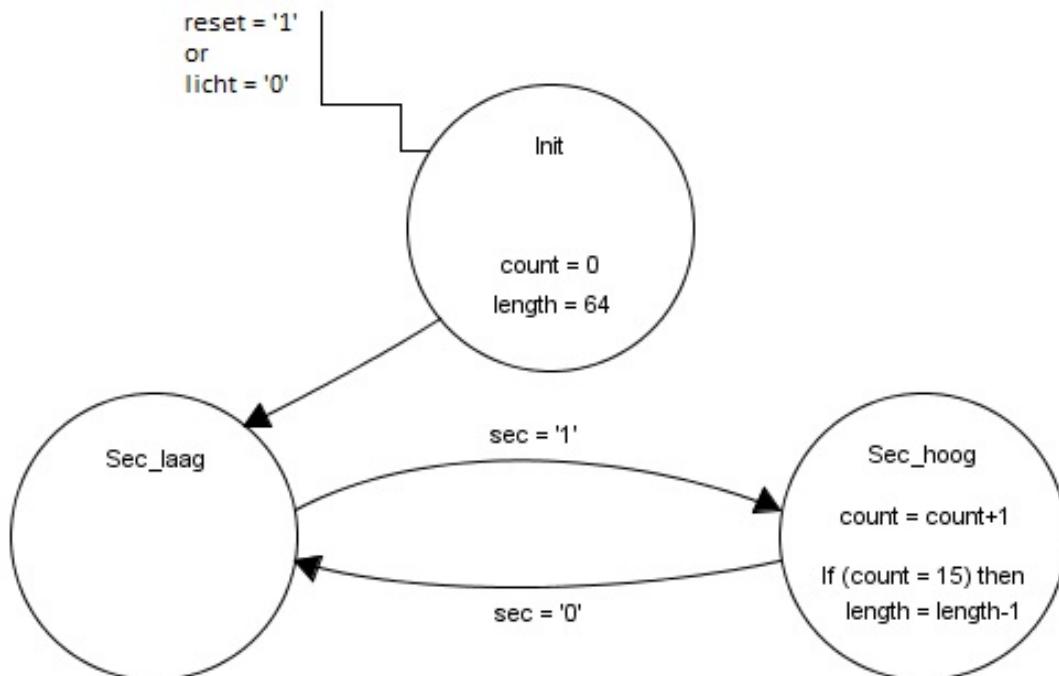
## 6.3. FUNCTIONALITEIT

Om te laten zien hoe het alarm werkt, zijn er FSM's gemaakt en is tevens VHDL code inbegrepen.

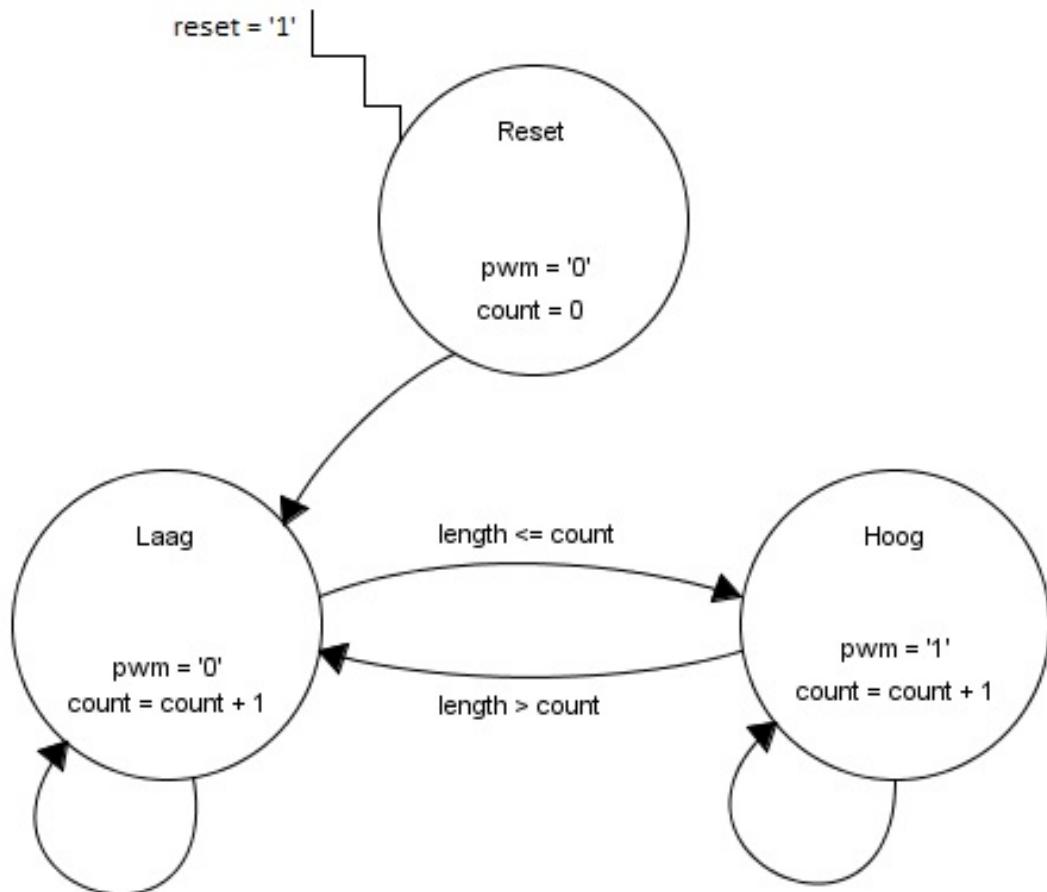
### 6.3.1. FSM



Figuur 6.1: Alarm Compare FSM



Figuur 6.2: Alarm counter FSM



Figuur 6.3: Alarm pwm FSM

### 6.3.2. CODE

De code voor het alarm is opgedeeld in 3 stukken:

- alarm-compare
- alarm-counter
- alarm-pwm

Alarm-counter en alarm-pwm zijn onderdeel van een top entity, alarm. Omdat het nog niet zeker is waar alarm-compare geplaatst gaat worden op de chip is die daar niet bij inbegrepen. De code voor het alarm is te vinden in appendix B.6.

- De entity en behavioural van alarm compare is te vinden in appendices B.6.1 en B.6.2.
- De top entity en port map van alarm is te vinden in appendices B.6.3 en B.6.4.
- De entity en behavioural van alarm-counter is te vinden in appendices B.6.5 en B.6.6.
- De entity en behavioural van alarm-pwm is te vinden in appendices B.6.7 en B.6.8.

## 6.4. RESULTATEN

Alle onderdelen werken in de simulaties. Zowel de simulatie van de behaviour als van de extracted vhdl. De simulaties zijn te vinden in appendix E.

Te zien is in fig's. E.1 en E.3 dat wanneer de huidige tijd (tijd\_uur en tijd\_min) gelijk is aan de wekker tijd (wekker\_uur en wekker\_min) minus 15 minuten, dan gaat het signaal licht naar '1'. Als de huidige tijd gelijk is aan de wekker tijd, gaat ook het geluid signaal naar '1'. Ook is te zien dat wanneer de knop (in de simulatie stop\_alarm) naar '1' gaat het licht en/of geluid weer naar '0' gaat.

fig's. E.2 en E.4 laten zien dat op het moment dat er 15 seconde zijn verstreken het pwm-signaal breder wordt.

## 6.5. TESTEN

Nadat alles er goed uitzag bij de simulaties is het alarm uitvoerig getest op de FPGA. Hier bleek alles ook prima te werken. Er was te zien dat een led steeds feller werd, dus het pwm-signaal werd steeds breder. Ook was te zien dat als de huidige tijd gelijk was aan de ingestelde wekker tijd minus 15 minuten er een signaal naar de pwm generator gaat om die aan te zetten.

# 7

## LCD CONTROLLER

### 7.1. LCD CONTROLLER

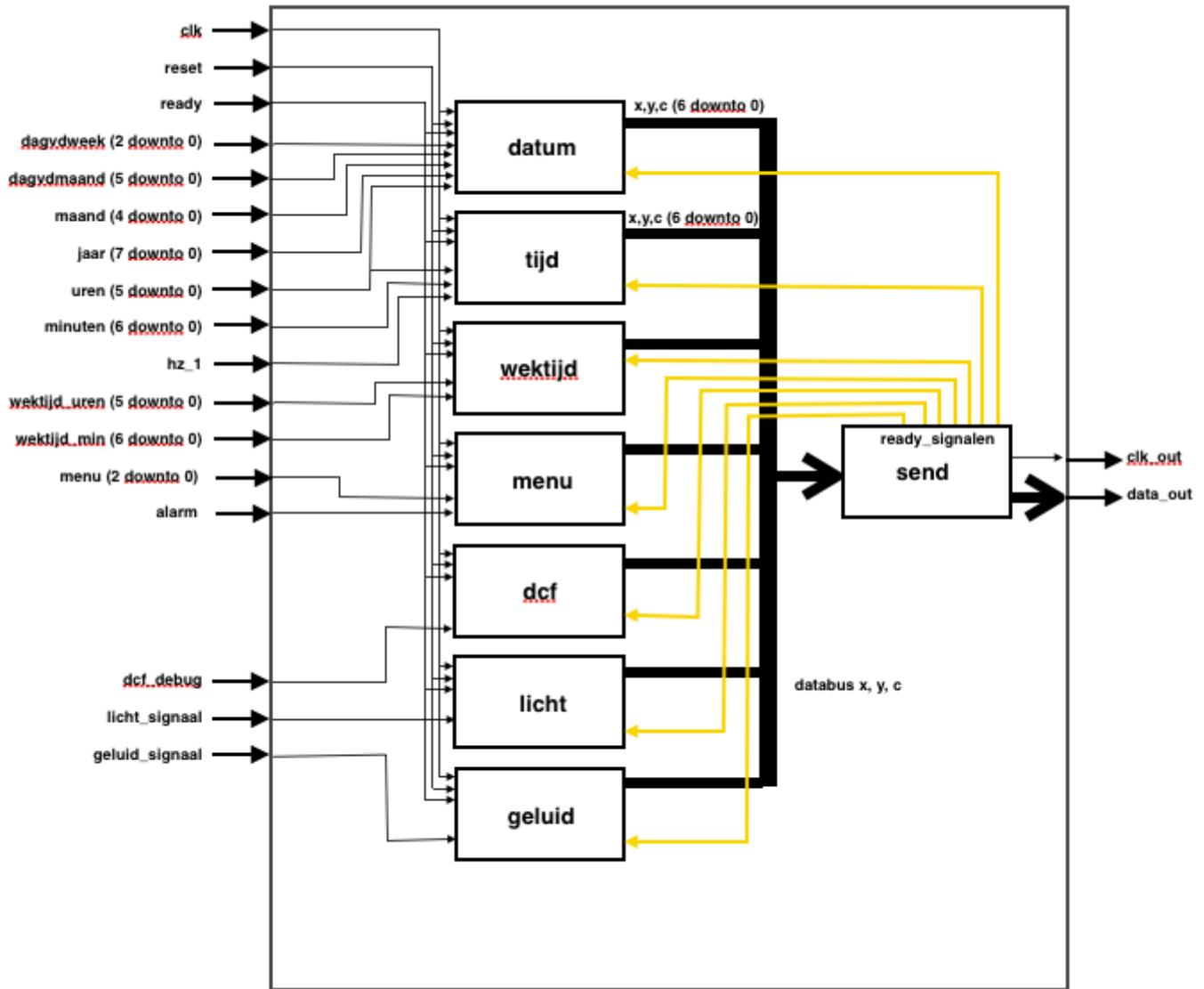
Op de LCD zal de huidige tijd, ingestelde wekkertijd, datum en ingeschakelde functies te zien zijn. Een LCD is daar handig voor omdat het veel ontwerp vrijheid biedt. Dat neemt ook mee dat het erg gecompliceerd kan worden. Het LCD dat zal worden gebruikt is van de fabrikant MIDAS, typenummer MC128064B6W-BNMLW. Het betreft een graphical LCD van 128 x 64 pixels met een register die geschreven kan worden. De bibliotheek met characters en de controller om het LCD te schrijven zal extern van deze chip plaatsvinden door middel van een atmega32-16pu. Deze keuze is gemaakt omdat voor de characters niet genoeg ruimte is op de chip. De LCD controller op de chip zal dus alleen de inkomende data moeten omzetten naar een positie waarnaar het geschreven moet worden en een bijbehorend character. De layout van het display is al vastgesteld, zodat de posities alvast bekend zijn. Zie voor de layout afbeelding 7.1.



Figuur 7.1: LCD layout

## 7.2. SPECIFICATIES

Naam	Type	Functie
clk	in std_logic	Klok
reset	in std_logic	Reset
ready	in std_logic	
uren	in std_logic_vector(5 downto 0)	data signaal met actuele uren afkomstig van DCF
minuten	in std_logic_vector(6 downto 0)	data signaal met actuele minuten afkomstig van DCF
dagvdweek	in std_logic_vector(2 downto 0)	data signaal met de actuele dag afkomstig van DCF
dagvdmaand	in std_logic_vector(5 downto 0)	data signaal met de actuele dag van de maand afkomstig van DCF
maand	in std_logic_vector(4 downto 0)	data signaal met de actuele maand afkomstig van DCF
jaar	in std_logic_vector(7 downto 0)	data signaal met het actuele jaar afkomstig van DCF
dcf_debug	in std_logic	signaal afkomstig van het dcf component en weergeeft of het DCF signaal ontvangen wordt of niet
menu	in std_logic_vector(2 downto 0)	data signaal die de actuele menu state weergeeft
alarm	in std_logic	buffer signaal dat weergeeft of alarmfunctie in of uitgeschakeld is
geluid_signaal	in std_logic	buffer signaal dat weergeeft of geluidsfunctie in of uitgeschakeld is
licht_signaal	in std_logic	buffer signaal dat weergeeft of lichtfunctie in of uitgeschakeld is
wektijd_uren	in std_logic_vector(5 downto 0)	data signaal met ingestelde wektijd uren
wektijd_min	in std_logic_vector(6 downto 0)	data signaal met ingestelde wektijd minuten
data_out	out std_logic_vector(6 downto 0)	data signaal dat de x,y,c informatie doorgeeft aan de microcontroller
clk_out	out std_logic	clock om microcontroller clock mee te synchroniseren



Figuur 7.2: Toplevel Entity

### 7.2.1. GEDRAG

De LCD controller zal na de reset alle informatie die hij binnen krijgt omzetten naar een karakter met bij behorende x en y positie en wegschrijven naar de microprocessor van de LCD. Daarna zal de controller alleen de data die veranderd op de ingangen omzetten en wegschrijven naar de microprocessor om tijd en onnodige acties te besparen.

Het verzenden van de x,y en c gaat door een data signaal van 7 bits samen met een clock\_out. Een neergaande klokflank geeft aan dat de data klaar staat om te verzenden zodat er op de opgaande klokflank kan worden gesampt. Zo zal eerst de x, daarna de y en als laatste de c worden verzonden. Het versturen van een karakter duurt dus 3 klokslagen van de clock\_out. een klokslag van de clock\_out is gelijk aan 2 klokslagen van de ingaande clk.

## 7.3. FUNCTIONALITEIT

De systemen links (datum, tijd, etc) zorgen per stuk voor het ontvangen van de inkomende informatie en het omzetten naar een x,y positie met een karakter. De x,y en de c zal op de uitgang van het component worden gezet. Het component send\_buffer is een MUX en zorgt voor het uitlezen van de x,y en c en zal door middel van de ready signalen aangeven welk signaal hij heeft uigelezen en naar de zender heeft verstuurd. Zodra de ready

laag wordt, weet het desbetreffende component dat de data is uitgelezen en zal daarna nieuwe data klaar zetten. Nadat de mux de data naar de zender heeft gebufferd, zal de zender de signalen een voor een door verzenden naar de microcontroller. Tegelijkertijd zal de zender een `clock_out` geven, zodra de `clock` laag staat de data klaar, zodat op de opgaande klokflank de data vanaf de chip kan worden uitgelezen.

## 7.4. SUBSYSTEMEN LCD

#### 7.4.1. VHDL CODE

Zie ?? voor de entity code.

Zie ?? voor de behavioural code.

Zie ?? voor de testbench die is gebruikt.

### 7.5. SIMULATIE

Zie ?? voor de simulatie van de behavioural code.

Zie ?? voor de simulatie van het gesynthetiseerde circuit.

### 7.6. TESTEN

Tijdens het testen van verschillende subcircuits zijn veel dingen fout gegaan. Vaak waren de circuits dan ook wel goed, maar was de testbench niet correct. Zo hebben we in het begin met een te hoge clock frequentie getest, maar ook met signalen die niet lang genoeg hoog bleven.

Het systeem is ook getest op een Altera FPGA bord. Eerst dachten we dat het niet werkte, maar achteraf bleek dat, wat best logisch is, het systeem zodanig snel werkt dat wij zelf het niet konden waarnemen. Door de klok te pulsen met een button konden we zien dat het systeem weldegelijk correct werkt.

### 7.7. RESULTATEN

De resultaten zijn goed. Het systeem heeft 2 errors in de switch level simulatie. Dat heeft te maken met de testbench waarin een situatie wordt gecreëerd die zich nooit zal voordoen. Dat maakt dus niks uit.

#### 7.7.1. CONCLUSIE EN DISCUSSIE

De controller werkt naar verwachting. Omdat vanwege tijdgebrek en gestelde prioriteiten de microcontroller nog niet af is voor het LCD kan nog niet worden getest of het systeem ook daarmee goed functioneert. Al verwachten wij, omdat de simulaties en de test op de FPGA positief zijn, dat het kan werken.

# 8

## RESULTS FOR TOTAL DESIGN

Er zijn nog geen subsystemen aan elkaar gekoppeld. Het testen met meer dan één blok is dus nog niet gebeurd.

# 9

## PLAN VOOR HET TESTEN VAN DE CHIP

### 9.1. LOGIC ANALYZER

Zodra de chip daadwerkelijk gemaakt is, kan die in kwartaal vier getest worden. Dit wordt gedaan met behulp van de logic analyzer LA-5580. Dit meetinstrument meet de uitgangsspanningen (logische één of nul) als functie van de tijd. De voordelen van deze analyzer ten opzichte van een oscilloscoop is dat de analyzer een groot aantal uitgangsspanningen kan meten en tegelijkertijd ingangssignalen de chip in kan sturen.

Voor het testen van de chip met een logic analyzer zijn een paar referentiedocumenten nodig die geproduceerd worden tijdens het ontwerpen van de chip. De ingangssignalen die in de referentiedocumenten staan worden als ingang naar de chip gestuurd en de uitgangssignalen van de chip en die van de documenten worden met elkaar vergeleken.

### 9.2. TESTSIGNALEN

Er hadden nog testsignalen op de lege pinnen gezet kunnen worden. Op die manier zouden ook signalen getest kunnen worden die van het ene subblok naar het andere subblok gaan. Mocht er een subblok niet functioneren, kan dat getest worden met behulp van de testsignalen en eventueel kunnen die testsignalen nog gebruikt worden om op een andere manier een bruikbaar en visueel uitgangssignaal te krijgen.

Het zou dan om de volgende signalen gaan

- Het DCF\_debug signaal
- Het Hz\_1 signaal
- En de vector menu\_state.

Door tijd en ruimtegebrek was dit alleen niet meer mogelijk.

# 10

## VOORTGANG VAN HET PROJECT

### 10.1. INLEIDING

Bij dit project zijn er vaak weinig resultaten, totdat het bijna afgelopen is. Dit is een van de redenen dat voor een wake-up light gekozen is. Een wekker zelf is relatief makkelijk te maken. Er zijn echter ook een hele hoop extra features die in een wekker geïmplementeerd kunnen worden. Op deze manier is dus een werkend resultaat relatief snel geproduceerd en kunnen daarna naar gelang extra toepassingen toegevoegd worden. Dit is goed voor het moreel in de groep, aangezien een werkend product al heel snel gerealiseerd is. Hierdoor is er ook meer aansporing om meer toepassingen te implementeren, omdat er al een werkend geheel is. In het ergste geval is er geen extra feature. Daarnaast, als uiteindelijk bleek dat de planning te krap was, kunnen er features geschrapt worden, en is er nog steeds een werkend product. Onder andere dit maakt een wake-up light zeer aantrekkelijk om te maken.

### 10.2. WERKVERDELING

Zodra duidelijk was wat gemaakt ging worden, werd een werkplan opgesteld. Dit was nodig zodat het duidelijk was wat er gedaan moest worden. Vervolgens zijn de taken zo snel mogelijk verdeeld door de wake-up light in blokken te verdelen. Van deze blokken werden ook eerst de specificaties bepaald, zodat er geen communicatieproblemen zouden ontstaan tussen de blokken. Uiteindelijk zijn er 4 hoofdblokken ontstaan, wat goed uitkwam, aangezien dit betekende dat er weer 4 tweetallen nodig waren, en de groep bestaat uit 8 mensen.

Deze blokken werden vervolgens door de tweetallen apart gemaakt en waar de specificaties niet duidelijk genoeg waren, of onhandig gedefinieerd, werden deze aangepast.

### 10.3. SAMENWERKING BINNEN DE GROEP

De samenwerking binnen de groep was goed. Als afspraken niet duidelijk waren werd snel contact gezocht, hiertoe was onder andere een Whatsapp-groepsgesprek aangemaakt. Van de vier onderdelen van het wake-up light waren er drie ruim op tijd af. De laatste, het LCD-scherm, liet iets langer op zich wachten. Toen de uiteindelijke deadline dichterbij kwam, werd er echter wel uitstekend samengewerkt om het toch nog af te krijgen.

### 10.4. AFSPRAKEN EN DEADLINES BINNEN DE GROEP

Afspraken binnen de groep verliepen meestal soepel. Zo gebeurde het dat op de dag van een presentatie bleek dat een van de leden ziek was. Andere leden sprongen in en zo kwam de presentatie toch nog tot een goed einde.

# 11

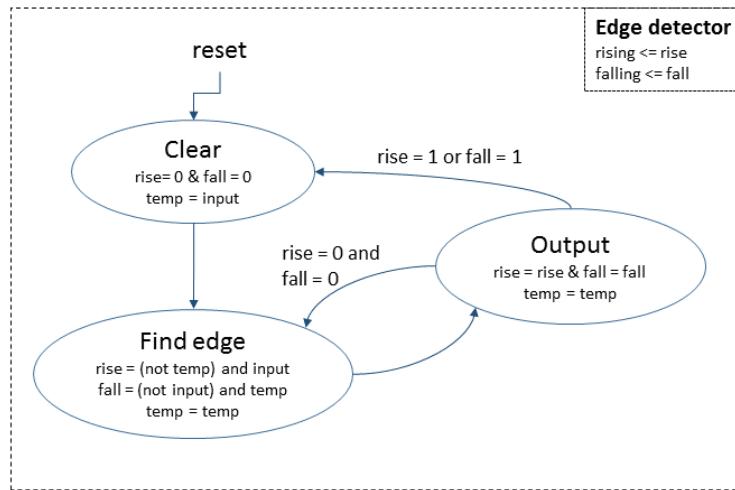
## CONCLUSIE

De wekker werkt nu naar behoren. Alle onderdelen voldoen aan de gestelde eisen. De interne klok wordt gesynchroniseert als een goed DCF-signalen wordt ontvangen en kan op zichzelf door tellen mocht het signaal wegvalLEN. De data wordt ook overgeverst en doorgegeven mits de parity check het signaal goedkeurd. De controller beschikt over een werkend menu en kan bepalen welke informatie op het LCD-scherm weergegeven wordt. Dit kan via de vier knoppen: menu, set, up en down. Daarnaast is de wekker in te stellen via het menu en kan het geluid apart aan- of uitgezet worden. Het licht begint 15 minuten van te voren langzaam meer licht te geven en op de ingestelde wekkertijd gaat het geluid af. De LCD\_controller geeft overzichtelijk alle informatie via een microcontroller weer op een LCD scherm. Alle onderdelen zijn succesvol gesimuleerd in modelsim®en via een switch-level simulatie. Nadat deze simulaties overeen kwamen zijn alle onderdelen op een FPGA getest. Dit bleek ook succesvol te zijn. Vervolgens werd met succes deze handelingen voor een top-entity herhaald. Hoewel de wekker theoretisch op nu op de chip kan staan, zal het moeten blijken of de chip het in praktijk ook doet. Naast deze test zijn er nog meer mogelijkheden om te onderzoeken. Zo zouden er meer functionaliteiten en opties op de wekker kunnen. Zoals bijvoorbeeld het instellen van de duur dat het licht aangaat of de temperatuur weergeven. Bij zo'n eventuele uitbreiding zal misschien overwogen moeten worden een grotere chip te nemen zodat er meer oppast.

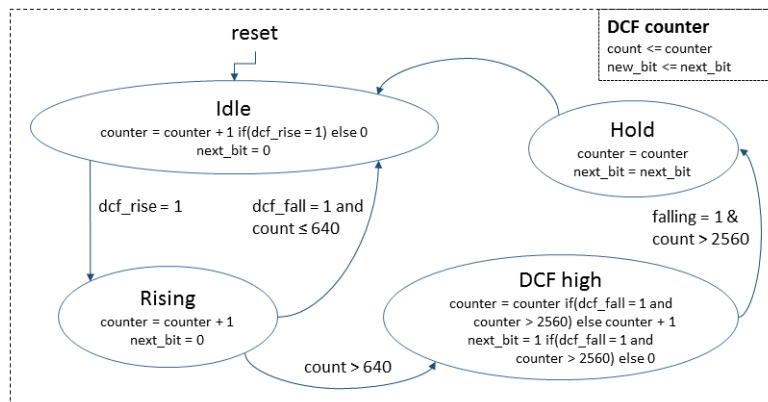
# A

## FSM DIAGRAMMEN (DCF77 BLOK)

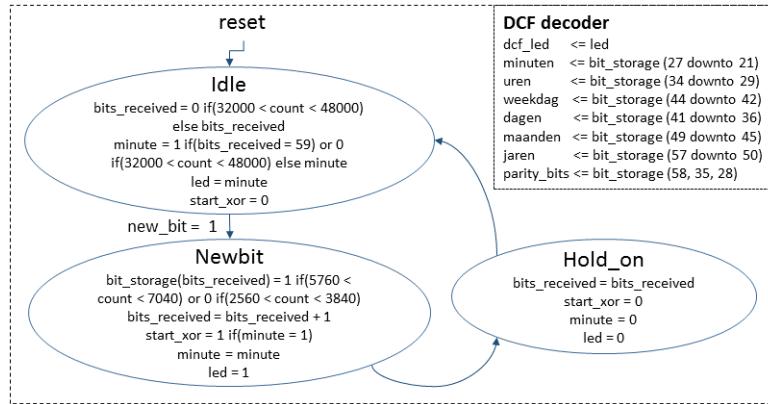
### SUBBLOKKEN VAN SYNCTIME



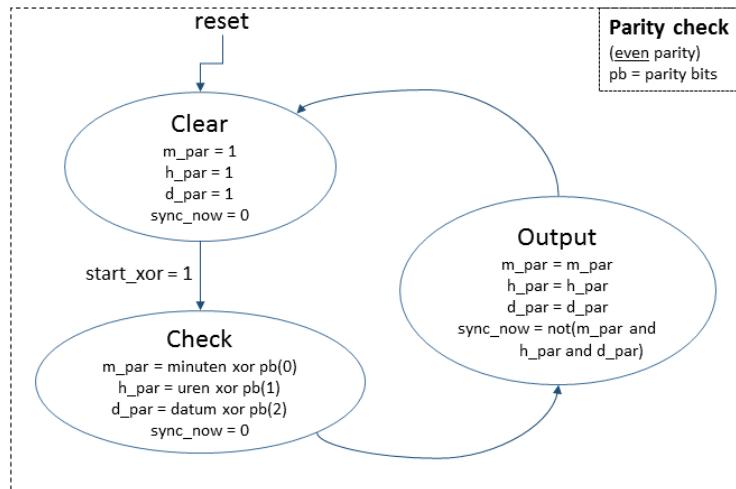
Figuur A.1: FSM diagram van het subblok edge detector



Figuur A.2: FSM diagram van het subblok DCF counter

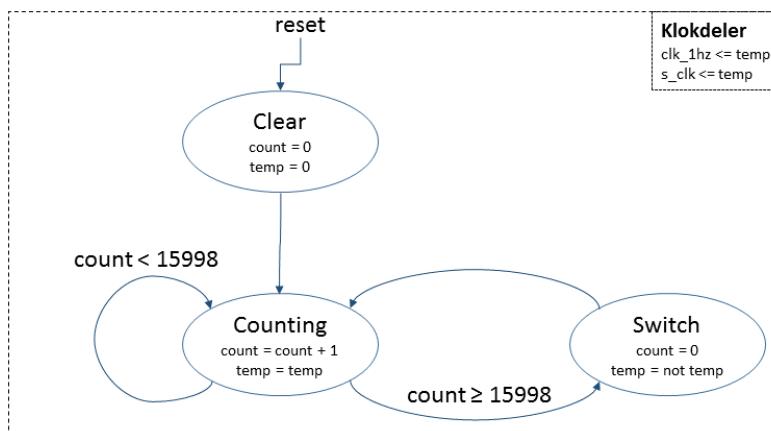


Figuur A.3: FSM diagram van het subblok DCF decoder



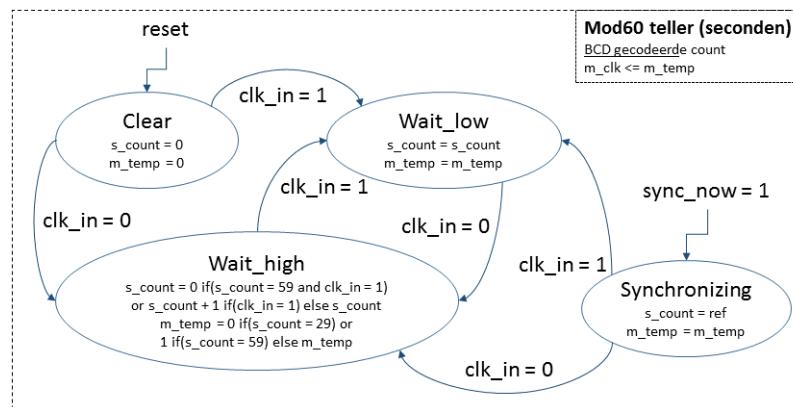
Figuur A.4: FSM diagram van het subblok parity check

## KLOKDELER

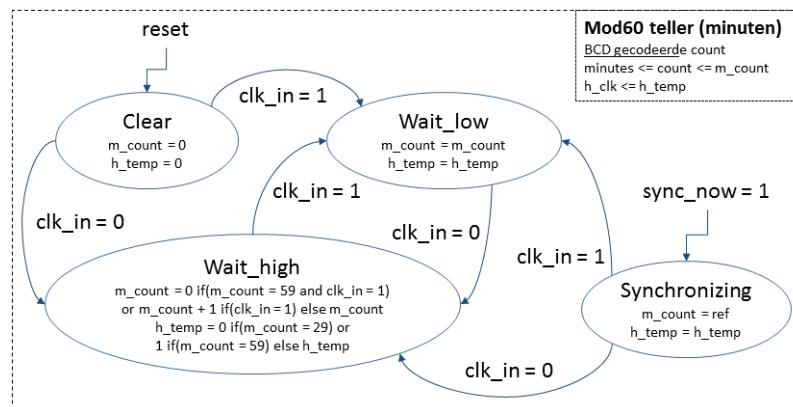


Figuur A.5: FSM diagram van het subblok klokdeeler

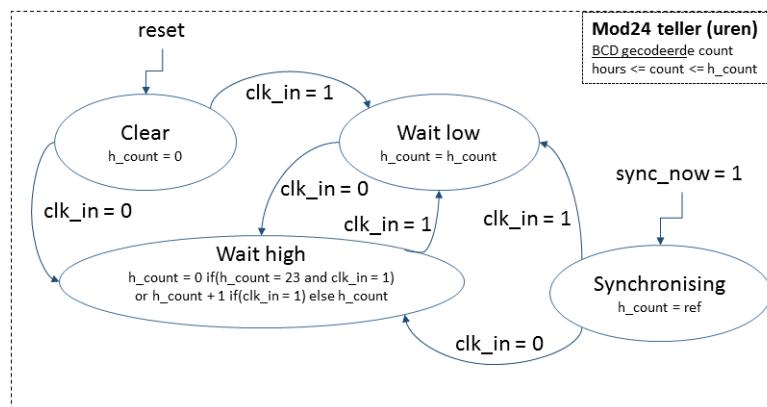
## SUBBLOKKEN VAN DE KLOK



Figuur A.6: FSM diagram van het subblok seconden teller



Figuur A.7: FSM diagram van het subblok minuten teller



Figuur A.8: FSM diagram van het subblok uren teller

# B

## VHDL CODE

### B.1. VHDL BESCHRIJVING VAN HET DCF77 BLOK

#### B.1.1. EDGE DETECTOR

```
1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity edge_detector is
7     port(clk      :in  std_logic;      -- 32 kHz systeemklok
8           reset   :in  std_logic;      -- Systeemreset
9           input    :in  std_logic;      -- digitaal DCF77 signaal
10          rising  :out  std_logic;      -- DCF77 rising edge
11          falling:out  std_logic);    -- DCF77 falling edge
12 end edge_detector;
13
14
15 -- Alex Oudsen, 4325494
16 -- De edge detector genereert uit het dcf77 signaal
17 -- aparte pulsen voor de rising en falling edges
18
19 library ieee;
20 use ieee.std_logic_1164.all;
21
22 architecture behaviour of edge_detector is
23
24     type edged is (clear, find_edge, output);
25     signal state, new_state: edged;
26
27     signal temp, new_temp: std_logic;
28     signal rise, new_rise: std_logic;
29     signal fall, new_fall: std_logic;
30
31     signal r_wait, new_r_wait: std_logic;
32     signal f_wait, new_f_wait: std_logic;
33
34 begin
35     rising <= rise; -- rising en falling zijn de daadwerkelijke uitgangen
36     falling <= fall;
37
38     process(clk)
39     begin
40         if(clk'event and clk = '1') then
41             if(reset = '1') then      -- Systeemreset
42                 rise <= '0';
43                 fall <= '0';
44                 temp <= '0';
45
46                 r_wait <= '0';
47                 f_wait <= '0';
48                 state <= clear;
49             end if;
50         end if;
51     end process;
52
53     process(state)
54     begin
55         case state is
56             when clear => begin
57                 state <= find_edge;
58             end;
59             when find_edge => begin
60                 if(rise = '1') then
61                     state <= output;
62                 else
63                     state <= find_edge;
64                 end if;
65             end;
66             when output => begin
67                 if(fall = '1') then
68                     state <= clear;
69                 else
70                     state <= find_edge;
71                 end if;
72             end;
73         end case;
74     end process;
75
76     process(state)
77     begin
78         if(state = output) then
79             if(rise = '1') then
80                 r_wait <= '1';
81             end if;
82         end if;
83
84         if(state = clear) then
85             if(fall = '1') then
86                 f_wait <= '1';
87             end if;
88         end if;
89     end process;
90
91     process(rise, fall)
92     begin
93         if(rise = '1') then
94             if(fall = '1') then
95                 new_rise <= '1';
96                 new_fall <= '1';
97             end if;
98         end if;
99     end process;
100
101    process(fall)
102    begin
103        if(fall = '1') then
104            if(rise = '1') then
105                new_fall <= '1';
106                new_rise <= '1';
107            end if;
108        end if;
109    end process;
110
111    process(rise)
112    begin
113        if(rise = '1') then
114            if(fall = '1') then
115                new_rise <= '1';
116                new_fall <= '1';
117            end if;
118        end if;
119    end process;
120
121    process(new_rise, new_fall)
122    begin
123        if(new_rise = '1') then
124            if(new_fall = '1') then
125                new_temp <= '1';
126            end if;
127        end if;
128    end process;
129
130    process(new_temp)
131    begin
132        if(new_temp = '1') then
133            if(state = find_edge) then
134                new_state <= output;
135            else
136                new_state <= find_edge;
137            end if;
138        end if;
139    end process;
140
141    process(new_state)
142    begin
143        if(new_state = output) then
144            if(rise = '1') then
145                new_rise <= '1';
146            end if;
147        end if;
148
149        if(new_state = clear) then
150            if(fall = '1') then
151                new_fall <= '1';
152            end if;
153        end if;
154    end process;
155
156    process(new_rise, new_fall)
157    begin
158        if(new_rise = '1') then
159            if(new_fall = '1') then
160                clk <= '1';
161            end if;
162        end if;
163    end process;
164
165    process(clk)
166    begin
167        if(clk = '1') then
168            if(new_rise = '1') then
169                rising <= '1';
170            end if;
171
172            if(new_fall = '1') then
173                falling <= '1';
174            end if;
175        end if;
176    end process;
177
178    process(rising, falling)
179    begin
180        if(rising = '1') then
181            if(falling = '1') then
182                new_rise <= '1';
183                new_fall <= '1';
184            end if;
185        end if;
186    end process;
187
188    process(new_rise, new_fall)
189    begin
190        if(new_rise = '1') then
191            if(new_fall = '1') then
192                new_temp <= '1';
193            end if;
194        end if;
195    end process;
196
197    process(new_temp)
198    begin
199        if(new_temp = '1') then
200            if(state = find_edge) then
201                new_state <= output;
202            else
203                new_state <= find_edge;
204            end if;
205        end if;
206    end process;
207
208    process(new_state)
209    begin
210        if(new_state = output) then
211            if(rising = '1') then
212                rising <= '1';
213            end if;
214
215            if(falling = '1') then
216                falling <= '1';
217            end if;
218        end if;
219    end process;
220
221    process(rising, falling)
222    begin
223        if(rising = '1') then
224            if(falling = '1') then
225                new_rise <= '1';
226                new_fall <= '1';
227            end if;
228        end if;
229    end process;
230
231    process(new_rise, new_fall)
232    begin
233        if(new_rise = '1') then
234            if(new_fall = '1') then
235                new_temp <= '1';
236            end if;
237        end if;
238    end process;
239
240    process(new_temp)
241    begin
242        if(new_temp = '1') then
243            if(state = find_edge) then
244                new_state <= output;
245            else
246                new_state <= find_edge;
247            end if;
248        end if;
249    end process;
250
251    process(new_state)
252    begin
253        if(new_state = output) then
254            if(rising = '1') then
255                rising <= '1';
256            end if;
257
258            if(falling = '1') then
259                falling <= '1';
260            end if;
261        end if;
262    end process;
263
264    process(rising, falling)
265    begin
266        if(rising = '1') then
267            if(falling = '1') then
268                new_rise <= '1';
269                new_fall <= '1';
270            end if;
271        end if;
272    end process;
273
274    process(new_rise, new_fall)
275    begin
276        if(new_rise = '1') then
277            if(new_fall = '1') then
278                new_temp <= '1';
279            end if;
280        end if;
281    end process;
282
283    process(new_temp)
284    begin
285        if(new_temp = '1') then
286            if(state = find_edge) then
287                new_state <= output;
288            else
289                new_state <= find_edge;
290            end if;
291        end if;
292    end process;
293
294    process(new_state)
295    begin
296        if(new_state = output) then
297            if(rising = '1') then
298                rising <= '1';
299            end if;
300
301            if(falling = '1') then
302                falling <= '1';
303            end if;
304        end if;
305    end process;
306
307    process(rising, falling)
308    begin
309        if(rising = '1') then
310            if(falling = '1') then
311                new_rise <= '1';
312                new_fall <= '1';
313            end if;
314        end if;
315    end process;
316
317    process(new_rise, new_fall)
318    begin
319        if(new_rise = '1') then
320            if(new_fall = '1') then
321                new_temp <= '1';
322            end if;
323        end if;
324    end process;
325
326    process(new_temp)
327    begin
328        if(new_temp = '1') then
329            if(state = find_edge) then
330                new_state <= output;
331            else
332                new_state <= find_edge;
333            end if;
334        end if;
335    end process;
336
337    process(new_state)
338    begin
339        if(new_state = output) then
340            if(rising = '1') then
341                rising <= '1';
342            end if;
343
344            if(falling = '1') then
345                falling <= '1';
346            end if;
347        end if;
348    end process;
349
350    process(rising, falling)
351    begin
352        if(rising = '1') then
353            if(falling = '1') then
354                new_rise <= '1';
355                new_fall <= '1';
356            end if;
357        end if;
358    end process;
359
360    process(new_rise, new_fall)
361    begin
362        if(new_rise = '1') then
363            if(new_fall = '1') then
364                new_temp <= '1';
365            end if;
366        end if;
367    end process;
368
369    process(new_temp)
370    begin
371        if(new_temp = '1') then
372            if(state = find_edge) then
373                new_state <= output;
374            else
375                new_state <= find_edge;
376            end if;
377        end if;
378    end process;
379
380    process(new_state)
381    begin
382        if(new_state = output) then
383            if(rising = '1') then
384                rising <= '1';
385            end if;
386
387            if(falling = '1') then
388                falling <= '1';
389            end if;
390        end if;
391    end process;
392
393    process(rising, falling)
394    begin
395        if(rising = '1') then
396            if(falling = '1') then
397                new_rise <= '1';
398                new_fall <= '1';
399            end if;
400        end if;
401    end process;
402
403    process(new_rise, new_fall)
404    begin
405        if(new_rise = '1') then
406            if(new_fall = '1') then
407                new_temp <= '1';
408            end if;
409        end if;
410    end process;
411
412    process(new_temp)
413    begin
414        if(new_temp = '1') then
415            if(state = find_edge) then
416                new_state <= output;
417            else
418                new_state <= find_edge;
419            end if;
420        end if;
421    end process;
422
423    process(new_state)
424    begin
425        if(new_state = output) then
426            if(rising = '1') then
427                rising <= '1';
428            end if;
429
430            if(falling = '1') then
431                falling <= '1';
432            end if;
433        end if;
434    end process;
435
436    process(rising, falling)
437    begin
438        if(rising = '1') then
439            if(falling = '1') then
440                new_rise <= '1';
441                new_fall <= '1';
442            end if;
443        end if;
444    end process;
445
446    process(new_rise, new_fall)
447    begin
448        if(new_rise = '1') then
449            if(new_fall = '1') then
450                new_temp <= '1';
451            end if;
452        end if;
453    end process;
454
455    process(new_temp)
456    begin
457        if(new_temp = '1') then
458            if(state = find_edge) then
459                new_state <= output;
460            else
461                new_state <= find_edge;
462            end if;
463        end if;
464    end process;
465
466    process(new_state)
467    begin
468        if(new_state = output) then
469            if(rising = '1') then
470                rising <= '1';
471            end if;
472
473            if(falling = '1') then
474                falling <= '1';
475            end if;
476        end if;
477    end process;
478
479    process(rising, falling)
480    begin
481        if(rising = '1') then
482            if(falling = '1') then
483                new_rise <= '1';
484                new_fall <= '1';
485            end if;
486        end if;
487    end process;
488
489    process(new_rise, new_fall)
490    begin
491        if(new_rise = '1') then
492            if(new_fall = '1') then
493                new_temp <= '1';
494            end if;
495        end if;
496    end process;
497
498    process(new_temp)
499    begin
500        if(new_temp = '1') then
501            if(state = find_edge) then
502                new_state <= output;
503            else
504                new_state <= find_edge;
505            end if;
506        end if;
507    end process;
508
509    process(new_state)
510    begin
511        if(new_state = output) then
512            if(rising = '1') then
513                rising <= '1';
514            end if;
515
516            if(falling = '1') then
517                falling <= '1';
518            end if;
519        end if;
520    end process;
521
522    process(rising, falling)
523    begin
524        if(rising = '1') then
525            if(falling = '1') then
526                new_rise <= '1';
527                new_fall <= '1';
528            end if;
529        end if;
530    end process;
531
532    process(new_rise, new_fall)
533    begin
534        if(new_rise = '1') then
535            if(new_fall = '1') then
536                new_temp <= '1';
537            end if;
538        end if;
539    end process;
540
541    process(new_temp)
542    begin
543        if(new_temp = '1') then
544            if(state = find_edge) then
545                new_state <= output;
546            else
547                new_state <= find_edge;
548            end if;
549        end if;
550    end process;
551
552    process(new_state)
553    begin
554        if(new_state = output) then
555            if(rising = '1') then
556                rising <= '1';
557            end if;
558
559            if(falling = '1') then
560                falling <= '1';
561            end if;
562        end if;
563    end process;
564
565    process(rising, falling)
566    begin
567        if(rising = '1') then
568            if(falling = '1') then
569                new_rise <= '1';
570                new_fall <= '1';
571            end if;
572        end if;
573    end process;
574
575    process(new_rise, new_fall)
576    begin
577        if(new_rise = '1') then
578            if(new_fall = '1') then
579                new_temp <= '1';
580            end if;
581        end if;
582    end process;
583
584    process(new_temp)
585    begin
586        if(new_temp = '1') then
587            if(state = find_edge) then
588                new_state <= output;
589            else
590                new_state <= find_edge;
591            end if;
592        end if;
593    end process;
594
595    process(new_state)
596    begin
597        if(new_state = output) then
598            if(rising = '1') then
599                rising <= '1';
600            end if;
601
602            if(falling = '1') then
603                falling <= '1';
604            end if;
605        end if;
606    end process;
607
608    process(rising, falling)
609    begin
610        if(rising = '1') then
611            if(falling = '1') then
612                new_rise <= '1';
613                new_fall <= '1';
614            end if;
615        end if;
616    end process;
617
618    process(new_rise, new_fall)
619    begin
620        if(new_rise = '1') then
621            if(new_fall = '1') then
622                new_temp <= '1';
623            end if;
624        end if;
625    end process;
626
627    process(new_temp)
628    begin
629        if(new_temp = '1') then
630            if(state = find_edge) then
631                new_state <= output;
632            else
633                new_state <= find_edge;
634            end if;
635        end if;
636    end process;
637
638    process(new_state)
639    begin
640        if(new_state = output) then
641            if(rising = '1') then
642                rising <= '1';
643            end if;
644
645            if(falling = '1') then
646                falling <= '1';
647            end if;
648        end if;
649    end process;
650
651    process(rising, falling)
652    begin
653        if(rising = '1') then
654            if(falling = '1') then
655                new_rise <= '1';
656                new_fall <= '1';
657            end if;
658        end if;
659    end process;
660
661    process(new_rise, new_fall)
662    begin
663        if(new_rise = '1') then
664            if(new_fall = '1') then
665                new_temp <= '1';
666            end if;
667        end if;
668    end process;
669
670    process(new_temp)
671    begin
672        if(new_temp = '1') then
673            if(state = find_edge) then
674                new_state <= output;
675            else
676                new_state <= find_edge;
677            end if;
678        end if;
679    end process;
680
681    process(new_state)
682    begin
683        if(new_state = output) then
684            if(rising = '1') then
685                rising <= '1';
686            end if;
687
688            if(falling = '1') then
689                falling <= '1';
690            end if;
691        end if;
692    end process;
693
694    process(rising, falling)
695    begin
696        if(rising = '1') then
697            if(falling = '1') then
698                new_rise <= '1';
699                new_fall <= '1';
700            end if;
701        end if;
702    end process;
703
704    process(new_rise, new_fall)
705    begin
706        if(new_rise = '1') then
707            if(new_fall = '1') then
708                new_temp <= '1';
709            end if;
710        end if;
711    end process;
712
713    process(new_temp)
714    begin
715        if(new_temp = '1') then
716            if(state = find_edge) then
717                new_state <= output;
718            else
719                new_state <= find_edge;
720            end if;
721        end if;
722    end process;
723
724    process(new_state)
725    begin
726        if(new_state = output) then
727            if(rising = '1') then
728                rising <= '1';
729            end if;
730
731            if(falling = '1') then
732                falling <= '1';
733            end if;
734        end if;
735    end process;
736
737    process(rising, falling)
738    begin
739        if(rising = '1') then
740            if(falling = '1') then
741                new_rise <= '1';
742                new_fall <= '1';
743            end if;
744        end if;
745    end process;
746
747    process(new_rise, new_fall)
748    begin
749        if(new_rise = '1') then
750            if(new_fall = '1') then
751                new_temp <= '1';
752            end if;
753        end if;
754    end process;
755
756    process(new_temp)
757    begin
758        if(new_temp = '1') then
759            if(state = find_edge) then
760                new_state <= output;
761            else
762                new_state <= find_edge;
763            end if;
764        end if;
765    end process;
766
767    process(new_state)
768    begin
769        if(new_state = output) then
770            if(rising = '1') then
771                rising <= '1';
772            end if;
773
774            if(falling = '1') then
775                falling <= '1';
776            end if;
777        end if;
778    end process;
779
780    process(rising, falling)
781    begin
782        if(rising = '1') then
783            if(falling = '1') then
784                new_rise <= '1';
785                new_fall <= '1';
786            end if;
787        end if;
788    end process;
789
790    process(new_rise, new_fall)
791    begin
792        if(new_rise = '1') then
793            if(new_fall = '1') then
794                new_temp <= '1';
795            end if;
796        end if;
797    end process;
798
799    process(new_temp)
800    begin
801        if(new_temp = '1') then
802            if(state = find_edge) then
803                new_state <= output;
804            else
805                new_state <= find_edge;
806            end if;
807        end if;
808    end process;
809
810    process(new_state)
811    begin
812        if(new_state = output) then
813            if(rising = '1') then
814                rising <= '1';
815            end if;
816
817            if(falling = '1') then
818                falling <= '1';
819            end if;
820        end if;
821    end process;
822
823    process(rising, falling)
824    begin
825        if(rising = '1') then
826            if(falling = '1') then
827                new_rise <= '1';
828                new_fall <= '1';
829            end if;
830        end if;
831    end process;
832
833    process(new_rise, new_fall)
834    begin
835        if(new_rise = '1') then
836            if(new_fall = '1') then
837                new_temp <= '1';
838            end if;
839        end if;
840    end process;
841
842    process(new_temp)
843    begin
844        if(new_temp = '1') then
845            if(state = find_edge) then
846                new_state <= output;
847            else
848                new_state <= find_edge;
849            end if;
850        end if;
851    end process;
852
853    process(new_state)
854    begin
855        if(new_state = output) then
856            if(rising = '1') then
857                rising <= '1';
858            end if;
859
860            if(falling = '1') then
861                falling <= '1';
862            end if;
863        end if;
864    end process;
865
866    process(rising, falling)
867    begin
868        if(rising = '1') then
869            if(falling = '1') then
870                new_rise <= '1';
871                new_fall <= '1';
872            end if;
873        end if;
874    end process;
875
876    process(new_rise, new_fall)
877    begin
878        if(new_rise = '1') then
879            if(new_fall = '1') then
880                new_temp <= '1';
881            end if;
882        end if;
883    end process;
884
885    process(new_temp)
886    begin
887        if(new_temp = '1') then
888            if(state = find_edge) then
889                new_state <= output;
890            else
891                new_state <= find_edge;
892            end if;
893        end if;
894    end process;
895
896    process(new_state)
897    begin
898        if(new_state = output) then
899            if(rising = '1') then
900                rising <= '1';
901            end if;
902
903            if(falling = '1') then
904                falling <= '1';
905            end if;
906        end if;
907    end process;
908
909    process(rising, falling)
910    begin
911        if(rising = '1') then
912            if(falling = '1') then
913                new_rise <= '1';
914                new_fall <= '1';
915            end if;
916        end if;
917    end process;
918
919    process(new_rise, new_fall)
920    begin
921        if(new_rise = '1') then
922            if(new_fall = '1') then
923                new_temp <= '1';
924            end if;
925        end if;
926    end process;
927
928    process(new_temp)
929    begin
930        if(new_temp = '1') then
931            if(state = find_edge) then
932                new_state <= output;
933            else
934                new_state <= find_edge;
935            end if;
936        end if;
937    end process;
938
939    process(new_state)
940    begin
941        if(new_state = output) then
942            if(rising = '1') then
943                rising <= '1';
944            end if;
945
946            if(falling = '1') then
947                falling <= '1';
948            end if;
949        end if;
950    end process;
951
952    process(rising, falling)
953    begin
954        if(rising = '1') then
955            if(falling = '1') then
956                new_rise <= '1';
957                new_fall <= '1';
958            end if;
959        end if;
960    end process;
961
962    process(new_rise, new_fall)
963    begin
964        if(new_rise = '1') then
965            if(new_fall = '1') then
966                new_temp <= '1';
967            end if;
968        end if;
969    end process;
970
971    process(new_temp)
972    begin
973        if(new_temp = '1') then
974            if(state = find_edge) then
975                new_state <= output;
976            else
977                new_state <= find_edge;
978            end if;
979        end if;
980    end process;
981
982    process(new_state)
983    begin
984        if(new_state = output) then
985            if(rising = '1') then
986                rising <= '1';
987            end if;
988
989            if(falling = '1') then
990                falling <= '1';
991            end if;
992        end if;
993    end process;
994
995    process(rising, falling)
996    begin
997        if(rising = '1') then
998            if(falling = '1') then
999                new_rise <= '1';
1000                new_fall <= '1';
1001            end if;
1002        end if;
1003    end process;
1004
1005    process(new_rise, new_fall)
1006    begin
1007        if(new_rise = '1') then
1008            if(new_fall = '1') then
1009                new_temp <= '1';
1010            end if;
1011        end if;
1012    end process;
1013
1014    process(new_temp)
1015    begin
1016        if(new_temp = '1') then
1017            if(state = find_edge) then
1018                new_state <= output;
1019            else
1020                new_state <= find_edge;
1021            end if;
1022        end if;
1023    end process;
1024
1025    process(new_state)
1026    begin
1027        if(new_state = output) then
1028            if(rising = '1') then
1029                rising <= '1';
1030            end if;
1031
1032            if(falling = '1') then
1033                falling <= '1';
1034            end if;
1035        end if;
1036    end process;
1037
1038    process(rising, falling)
1039    begin
1040        if(rising = '1') then
1041            if(falling = '1') then
1042                new_rise <= '1';
1043                new_fall <= '1';
1044            end if;
1045        end if;
1046    end process;
1047
1048    process(new_rise, new_fall)
1049    begin
1050        if(new_rise = '1') then
1051            if(new_fall = '1') then
1052                new_temp <= '1';
1053            end if;
1054        end if;
1055    end process;
1056
1057    process(new_temp)
1058    begin
1059        if(new_temp = '1') then
1060            if(state = find_edge) then
1061                new_state <= output;
1062            else
1063                new_state <= find_edge;
1064            end if;
1065        end if;
1066    end process;
1067
1068    process(new_state)
1069    begin
1070        if(new_state = output) then
1071            if(rising = '1') then
1072                rising <= '1';
1073            end if;
1074
1075            if(falling = '1') then
1076                falling <= '1';
1077            end if;
1078        end if;
1079    end process;
1080
1081    process(rising, falling)
1082    begin
1083        if(rising = '1') then
1084            if(falling = '1') then
1085                new_rise <= '1';
1086                new_fall <= '1';
1087            end if;
1088        end if;
1089    end process;
1090
1091    process(new_rise, new_fall)
1092    begin
1093        if(new_rise = '1') then
1094            if(new_fall = '1') then
1095                new_temp <= '1';
1096            end if;
1097        end if;
1098    end process;
1099
1100    process(new_temp)
1101    begin
1102        if(new_temp = '1') then
1103            if(state = find_edge) then
1104                new_state <= output;
1105            else
1106                new_state <= find_edge;
1107            end if;
1108        end if;
1109    end process;
1110
1111    process(new_state)
1112    begin
1113        if(new_state = output) then
1114            if(rising = '1') then
1115                rising <= '1';
1116            end if;
1117
1118            if(falling = '1') then
1119                falling <= '1';
1120            end if;
1121        end if;
1122    end process;
1123
1124    process(rising, falling)
1125    begin
1126        if(rising = '1') then
1127            if(falling = '1') then
1128                new_rise <= '1';
1129                new_fall <= '1';
1130            end if;
1131        end if;
1132    end process;
1133
1134    process(new_rise, new_fall)
1135    begin
1136        if(new_rise = '1') then
1137            if(new_fall = '1') then
1138                new_temp <= '1';
1139            end if;
1140        end if;
1141    end process;
1142
1143    process(new_temp)
1144    begin
1145        if(new_temp = '1') then
1146            if(state = find_edge) then
1147                new_state <= output;
1148            else
1149                new_state <= find_edge;
1150            end if;
1151        end if;
1152    end process;
1153
1154    process(new_state)
1155    begin
1156        if(new_state = output) then
1157            if(rising = '1') then
1158                rising <= '1';
1159            end if;
1160
1161            if(falling = '1') then
1162                falling <= '1';
1163            end if;
1164        end if;
1165    end process;
1166
1167    process(rising, falling)
1168    begin
1169        if(rising = '1') then
1170            if(falling = '1') then
1171                new_rise <= '1';
1172                new_fall <= '1';
1173            end if;
1174        end if;
1175    end process;
1176
1177    process(new_rise, new_fall)
1178    begin
1179        if(new_rise = '1') then
1180            if(new_fall = '1') then
1181                new_temp <= '1';
1182            end if;
1183        end if;
1184    end process;
1185
1186    process(new_temp)
1187    begin
1188        if(new_temp = '1') then
1189            if(state = find_edge) then
1190                new_state <= output;
1191            else
1192                new_state <= find_edge;
1193            end if;
1194        end if;
1195    end process;
1196
1
```

```

34      else
35          rise <= new_rise;
36          fall <= new_fall;
37          temp <= new_temp;
38
39          r_wait <= new_r_wait;
40          f_wait <= new_f_wait;
41          state <= new_state;
42      end if;
43  end if;
44 end process;
45
46 process(state, input, temp, rise, fall, r_wait, f_wait) is
47
48 begin
49     case state is
50         when clear =>          -- Reset state
51             new_rise <= '0';
52             new_fall <= '0';
53             new_temp <= input;
54
55             new_r_wait <= '0';
56             new_f_wait <= '0';
57             new_state <= find_edge;
58         when find_edge =>    -- Maakt gebruik van vertragingstijd van de not
59             new_rise <= (not temp) and input;
60             new_fall <= (not input) and temp;
61             new_temp <= temp;
62
63             new_r_wait <= '0';
64             new_f_wait <= '0';
65             new_state <= output;
66         when output =>        -- Hier wordt gezorgd voor een bruikbare uitgangspuls
67             new_rise <= rise;
68             new_fall <= fall;
69             new_temp <= temp;
70
71         if(rise = '1' or fall = '1') then
72             if(rise <= '1' and f_wait = '0') then
73                 new_r_wait <= '0';
74                 new_f_wait <= '1';
75                 new_state <= clear;
76             elsif(fall <= '1' and r_wait = '0') then
77                 new_r_wait <= '1';
78                 new_f_wait <= '0';
79                 new_state <= clear;
80             else
81                 new_r_wait <= '0';
82                 new_f_wait <= '0';
83                 new_state <= find_edge;
84             end if;
85         else
86             new_r_wait <= '0';
87             new_f_wait <= '0';
88             new_state <= find_edge;
89         end if;
90     when others =>        -- Zou nooit mogen voorkomen
91         new_rise <= '0';
92         new_fall <= '0';
93         new_temp <= '0';
94
95         new_r_wait <= '0';
96         new_f_wait <= '0';
97         new_state <= clear;
98     end case;
99 end process;
100 end behaviour;

```

### B.1.2. DCF COUNTER

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity dcf_counter is
7     port(clk      :in  std_logic;    -- 32 kHz systeemklok
8           reset     :in  std_logic;    -- Systeemreset
9           dcf_rise:in  std_logic;    -- DCF77 signaal - rising edge
10          dcf_fall:in  std_logic;    -- DCF77 signaal - falling edge
11          count    :out  std_logic_vector(15 downto 0);  -- Tellerwaarde
12          new_bit  :out  std_logic); -- Een nieuwe bit is geteld
13 end dcf_counter;


---


1 -- Alex Oudsen, 4325494
2 -- De dcf_counter is verantwoordelijk voor het detecteren van de pulsen
3 -- van het dcf77 signaal waarmee de informatiebits zijn gecodeerd
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7 use ieee.std_logic_unsigned.all;
8
9 architecture behaviour of dcf_counter is
10
11    type count_state is (idle, rising, dcf_high, hold);
12    signal state, new_state: count_state;
13
14    signal next_bit, new_next_bit: std_logic;
15    signal counter, new_counter: std_logic_vector(15 downto 0);
16
17 begin
18     count <= counter;
19     new_bit <= next_bit;
20
21     process(clk) is
22     begin
23         if(clk'event and clk = '1') then
24             if(reset = '1') then          -- Systeemreset
25                 counter <= (others => '0');
26                 next_bit <= '0';
27                 state <= idle;
28             else
29                 counter <= new_counter;
30                 next_bit <= new_next_bit;
31                 state <= new_state;
32             end if;
33         end if;
34     end process;
35
36     process(state, dcf_rise, dcf_fall, counter, next_bit) is
37     begin
38         case state is
39             when idle =>
40                 -- Controleer op een rising edge van het dcf77 signaal
41                 if(dcf_rise = '1') then
42                     new_counter <= (others => '0');
43                     new_next_bit <= '0';
44                     new_state <= rising;
45                 else
46                     new_counter <= counter + 1;
47                     new_next_bit <= '0';
48                     new_state <= idle;
49                 end if;
50             when rising =>
51                 -- Hier wordt bepaald of een rising edge hoort bij
52                 -- een daadwerkelijke puls of een spike/glitch
53                 new_counter <= counter + 1;
54                 new_next_bit <= '0';
55                 if(counter > 640) then

```

```

56         new_state <= dcf_high;
57     elsif(dcf_fall = '1') then
58         new_state <= idle;
59     else
60         new_state <= rising;
61     end if;
62     when dcf_high =>
63         -- Wacht op een falling edge van het dcf77 signaal
64         -- Negeer bovendien falling edges t.g.v. spikes/glitches
65     if(dcf_fall = '1' and counter > 2560) then
66         new_counter <= counter;
67         new_next_bit <= '1';
68         new_state <= hold;
69     else
70         new_counter <= counter + 1;
71         new_next_bit <= '0';
72         new_state <= dcf_high;
73     end if;
74     when hold =>
75         new_counter <= counter;
76         new_next_bit <= next_bit;
77         new_state <= idle;
78     when others => -- Zou nooit mogen voorkomen
79         new_counter <= (others => '0');
80         new_next_bit <= '0';
81         new_state <= idle;
82     end case;
83 end process;
84 end behaviour;

```

### B.1.3. DCF DECODER

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity dcf_decoder is
7     port(clk           :in std_logic; -- 32 kHz systeemklok
8          reset        :in std_logic; -- Systeemreset
9          count        :in std_logic_vector(15 downto 0); -- Tellerwaarde
10         new_bit      :in std_logic; -- Een nieuwe bit is geteld
11         dcf_led      :out std_logic; -- Debug signaal voor signaalontvangst
12         start_xor   :out std_logic; -- Enable signaal voor parity_check
13         minuten     :out std_logic_vector(6 downto 0); -- Minuten in BCD
14         uren        :out std_logic_vector(5 downto 0); -- Uren in BCD
15         weekdag    :out std_logic_vector(2 downto 0); -- Dagen (001 is maandag, enz.)
16         dagen       :out std_logic_vector(5 downto 0); -- Dagen (de cijfers) in BCD
17         maanden    :out std_logic_vector(4 downto 0); -- Nummer van de maand in BCD
18         jaren       :out std_logic_vector(7 downto 0); -- Laatste 2 cijfers van het jaartal; in
19             BCD
20         parity_bits :out std_logic_vector(2 downto 0)); -- De 3 parity bits uit het DCF77
21         signaal
22     end dcf_decoder;


---


1 -- Joran Out, 4331958 & Alex Oudsen, 4325494
2 -- De dcf_decoder filtert met behulp van de informatie die
3 -- de dcf_counter over het digitale dcf ingangssignaal levert
4 -- de gewenste bits waarmee datum en tijd zijn gecodeerd
5 -- Bovendien worden de drie bijbehorende parity_bits
6 -- ook meegegeven aan het volgende onderdeel (parity_check)
7 -- Het uitgangssignaal dcf_led geeft aan of het dcf signaal
8 -- goed wordt ontvangen door uit te gaan op het moment dat
9 -- er bits uit de reeks van 59 die het dcf77 signaal bevat, ontbreken
10
11 library ieee;
12 use ieee.std_logic_1164.all;
13 use ieee.std_logic_unsigned.all;
14 use ieee.numeric_std.all;
15
16 architecture behaviour of dcf_decoder is

```

```

17
18 type dcf_state is (idle, newbit, hold_on);
19
20 signal led, new_led: std_logic;
21 signal state, new_state: dcf_state;
22 signal minute, new_minute: std_logic;
23 signal bit_storage, new_bit_storage: std_logic_vector(58 downto 0);
24 signal bits_received, new_bits_received: std_logic_vector(5 downto 0);
25
26 begin
27     dcf_led      <= led;
28     minuten      <= bit_storage(27 downto 21);
29     uren         <= bit_storage(34 downto 29);
30     weekdag     <= bit_storage(44 downto 42);
31     dagen        <= bit_storage(41 downto 36);
32     maanden     <= bit_storage(49 downto 45);
33     jaren        <= bit_storage(57 downto 50);
34     parity_bits(2)    <= bit_storage(58);
35     parity_bits(1)    <= bit_storage(35);
36     parity_bits(0)    <= bit_storage(28);
37
38 process(clk, reset) is
39 begin
40     if(reset = '1') then                      -- Systeemreset
41         bits_received <= (others => '0');
42         bit_storage <= (others => '0');
43         minute <= '0';
44         led <= '0';
45         state <= idle;
46     elsif(clk'event and clk = '1') then          -- Opgaande klokflank v.d.
47         systeemklok
48         bits_received <= new_bits_received;
49         bit_storage <= new_bit_storage;
50         minute <= new_minute;
51         led <= new_led;
52         state <= new_state;
53     end if;
54 end process;
55
56 process(state, count, new_bit, minute, bits_received, bit_storage, led) is
57
58     variable location: natural range 0 to 59;
59
60 begin
61     location := to_integer(unsigned(bits_received));
62     new_bit_storage <= bit_storage;
63
64     case state is
65         when idle =>
66             if(new_bit = '1') then
67                 new_bits_received <= bits_received;
68                 new_minute <= minute;
69                 new_led <= led;
70                 start_xor <= '0';
71                 new_state <= newbit;
72                 -- Controleer of er een nieuwe minuut gaat beginnen
73             elsif(count > 32000 and count < 48000) then
74                 -- Initialiseer voor een nieuwe minuut
75                 new_bits_received <= (others => '0');
76
77                 -- Controle of er niet toevallig een seconde is gemist
78             if(bits_received = 59) then
79                 new_minute <= '1';
80                 new_led <= '1';
81             else
82                 new_minute <= minute;
83                 new_led <= '0';
84             end if;
85
86             start_xor <= '0';
87             new_state <= idle;

```

```

87         else
88             new_bits_received <= bits_received;
89             new_minute <= minute;
90             new_led <= led;
91             start_xor <= '0';
92             new_state <= idle;
93         end if;
94     when newbit =>
95         -- Ga na of de ontvangen bit een 1 of een 0 is
96         if(count <= 7040 and count >= 5760) then
97             new_bits_received <= bits_received + 1;
98             new_bit_storage(location) <= '1';
99         elsif(count <= 3840 and count >= 2560) then
100            new_bits_received <= bits_received + 1;
101            new_bit_storage(location) <= '0';
102        else      -- Zou nooit mogen voorkomen
103            new_bits_received <= (others => '0');
104        end if;
105
106        new_minute <= minute;
107        new_led <= '1';
108
109        -- Geef een seintje (start_xor) als een nieuwe minuut is begonnen
110        if(minute = '1') then
111            start_xor <= '1';
112        else
113            start_xor <= '0';
114        end if;
115
116        new_state <= hold_on;
117    when hold_on =>
118        new_bits_received <= bits_received;
119        new_minute <= '0';
120        new_led <= '1';
121        start_xor <= '0';
122        new_state <= idle;
123    when others =>          -- Zou nooit mogen voorkomen
124        new_bits_received <= bits_received;
125        new_minute <= '0';
126        new_led <= '0';
127        start_xor <= '0';
128        new_state <= idle;
129    end case;
130 end process;
131 end behaviour;

```

#### B.1.4. PARITY CHECK

```

1 -- Joran Out, 4331958 & Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity parity_check is
7     port(clk           :in  std_logic;      -- 32 kHz systeemklok
8         reset         :in  std_logic;      -- Systeemreset
9         start_xor     :in  std_logic;      -- Enable voor parity_check
10        minuten       :in  std_logic_vector(6 downto 0);
11        uren          :in  std_logic_vector(5 downto 0);
12        weekdag       :in  std_logic_vector(2 downto 0);
13        dagen          :in  std_logic_vector(5 downto 0);
14        maanden       :in  std_logic_vector(4 downto 0);
15        jaren          :in  std_logic_vector(7 downto 0);
16        parity_bits   :in  std_logic_vector(2 downto 0);
17        sync_now       :out std_logic);    -- Ready signaal van parity_check
18 end parity_check;

```

---

```

1 -- Joran Out, 4331958 & Alex Oudsen, 4325494
2 -- De parity check controleert of de bits die uit de
3 -- dcf decoder komen 'kloppen' volgens de parity bits,
4 -- welke eveneens uit de decoder komen. Indien een

```

```

5  -- parity bit 1 is, zou er in de bijbehorende bits
6  -- een even aantal enen moeten voorkomen
7
8  library ieee;
9  use ieee.std_logic_1164.all;
10 use ieee.numeric_std.all;
11
12 architecture behaviour of parity_check is
13
14  type checks is (clear, check, output);
15  signal state, new_state: checks;
16
17  signal m_par, new_m_par: std_logic; -- parity minuten (correct = 0)
18  signal h_par, new_h_par: std_logic; -- parity uren (correct = 0)
19  signal d_par, new_d_par: std_logic; -- parity datum (correct = 0)
20
21 begin
22  process(clk) is
23  begin
24    if(clk'event and clk = '1') then
25      if(reset = '1') then          -- Systeemreset
26        m_par <= '1';
27        h_par <= '1';
28        d_par <= '1';
29        state <= clear;
30      else
31        m_par <= new_m_par;
32        h_par <= new_h_par;
33        d_par <= new_d_par;
34        state <= new_state;
35      end if;
36    end if;
37  end process;
38
39  process(state, start_xor, m_par, h_par, d_par, minuten, uren, weekdag, dagen, maanden,
40          jaren, parity_bits) is
41  begin
42    case state is
43      when clear =>           -- Dit is de reset state
44        new_m_par <= '1';
45        new_h_par <= '1';
46        new_d_par <= '1';
47
48        sync_now <= '0';
49        if(start_xor = '1') then
50          new_state <= check; -- Start een parity check
51        else
52          new_state <= clear;
53        end if;
54      when check =>           -- Voer de parity check uit
55        new_m_par <= minuten(6) xor minuten(5) xor minuten(4) xor
56                    minuten(3) xor minuten(2) xor minuten(1) xor
57                    minuten(0) xor parity_bits(0);
58        new_h_par <= uren(5) xor uren(4) xor uren(3) xor
59                    uren(2) xor uren(1) xor uren(0) xor
60                    parity_bits(1);
61        new_d_par <= weekdag(2) xor weekdag(1) xor weekdag(0) xor
62                    dagen(5) xor dagen(4) xor dagen(3) xor
63                    dagen(2) xor dagen(1) xor dagen(0) xor
64                    maanden(4) xor maanden(3) xor maanden(2) xor
65                    maanden(1) xor maanden(0) xor jaren(7) xor
66                    jaren(6) xor jaren(5) xor jaren(4) xor
67                    jaren(3) xor jaren(2) xor jaren(1) xor
68                    jaren(0) xor parity_bits(2);
69
70        sync_now <= '0';
71        new_state <= output;
72      when output =>
73        new_m_par <= m_par;
74        new_h_par <= h_par;
75        new_d_par <= d_par;

```

```

75
76      if(m_par = '0' and h_par = '0' and d_par = '0') then
77          sync_now <= '1';      -- Parity is correct
78      else
79          sync_now <= '0';      -- Parity is niet correct
80      end if;
81      new_state <= clear;
82  when others =>           -- Zou nooit mogen voorkomen
83      new_m_par <= m_par;
84      new_h_par <= h_par;
85      new_d_par <= d_par;
86
87      sync_now <= '0';
88      new_state <= clear;
89  end case;
90 end process;
91 end behaviour;

```

### B.1.5. SYNCTIME

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity synctime is
7     port (clk:          in std_logic;
8           reset:        in std_logic;
9           dcf_in:       in std_logic;
10          dcf_led:      out std_logic;
11          ready:        out std_logic;
12          minuten:     out std_logic_vector(6 downto 0);
13          uren:         out std_logic_vector(5 downto 0);
14          weekdag:    out std_logic_vector(2 downto 0);
15          dagen:       out std_logic_vector(5 downto 0);
16          maanden:    out std_logic_vector(4 downto 0);
17          jaren:        out std_logic_vector(7 downto 0));
18 end synctime;


---


1 -- Alex Oudsen, 4325494
2 -- Dit onderdeel is verantwoordelijk voor het genereren
3 -- van synchronisatiemomenten uit het te ontvangen
4 -- digitale dcf77 signaal
5
6 -- Er wordt gebruik gemaakt van de volgende subblokken:
7 -- edge_detector, dcf_counter, dcf_decoder en parity_check
8
9 library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 architecture structure of synctime is
14     component edge_detector is
15         port(clk:          in std_logic;
16               reset:        in std_logic;
17               input:         in std_logic;
18               rising:       out std_logic;
19               falling:      out std_logic);
20     end component edge_detector;
21
22     component dcf_counter is
23         port(clk:          :in  std_logic;
24               reset:        :in  std_logic;
25               dcf_rise:     :in  std_logic;
26               dcf_fall:     :in  std_logic;
27               count:        :out std_logic_vector(15 downto 0);
28               new_bit:      :out  std_logic);
29     end component dcf_counter;
30
31     component dcf_decoder is
32         port(clk:          :in  std_logic;

```

```

33      reset      :in  std_logic;
34      count       :in  std_logic_vector(15 downto 0);
35      new_bit     :in  std_logic;
36      dcf_led     :out std_logic;
37      start_xor   :out std_logic;
38      minuten     :out std_logic_vector(6 downto 0);
39      uren        :out std_logic_vector(5 downto 0);
40      weekdag    :out std_logic_vector(2 downto 0);
41      dagen       :out std_logic_vector(5 downto 0);
42      maanden     :out std_logic_vector(4 downto 0);
43      jaren        :out std_logic_vector(7 downto 0);
44      parity_bits :out std_logic_vector(2 downto 0));
45  end component dcf_decoder;
46
47  component parity_check is
48    port (clk:           in  std_logic;
49          reset:         in  std_logic;
50          start_xor:    in  std_logic;
51          minuten:      in  std_logic_vector(6 downto 0);
52          uren:          in  std_logic_vector(5 downto 0);
53          weekdag:      in  std_logic_vector(2 downto 0);
54          dagen:         in  std_logic_vector(5 downto 0);
55          maanden:      in  std_logic_vector(4 downto 0);
56          jaren:         in  std_logic_vector(7 downto 0);
57          parity_bits:  in  std_logic_vector(2 downto 0);
58          sync_now:      out std_logic);
59  end component parity_check;
60
61  signal dcf_rise, dcf_fall, new_bit, start_xor: std_logic;
62  signal count: std_logic_vector(15 downto 0);
63  signal jaar: std_logic_vector(7 downto 0);
64  signal minuut: std_logic_vector(6 downto 0);
65  signal uur, dag: std_logic_vector(5 downto 0);
66  signal maand: std_logic_vector(4 downto 0);
67  signal weekday, par: std_logic_vector(2 downto 0);
68
69 begin
70   minuten <= minuut;
71   uren <= uur;
72   weekdag <= weekday;
73   dagen <= dag;
74   maanden <= maand;
75   jaren <= jaar;
76
77   edging: edge_detector  port map(clk, reset, dcf_in, dcf_rise, dcf_fall);
78   counts: dcf_counter   port map(clk, reset, dcf_rise, dcf_fall, count, new_bit);
79   decode: dcf_decoder   port map(clk, reset, count, new_bit, dcf_led, start_xor, minuut, uur,
80                                 weekday, dag, maand, jaar, par);
80   parity: parity_check  port map(clk, reset, start_xor, minuut, uur, weekday, dag, maand,
81                                   jaar, par, ready);
81
82 end structure;

```

### B.1.6. KLOKDELER

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity klokdeler is
7   port(clk   :in  std_logic;    -- 32 kHz systeemklok
8        reset :in  std_logic;    -- Systeemreset
9        clk_1hz:out std_logic); -- 1 Hz uitgangssignaal
10 end klokdeler;

```

---

```

1 -- Alex Oudsen, 4325494
2 -- Deze klokdeler deelt de frequentie van de ingang clk met een factor 32000
3 -- en wordt gebruikt om de systeemklok van 32 kHz te delen tot een 1 Hz signaal
4
5 library ieee;

```

```

6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_unsigned.all;
8
9  architecture behaviour of klokdeler is
10
11     type kd_state is (clear, counting, switch);           -- Declaratie van de
12     signal state, new_state: kd_state;
13
14     signal count, new_count: std_logic_vector(13 downto 0);    -- Voor het tellen tot 32000
15     signal temp, new_temp: std_logic;                      -- Interne versie van het 1 Hz
16     signal clk_1hz: std_logic;                            -- signaal
17
18 begin
19     clk_1hz <= temp;                                     -- Uitvoeren van het interne 1 Hz signaal
20
21     process(clk) is
22     begin
23         if(clk'event and clk = '1') then
24             if(reset = '1') then                         -- Systeemreset
25                 temp <= '0';
26                 count <= (others => '0');
27                 state <= clear;
28             else
29                 temp <= new_temp;
30                 count <= new_count;
31                 state <= new_state;
32             end if;
33         end if;
34     end process;
35
36     process(state, count, temp) is
37     begin
38         case state is
39             when clear =>          -- Dit is de reset state
40                 new_temp <= '0';
41                 new_count <= (others => '0');
42                 new_state <= counting;
43             when counting =>        -- Er wordt geteld
44                 new_temp <= temp;
45                 new_count <= count + 1;
46                 if(count < 15998) then
47                     new_state <= counting;
48                 else
49                     new_state <= switch;
50                 end if;
51             when switch =>          -- Inverteer de uitgangswaarde
52                 new_temp <= not temp;
53                 new_count <= (others => '0');
54                 new_state <= counting;
55             when others =>
56                 new_temp <= '1';
57                 new_count <= (others => '0');
58                 new_state <= clear;
59         end case;
60     end process;
61 end behaviour;

```

### B.1.7. MOD60 (SECONDEN) TELLER

```

1  -- Joran Out, 4331958 & Alex Oudsen, 4325494
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  entity mod60_clk_bcd is
7      port(clk      :in  std_logic;   -- 32 kHz systeemklok
8            clk_in   :in  std_logic;   -- Sturende klok(lokaal gegenereerd)
9            reset    :in  std_logic;   -- Systeemreset
10           sync_now:in  std_logic;   -- Enable signaal voor synchronisatie
11           ref      :in  std_logic_vector(6 downto 0);   -- Tijdsreferentie

```

```

12      m_clk    :out  std_logic); -- Sturende klok voor volgende teller
13  end mod60_clk_bcd;


---


1  -- Joran Out, 4331958 & Alex Oudsen, 4325494
2  -- Deze mod60 teller telt elke rising edge van clk_in 1
3  -- bij de huidige waarde van count op in bcd codering
4  -- De waarde van count wordt echter gelijk gemaakt aan ref,
5  -- wanneer het signaal sync_now hoog wordt
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9  use ieee.std_logic_unsigned.all;
10
11 architecture behaviour of mod60_clk_bcd is
12
13  type m60_state is (clear, wait_high, wait_low, synchronising);
14  signal state, new_state: m60_state;
15
16  signal s_count, new_s_count: std_logic_vector(6 downto 0);
17  signal m_temp, new_m_temp: std_logic;
18
19 begin
20     m_clk <= m_temp;
21
22     process(clk) is
23     begin
24         if(clk'event and clk = '1') then
25             if(reset = '1') then
26                 s_count <= (others => '0'); -- Systeemreset
27                 m_temp <= '0';
28                 state <= clear;
29             elsif(sync_now = '1') then
30                 s_count <= ref;
31                 m_temp <= new_m_temp;
32                 state <= synchronising;
33             else
34                 s_count <= new_s_count;
35                 m_temp <= new_m_temp;
36                 state <= new_state;
37             end if;
38         end if;
39     end process;
40
41     process(state, clk_in, ref, s_count, m_temp) is
42     begin
43         case state is
44             when clear =>          -- Reset state
45                 new_s_count <= (others => '0');
46                 new_m_temp <= '0';
47                 if(clk_in = '1') then
48                     new_state <= wait_low;
49                 else
50                     new_state <= wait_high;
51                 end if;
52             when wait_high =>        -- Er wordt geteld op de sturende klok
53                 if(clk_in = '1') then
54                     if(s_count = "1011001") then
55                         new_s_count <= (others => '0');
56                     elsif(s_count(3 downto 0) = "1001") then
57                         new_s_count <= s_count + 7;
58                     else
59                         new_s_count <= s_count + 1;
60                     end if;
61                     if(s_count = "0101001") then
62                         new_m_temp <= '0';
63                     elsif(s_count = "1011001") then
64                         new_m_temp <= '1';
65                     else
66                         new_m_temp <= m_temp;
67                     end if;
68                     new_state <= wait_low;

```

```

69         else
70             new_s_count <= s_count;
71             new_m_temp <= m_temp;
72             new_state <= wait_high;
73         end if;
74     when wait_low =>
75         if(clk_in = '0') then
76             new_s_count <= s_count;
77             new_m_temp <= m_temp;
78             new_state <= wait_high;
79         else
80             new_s_count <= s_count;
81             new_m_temp <= m_temp;
82             new_state <= wait_low;
83         end if;
84     when synchronising =>
85         new_s_count <= ref;
86         new_m_temp <= m_temp;
87         if(clk_in = '1') then
88             new_state <= wait_low;
89         else
90             new_state <= wait_high;
91         end if;
92     when others =>          -- Zou nooit mogen voorkomen
93         new_s_count <= (others => '0');
94         new_m_temp <= '0';
95         new_state <= clear;
96     end case;
97 end process;
98 end behaviour;

```

### B.1.8. MOD60 (MINUTEN) TELLER

```

1  -- Joran Out, 4331958 & Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity mod60_tel_bcd is
7     port(clk      :in  std_logic;    -- 32 kHz systeemklok
8           clk_in   :in  std_logic;    -- Sturende klok(lokaal gegenereerd)
9           reset    :in  std_logic;    -- Systeemreset
10          sync_now:in  std_logic;    -- Enable signaal voor synchronisatie
11          ref      :in  std_logic_vector(6 downto 0);    -- Tijdsreferentie
12          count    :out std_logic_vector(6 downto 0);    -- Huidige tellerstand
13          h_clk    :out std_logic); -- Sturende klok voor volgende teller
14 end mod60_tel_bcd;


---


1  -- Joran Out, 4331958 & Alex Oudsen, 4325494
2  -- Deze mod60 teller telt elke rising edge van clk_in 1
3  -- bij de huidige waarde van count op in bcd codering
4  -- De waarde van count wordt echter gelijk gemaakt aan ref,
5  -- wanneer het signaal sync_now hoog wordt
6
7 library ieee;
8 use ieee.std_logic_1164.all;
9 use ieee.std_logic_unsigned.all;
10
11 architecture behaviour of mod60_tel_bcd is
12
13     type m60_state is (clear, wait_high, wait_low, synchronising);
14     signal state, new_state: m60_state;
15
16     signal m_count, new_m_count: std_logic_vector(6 downto 0);
17     signal h_temp, new_h_temp: std_logic;
18
19 begin
20     count <= m_count;
21     h_clk <= h_temp;
22
23     process(clk) is

```

```

24 begin
25   if(clk'event and clk = '1') then
26     if(reset = '1') then          -- Systeemreset
27       m_count <= (others => '0');
28       h_temp <= '0';
29       state <= clear;
30     elsif(sync_now = '1') then
31       m_count <= ref;
32       h_temp <= new_h_temp;
33       state <= synchronising;
34     else
35       m_count <= new_m_count;
36       h_temp <= new_h_temp;
37       state <= new_state;
38     end if;
39   end if;
40 end process;
41
42 process(state, clk_in, ref, m_count, h_temp) is
43 begin
44   case state is
45     when clear =>           -- Reset state
46       new_m_count <= (others => '0');
47       new_h_temp <= '0';
48       if(clk_in = '1') then
49         new_state <= wait_low;
50       else
51         new_state <= wait_high;
52       end if;
53     when wait_high =>        -- Er wordt geteld op de sturende klok
54       if(clk_in = '1') then
55         if(m_count = "1011001") then
56           new_m_count <= (others => '0');
57         elsif(m_count(3 downto 0) = "1001") then
58           new_m_count <= m_count + 7;
59         else
60           new_m_count <= m_count + 1;
61         end if;
62         if(m_count = "0101001") then
63           new_h_temp <= '0';
64         elsif(m_count = "1011001") then
65           new_h_temp <= '1';
66         else
67           new_h_temp <= h_temp;
68         end if;
69         new_state <= wait_low;
70       else
71         new_m_count <= m_count;
72         new_h_temp <= h_temp;
73         new_state <= wait_high;
74       end if;
75     when wait_low =>
76       if(clk_in = '0') then
77         new_m_count <= m_count;
78         new_h_temp <= h_temp;
79         new_state <= wait_high;
80       else
81         new_m_count <= m_count;
82         new_h_temp <= h_temp;
83         new_state <= wait_low;
84       end if;
85     when synchronising =>
86       new_m_count <= ref;
87       new_h_temp <= h_temp;
88       if(clk_in = '1') then
89         new_state <= wait_low;
90       else
91         new_state <= wait_high;
92       end if;
93     when others =>           -- Zou nooit mogen voorkomen
94       new_m_count <= (others => '0');

```

```

95          new_h_temp <= '0';
96          new_state <= clear;
97      end case;
98  end process;
99 end behaviour;
```

### B.1.9. MOD24 (UREN) TELLER

```

1 -- Joran Out, 4331958 & Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity mod24_tel_bcd is
7     port(clk      :in  std_logic; -- 32 kHz systeemklok
8           clk_in   :in  std_logic; -- 1/3600 Hz lokale klok
9           reset    :in  std_logic; -- Systeemreset
10          sync_now:in  std_logic; -- Enable signaal voor synchronisatie
11          ref      :in  std_logic_vector(5 downto 0);      -- Tijdsreferentie (uren)
12          count   :out  std_logic_vector(5 downto 0));    -- Teller met huidig aantal uren
13 end mod24_tel_bcd;

-- Joran Out, 4331958 & Alex Oudsen, 4325494
-- Deze mod24 teller telt elke rising edge van clk_in 1
-- bij de huidige waarde van count op in bcd codering
-- De waarde van count wordt echter gelijk gemaakt aan ref,
-- wanneer het signaal sync_now hoog wordt
6
7 library ieee;
8 use ieee.std_logic_1164.all;
9 use ieee.std_logic_unsigned.all;
10
11 architecture behaviour of mod24_tel_bcd is
12
13     type m24_state is (clear, wait_high, wait_low, synchronising);
14     signal state, new_state: m24_state;
15
16     signal h_count, new_h_count: std_logic_vector(5 downto 0);
17
18 begin
19     count <= h_count;
20
21     process(clk) is
22     begin
23         if(clk'event and clk = '1') then
24             if(reset = '1') then          -- Systeemreset
25                 h_count <= (others => '0');
26                 state <= clear;
27             elsif(sync_now = '1') then
28                 h_count <= ref;
29                 state <= synchronising;
30             else
31                 h_count <= new_h_count;
32                 state <= new_state;
33             end if;
34         end if;
35     end process;
36
37     process(state, clk_in, ref, h_count) is
38     begin
39         case state is
40             when clear =>          -- Reset state
41                 new_h_count <= (others => '0');
42                 if(clk_in = '1') then
43                     new_state <= wait_low;
44                 else
45                     new_state <= wait_high;
46                 end if;
47             when wait_high =>        -- Er wordt geteld op de sturende klok
48                 if(clk_in = '1') then
49                     if(h_count = "100011") then
```

```

50          new_h_count <= (others => '0');
51      elsif(h_count(3 downto 0) = "1001") then
52          new_h_count <= h_count + 7;
53      else
54          new_h_count <= h_count + 1;
55      end if;
56      new_state <= wait_low;
57  else
58      new_h_count <= h_count;
59      new_state <= wait_high;
60  end if;
61 when wait_low =>
62     if(clk_in = '0') then
63         new_h_count <= h_count;
64         new_state <= wait_high;
65     else
66         new_h_count <= h_count;
67         new_state <= wait_low;
68     end if;
69 when synchronising =>
70     new_h_count <= ref;
71     if(clk_in = '1') then
72         new_state <= wait_low;
73     else
74         new_state <= wait_high;
75     end if;
76 when others => -- Zou nooit mogen voorkomen
77     new_h_count <= (others => '0');
78     new_state <= clear;
79 end case;
80 end process;
81 end behaviour;

```

### B.1.10. AUTONOME SYNCHRONISEERBARE KLOK

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity ausy_klok_bcd is
7     port (clk:           in std_logic;    -- 32 kHz systeemklok
8           s_clk:        in std_logic;    -- 1 Hz klok uit klokdeler
9           reset:        in std_logic;    -- Systeemreset
10          sync_now:     in std_logic;    -- Enable signaal voor synchronisatie
11          min_ref:      in std_logic_vector(6 downto 0); -- Referentietijd (minuten bcd)
12          hr_ref:       in std_logic_vector(5 downto 0); -- Referentietijd (uren bcd)
13          minutes:      out std_logic_vector(6 downto 0); -- Huidige tijd (minuten bcd)
14          hours:        out std_logic_vector(5 downto 0)); -- Huidige tijd (uren bcd)
15 end ausy_klok_bcd;


---


1 -- Alex Oudsen, 4325494
2 -- Deze autonome, synchroniseerbare, bcd gecodeerde klok wordt gebruikt om
3 -- de huidige tijd bij te houden, ook als het dcf signaal niet beschikbaar is,
4 -- Wanneer het dcf signaal wel beschikbaar is,
5 -- wordt de klok gesynchroniseerd met dit signaal
6
7 -- Voor de bcd gecodeerde klok wordt gebruik gemaakt van de volgende subblokken:
8 -- mod24_tel_bcd, mod60_tel_bcd & mod60_clk_bcd
9
10 library ieee;
11 use ieee.std_logic_1164.all;
12
13 architecture structure of ausy_klok_bcd is
14     component mod60_clk_bcd is
15         port(clk:           in std_logic;
16               clk_in:        in std_logic;
17               reset:        in std_logic;
18               sync_now:     in std_logic;
19               ref:          in std_logic_vector(6 downto 0);
20               m_clk:         out std_logic);

```

```

21      end component mod60_clk_bcd;
22
23  component mod60_tel_bcd is
24    port (clk:      in std_logic;
25          clk_in:    in std_logic;
26          reset:     in std_logic;
27          sync_now:  in std_logic;
28          ref:       in std_logic_vector(6 downto 0);
29          count:     out std_logic_vector(6 downto 0);
30          h_clk:     out std_logic);
31  end component mod60_tel_bcd;
32
33  component mod24_tel_bcd is
34    port (clk:      in std_logic;
35          clk_in:    in std_logic;
36          reset:     in std_logic;
37          sync_now:  in std_logic;
38          ref:       in std_logic_vector(5 downto 0);
39          count:     out std_logic_vector(5 downto 0));
40  end component mod24_tel_bcd;
41
42  signal m_clk: std_logic;
43  signal h_clk: std_logic;
44  signal sec_ref: std_logic_vector(6 downto 0);
45
46 begin
47
48   sec_ref <= "0000000";
49
50   SEC: mod60_clk_bcd port map(clk, s_clk, reset, sync_now, sec_ref, m_clk);
51   MIN: mod60_tel_bcd port map(clk, m_clk, reset, sync_now, min_ref, minutes, h_clk);
52   HRS: mod24_tel_bcd port map(clk, h_clk, reset, sync_now, hr_ref, hours);
53
54 end structure;

```

### B.1.11. DCF77 (TOP-LEVEL)

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity dcf77_bcd is
7   port (clk:      in std_logic;
8         reset:    in std_logic;
9         dcf_in:   in std_logic;
10        dcf_led:  out std_logic;
11        clk_1hz:  out std_logic;
12        minutes: out std_logic_vector(6 downto 0);
13        hours:   out std_logic_vector(5 downto 0);
14        weekday: out std_logic_vector(2 downto 0);
15        day:     out std_logic_vector(5 downto 0);
16        month:   out std_logic_vector(4 downto 0);
17        year:    out std_logic_vector(7 downto 0);
18        date_ready: out std_logic);
19 end dcf77_bcd;

```

---

```

1 -- Alex Oudsen, 4325494
2 -- Dit is de top-level beschrijving van
3 -- de bcd versie van het dcf77 blok
4
5 -- Er wordt gebruik gebmaakt van de volgende subblokken:
6 -- synctime, klokdeler en ausy_klok_bcd
7
8 library ieee;
9 use ieee.std_logic_1164.all;
10 use ieee.numeric_std.all;
11
12 architecture structure of dcf77_bcd is
13   component synctime is
14     port(clk:      in std_logic;

```



## B.2. TESTBENCHES VOOR HET DCF77 BLOK

### B.2.1. TESTBENCH EDGE DETECTOR

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity edge_detect_tb is
7 end entity edge_detect_tb;

1 -- Alex Oudsen, 4325494
2 -- Dit is de testbench voor simulatie op schaal
3 -- De verwachte respons geeft een korte puls rising
4 -- direct na iedere rising edge van het ingangssignaal
5 -- en een korte puls falling direct na iedere falling
6 -- edge van het ingangssignaal
7 -- Advies simulatietijd: 50 ms.
8
9 library ieee;
10 use ieee.std_logic_1164.all;
11
12 architecture behaviour of edge_detect_tb is
13   component edge_detector is
14     port (clk:      in std_logic;
15           reset:    in std_logic;
16           input:    in std_logic;
17           rising:   out std_logic;
18           falling:  out std_logic);
19   end component edge_detector;
20
21 signal clk, reset, input, rising, falling: std_logic;
22
23 begin
24
25   clk <= '1' after 0 ns, -- Dit genereert een signaal van iets meer dan 3.2 MHz
26   '0' after 156 ns when clk /= '0' else '1' after 156 ns;
27   reset <= '1' after 0 ns, '0' after 500 ns;
28   input <= '0' after 0 ms, '1' after 4 ms, '0' after 5 ms,
29   '1' after 15 ms, '0' after 16 ms,
30   '1' after 25 ms, '0' after 26 ms,
31   '1' after 35 ms, '0' after 36 ms,
32   '1' after 45 ms, '0' after 46 ms;
33
34   detect: edge_detector port map(clk, reset, input, rising, falling);
35
36 end behaviour;

```

### B.2.2. TESTBENCH DCF COUNTER

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity dcf_counter_tb is
7 end dcf_counter_tb;

1 -- Alex Oudsen, 4325494
2 -- Dit is testbench van de simulatie op schaal
3 -- De verwachte respons bestaat uit 59 achtereenvolgende
4 -- new_bit pulsen, gevuld door een pauze en nog 1 next_bit puls
5 -- Bovendien dient de waarde count aan te uitgang te verschijnen,
6 -- omdat hier buiten dit blok nog verder mee gewerkt wordt
7 -- Advies simulatietijd: 75 ms.
8
9 library ieee;
10 use ieee.std_logic_1164.all;
11
12 architecture behaviour of dcf_counter_tb is

```

```

13  component dcf_counter is
14    port (clk:      in std_logic;
15          reset:     in std_logic;
16          dcf_rise:   in std_logic;
17          dcf_fall:   in std_logic;
18          count:     out std_logic_vector(15 downto 0);
19          new_bit:    out std_logic);
20  end component dcf_counter;
21
22  signal clk, reset, dcf_rise, dcf_fall, new_bit: std_logic;
23  signal count: std_logic_vector(15 downto 0);
24
25 begin
26   clk      <= '1' after 0 ns, -- De frequentie van dit signaal is iets meer dan 32 MHz
27           '0' after 15 ns when clk /= '0' else '1' after 15 ns;
28   reset    <= '1' after 0 ns, '0' after 50 ns;
29   dcf_rise <= '0' after 0 ns,
30           '1' after 400 us, '0' after 401 us,
31           '1' after 1500 us, '0' after 1501 us,
32           '1' after 2500 us, '0' after 2501 us,
33           '1' after 3500 us, '0' after 3501 us,
34           '1' after 4500 us, '0' after 4501 us,
35           '1' after 5500 us, '0' after 5501 us,
36           '1' after 6500 us, '0' after 6501 us,
37           '1' after 7500 us, '0' after 7501 us,
38           '1' after 8500 us, '0' after 8501 us,
39           '1' after 9500 us, '0' after 9501 us,
40           '1' after 10500 us, '0' after 10501 us,
41           '1' after 11500 us, '0' after 11501 us,
42           '1' after 12500 us, '0' after 12501 us,
43           '1' after 13500 us, '0' after 13501 us,
44           '1' after 14500 us, '0' after 14501 us,
45           '1' after 15500 us, '0' after 15501 us,
46           '1' after 16500 us, '0' after 16501 us,
47           '1' after 17500 us, '0' after 17501 us,
48           '1' after 18500 us, '0' after 18501 us,
49           '1' after 19500 us, '0' after 19501 us,
50           '1' after 20500 us, '0' after 20501 us,
51           '1' after 21500 us, '0' after 21501 us,
52           '1' after 22500 us, '0' after 22501 us,
53           '1' after 23500 us, '0' after 23501 us,
54           '1' after 24500 us, '0' after 24501 us,
55           '1' after 25500 us, '0' after 25501 us,
56           '1' after 26500 us, '0' after 26501 us,
57           '1' after 27500 us, '0' after 27501 us,
58           '1' after 28500 us, '0' after 28501 us,
59           '1' after 29500 us, '0' after 29501 us,
60           '1' after 30500 us, '0' after 30501 us,
61           '1' after 31500 us, '0' after 31501 us,
62           '1' after 32500 us, '0' after 32501 us,
63           '1' after 33500 us, '0' after 33501 us,
64           '1' after 34500 us, '0' after 34501 us,
65           '1' after 35500 us, '0' after 35501 us,
66           '1' after 36500 us, '0' after 36501 us,
67           '1' after 37500 us, '0' after 37501 us,
68           '1' after 38500 us, '0' after 38501 us,
69           '1' after 39500 us, '0' after 39501 us,
70           '1' after 40500 us, '0' after 40501 us,
71           '1' after 41500 us, '0' after 41501 us,
72           '1' after 42500 us, '0' after 42501 us,
73           '1' after 43500 us, '0' after 43501 us,
74           '1' after 44500 us, '0' after 44501 us,
75           '1' after 45500 us, '0' after 45501 us,
76           '1' after 46500 us, '0' after 46501 us,
77           '1' after 47500 us, '0' after 47501 us,
78           '1' after 48500 us, '0' after 48501 us,
79           '1' after 49500 us, '0' after 49501 us,
80           '1' after 50500 us, '0' after 50501 us,
81           '1' after 51500 us, '0' after 51501 us,
82           '1' after 52500 us, '0' after 52501 us,
83           '1' after 53500 us, '0' after 53501 us,
```

```

84          '1' after 54500 us, '0' after 54501 us,
85          '1' after 55500 us, '0' after 55501 us,
86          '1' after 56500 us, '0' after 56501 us,
87          '1' after 57500 us, '0' after 57501 us,
88          '1' after 58500 us, '0' after 58501 us,
89          '1' after 60500 us, '0' after 60501 us;
90      dcf_fall    <= '0' after 0 ns,
91          '1' after 500 us, '0' after 501 us,
92          '1' after 1600 us, '0' after 1601 us,
93          '1' after 2600 us, '0' after 2601 us,
94          '1' after 3600 us, '0' after 3601 us,
95          '1' after 4600 us, '0' after 4601 us,
96          '1' after 5600 us, '0' after 5601 us,
97          '1' after 6600 us, '0' after 6601 us,
98          '1' after 7600 us, '0' after 7601 us,
99          '1' after 8600 us, '0' after 8601 us,
100         '1' after 9600 us, '0' after 9601 us,
101         '1' after 10600 us, '0' after 10601 us,
102         '1' after 11600 us, '0' after 11601 us,
103         '1' after 12600 us, '0' after 12601 us,
104         '1' after 13600 us, '0' after 13601 us,
105         '1' after 14600 us, '0' after 14601 us,
106         '1' after 15600 us, '0' after 15601 us,
107         '1' after 16600 us, '0' after 16601 us,
108         '1' after 17600 us, '0' after 17601 us,
109         '1' after 18700 us, '0' after 18701 us,
110         '1' after 19600 us, '0' after 19601 us,
111         '1' after 20700 us, '0' after 20701 us,
112         '1' after 21600 us, '0' after 21601 us,
113         '1' after 22600 us, '0' after 22601 us,
114         '1' after 23600 us, '0' after 23601 us,
115         '1' after 24700 us, '0' after 24701 us,
116         '1' after 25600 us, '0' after 25601 us,
117         '1' after 26600 us, '0' after 26601 us,
118         '1' after 27700 us, '0' after 27701 us,
119         '1' after 28700 us, '0' after 28701 us,
120         '1' after 29700 us, '0' after 29701 us,
121         '1' after 30600 us, '0' after 30601 us,
122         '1' after 31600 us, '0' after 31601 us,
123         '1' after 32600 us, '0' after 32601 us,
124         '1' after 33700 us, '0' after 33701 us,
125         '1' after 34600 us, '0' after 34601 us,
126         '1' after 35700 us, '0' after 35701 us,
127         '1' after 36600 us, '0' after 36601 us,
128         '1' after 37600 us, '0' after 37601 us,
129         '1' after 38600 us, '0' after 38601 us,
130         '1' after 39700 us, '0' after 39701 us,
131         '1' after 40600 us, '0' after 40601 us,
132         '1' after 41600 us, '0' after 41601 us,
133         '1' after 42700 us, '0' after 42701 us,
134         '1' after 43600 us, '0' after 43601 us,
135         '1' after 44600 us, '0' after 44601 us,
136         '1' after 45600 us, '0' after 45601 us,
137         '1' after 46700 us, '0' after 46701 us,
138         '1' after 47600 us, '0' after 47601 us,
139         '1' after 48600 us, '0' after 48601 us,
140         '1' after 49700 us, '0' after 49701 us,
141         '1' after 50600 us, '0' after 50601 us,
142         '1' after 51600 us, '0' after 51601 us,
143         '1' after 52700 us, '0' after 52701 us,
144         '1' after 53600 us, '0' after 53601 us,
145         '1' after 54700 us, '0' after 54701 us,
146         '1' after 55600 us, '0' after 55601 us,
147         '1' after 56600 us, '0' after 56601 us,
148         '1' after 57600 us, '0' after 57601 us,
149         '1' after 58700 us, '0' after 58701 us,
150         '1' after 60600 us, '0' after 60601 us;
151
152     dcf_count: dcf_counter port map(clk, reset, dcf_rise, dcf_fall, count, new_bit);
153
154 end behaviour;

```

### B.2.3. TESTBENCH DCF DECODER

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity dcf_decoder_tb is
7 end dcf_decoder_tb;

1 -- Alex Oudsen, 4325494
2 -- Dit is testbench voor simulatie op schaal
3 -- De verwachte respons bestaat uit een led-signaal wat
4 -- afhankelijk is van de kwaliteit van het DCF77 signaal
5 -- Verder dient bij het begin van een nieuwe minuut start_xor
6 -- even hoog te worden; op dit moment moet het volgende aan de
7 -- uitgangen staan: maandag 8 december, 11:48
8 -- Advies simulatietijd: 70 ms.
9

10 library ieee;
11 use ieee.std_logic_1164.all;
12 use ieee.std_logic_unsigned.all;
13
14 architecture behaviour of dcf_decoder_tb is
15     component dcf_decoder is
16         port(clk :in std_logic;
17               reset :in std_logic;
18               count :in std_logic_vector(15 downto 0);
19               new_bit :in std_logic;
20               dcf_led :out std_logic;
21               start_xor :out std_logic;
22               minuten :out std_logic_vector(6 downto 0);
23               uren :out std_logic_vector(5 downto 0);
24               weekdag :out std_logic_vector(2 downto 0);
25               dagen :out std_logic_vector(5 downto 0);
26               maanden :out std_logic_vector(4 downto 0);
27               jaren :out std_logic_vector(7 downto 0);
28               parity_bits:out std_logic_vector(2 downto 0));
29     end component dcf_decoder;
30
31 signal clk, reset, new_bit, dcf_led, start_xor: std_logic;
32 signal count: std_logic_vector(15 downto 0);
33 signal jr: std_logic_vector(7 downto 0);
34 signal min: std_logic_vector(6 downto 0);
35 signal hr, dag: std_logic_vector(5 downto 0);
36 signal mnd: std_logic_vector(4 downto 0);
37 signal wk, pb: std_logic_vector(2 downto 0);
38
39 begin
40     clk      <= '0' after 0 ns,      -- Dit genereert een 32 MHz signaal (ongeveer)
41                  '1' after 313 ns when clk /= '1' else '0' after 313 ns;
42     reset    <= '1' after 0 ns, '0' after 50 ns;
43     count    <= "0000110010000000" after 0 ns,
44                  "0001100100000000" after 18 ms,
45                  "0000110010000000" after 19 ms,
46                  "0001100100000000" after 20 ms,
47                  "0000110010000000" after 21 ms,
48                  "0001100100000000" after 24 ms,
49                  "0000110010000000" after 25 ms,
50                  "0001100100000000" after 27 ms,
51                  "0000110010000000" after 30 ms,
52                  "0001100100000000" after 33 ms,
53                  "0000110010000000" after 34 ms,
54                  "0001100100000000" after 35 ms,
55                  "0000110010000000" after 36 ms,
56                  "0001100100000000" after 39 ms,
57                  "0000110010000000" after 40 ms,
58                  "0001100100000000" after 42 ms,
59                  "0000110010000000" after 43 ms,
60                  "0001100100000000" after 46 ms,
61                  "0000110010000000" after 47 ms,

```

```

62          "0001100100000000" after 49 ms,
63          "0000110010000000" after 50 ms,
64          "0001100100000000" after 52 ms,
65          "0000110010000000" after 53 ms,
66          "0001100100000000" after 54 ms,
67          "0000110010000000" after 55 ms,
68          "0001100100000000" after 58 ms,
69          "1001110001000000" after 59 ms,
70          "0000110010000000" after 60 ms,
71          "1001110001000000" after 61 ms;
72 new_bit <= '0' after 0 ns,
73          '1' after 500000 ns, '0' after 500800 ns,
74          '1' after 1600000 ns, '0' after 1600800 ns,
75          '1' after 2600000 ns, '0' after 2600800 ns,
76          '1' after 3600000 ns, '0' after 3600800 ns,
77          '1' after 4600000 ns, '0' after 4600800 ns,
78          '1' after 5600000 ns, '0' after 5600800 ns,
79          '1' after 6600000 ns, '0' after 6600800 ns,
80          '1' after 7600000 ns, '0' after 7600800 ns,
81          '1' after 8600000 ns, '0' after 8600800 ns,
82          '1' after 9600000 ns, '0' after 9600800 ns,
83          '1' after 10600000 ns, '0' after 10600800 ns,
84          '1' after 11600000 ns, '0' after 11600800 ns,
85          '1' after 12600000 ns, '0' after 12600800 ns,
86          '1' after 13600000 ns, '0' after 13600800 ns,
87          '1' after 14600000 ns, '0' after 14600800 ns,
88          '1' after 15600000 ns, '0' after 15600800 ns,
89          '1' after 16600000 ns, '0' after 16600800 ns,
90          '1' after 17600000 ns, '0' after 17600800 ns,
91          '1' after 18700000 ns, '0' after 18700800 ns,
92          '1' after 19600000 ns, '0' after 19600800 ns,
93          '1' after 20700000 ns, '0' after 20700800 ns,
94          '1' after 21600000 ns, '0' after 21600800 ns,
95          '1' after 22600000 ns, '0' after 22600800 ns,
96          '1' after 23600000 ns, '0' after 23600800 ns,
97          '1' after 24700000 ns, '0' after 24700800 ns,
98          '1' after 25600000 ns, '0' after 25600800 ns,
99          '1' after 26600000 ns, '0' after 26600800 ns,
100         '1' after 27700000 ns, '0' after 27700800 ns,
101         '1' after 28700000 ns, '0' after 28700800 ns,
102         '1' after 29700000 ns, '0' after 29700800 ns,
103         '1' after 30600000 ns, '0' after 30600800 ns,
104         '1' after 31600000 ns, '0' after 31600800 ns,
105         '1' after 32600000 ns, '0' after 32600800 ns,
106         '1' after 33700000 ns, '0' after 33700800 ns,
107         '1' after 34600000 ns, '0' after 34600800 ns,
108         '1' after 35700000 ns, '0' after 35700800 ns,
109         '1' after 36600000 ns, '0' after 36600800 ns,
110         '1' after 37600000 ns, '0' after 37600800 ns,
111         '1' after 38600000 ns, '0' after 38600800 ns,
112         '1' after 39700000 ns, '0' after 39700800 ns,
113         '1' after 40600000 ns, '0' after 40600800 ns,
114         '1' after 41600000 ns, '0' after 41600800 ns,
115         '1' after 42700000 ns, '0' after 42700800 ns,
116         '1' after 43600000 ns, '0' after 43600800 ns,
117         '1' after 44600000 ns, '0' after 44600800 ns,
118         '1' after 45600000 ns, '0' after 45600800 ns,
119         '1' after 46700000 ns, '0' after 46700800 ns,
120         '1' after 47600000 ns, '0' after 47600800 ns,
121         '1' after 48600000 ns, '0' after 48600800 ns,
122         '1' after 49700000 ns, '0' after 49700800 ns,
123         '1' after 50600000 ns, '0' after 50600800 ns,
124         '1' after 51600000 ns, '0' after 51600800 ns,
125         '1' after 52700000 ns, '0' after 52700800 ns,
126         '1' after 53600000 ns, '0' after 53600800 ns,
127         '1' after 54700000 ns, '0' after 54700800 ns,
128         '1' after 55600000 ns, '0' after 55600800 ns,
129         '1' after 56600000 ns, '0' after 56600800 ns,
130         '1' after 57600000 ns, '0' after 57600800 ns,
131         '1' after 58700000 ns, '0' after 58700800 ns,
132         '1' after 60600000 ns, '0' after 60600800 ns;

```

```

133
134     decode: dcf_decoder port map(clk, reset, count, new_bit, dcf_led, start_xor, min, hr, wk,
135         dag, mnd, jr, pb);
136 end behaviour;

```

#### B.2.4. TESTBENCH PARITY CHECK

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity parity_tb is
7 end parity_tb;


---


1 -- Joran Out, 4331958 & Alex Oudsen, 4325494
2 -- Er wordt gebruik gemaakt van de 'echte' klok
3 -- Verwachte respons is een sync_now puls, omdat
4 -- de parity van de gesimuleerde ingangen correct is
5 -- Advies simulatietijd: 150 microsec.
6
7 library ieee;
8 use ieee.std_logic_1164.all;
9
10 architecture behaviour of parity_tb is
11     component parity_check is
12         port (clk:      in std_logic;
13             reset:    in std_logic;
14             start_xor: in std_logic;
15             Minuten:   in std_logic_vector(6 downto 0);
16             Uren:      in std_logic_vector(5 downto 0);
17             Weekdag:   in std_logic_vector(2 downto 0);
18             Dagen:     in std_logic_vector(5 downto 0);
19             Maanden:   in std_logic_vector(4 downto 0);
20             Jaren:     in std_logic_vector(7 downto 0);
21             Parity_bits: in std_logic_vector(2 downto 0);
22             Sync_now:  out std_logic);
23     end component parity_check;
24
25     signal clk, reset, start_xor, sync_now: std_logic;
26     signal jaren: std_logic_vector(7 downto 0);
27     signal Minuten: std_logic_vector(6 downto 0);
28     signal Uren, Dagen: std_logic_vector(5 downto 0);
29     signal Maanden: std_logic_vector(4 downto 0);
30     signal Weekdag, Parity_bits: std_logic_vector(2 downto 0);
31
32 begin
33     clk      <= '1' after 0 ns,          -- Dit genereert een 32 kHz signaal
34                 '0' after 15625 ns when clk /= '0' else '1' after 15625 ns;
35     reset    <= '1' after 0 ns, '0' after 50000 ns;
36     start_xor <= '0' after 0 ns, '1' after 60000 ns, '0' after 65000 ns;
37     Minuten   <= "1001000" after 0 ns;
38     Uren      <= "010001" after 0 ns;
39     Weekdag   <= "001" after 0 ns;
40     Dagen     <= "001000" after 0 ns;
41     Maanden   <= "10010" after 0 ns;
42     Jaren     <= "00010100" after 0 ns;
43     Parity_bits <= "000" after 0 ns;
44
45     check: parity_check port map(clk, reset, start_xor, Minuten, Uren, Weekdag, Dagen,
46                               Maanden, Jaren, Parity_bits, sync_now);
47 end behaviour;

```

#### B.2.5. TESTBENCH SYNCTIME

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;

```

```

5
6 entity synctime_tb is
7 end synctime_tb;


---


1 -- Alex Oudsen, 4325494
2 -- Dit is de testbench voor simulatie op schaal
3 -- De verwachte respons geeft 1 synchronisatiemoment,
4 -- namelijk maandag 8 december 2014; 11:48
5 -- Advies simulatietijd: 65 ms.
6
7 library ieee;
8 use ieee.std_logic_1164.all;
9
10 architecture behaviour of synctime_tb is
11     component synctime is
12         port (clk:      in std_logic;
13             reset:    in std_logic;
14             dcf_in:   in std_logic;
15             dcf_led:  out std_logic;
16             ready:    out std_logic;
17             Minuten: out std_logic_vector(6 downto 0);
18             Uren:     out std_logic_vector(5 downto 0);
19             Weekdag: out std_logic_vector(2 downto 0);
20             Dagen:    out std_logic_vector(5 downto 0);
21             Maanden: out std_logic_vector(4 downto 0);
22             Jaren:    out std_logic_vector(7 downto 0));
23     end component synctime;
24
25     signal clk, reset, dcf_in, dcf_led, ready: std_logic;
26     signal Minuten: std_logic_vector(6 downto 0);
27     signal Uren, Dagen: std_logic_vector(5 downto 0);
28     signal Maanden: std_logic_vector(4 downto 0);
29     signal Weekdag: std_logic_vector(2 downto 0);
30     signal Jaren: std_logic_vector(7 downto 0);
31
32 begin
33
34     clk <= '1' after 0 ns,
35         '0' after 16 ns when clk /= '0' else '1' after 16 ns;
36     reset <= '1' after 0 ns, '0' after 50 us;
37     dcf_in <= '0' after 0 ns,
38         '1' after 1500 us, '0' after 1600 us, -- bit 59 & bit 0
39         '1' after 2500 us, '0' after 2600 us, -- bit 1
40         '1' after 3500 us, '0' after 3600 us, -- bit 2
41         '1' after 4500 us, '0' after 4600 us, -- bit 3
42         '1' after 5500 us, '0' after 5600 us, -- bit 4
43         '1' after 6500 us, '0' after 6600 us, -- bit 5
44         '1' after 7500 us, '0' after 7600 us, -- bit 6
45         '1' after 8500 us, '0' after 8600 us, -- bit 7
46         '1' after 9500 us, '0' after 9600 us, -- bit 8
47         '1' after 10500 us, '0' after 10600 us, -- bit 9
48         '1' after 11500 us, '0' after 11600 us, -- bit 10
49         '1' after 12500 us, '0' after 12600 us, -- bit 11
50         '1' after 13500 us, '0' after 13600 us, -- bit 12
51         '1' after 14500 us, '0' after 14600 us, -- bit 13
52         '1' after 15500 us, '0' after 15600 us, -- bit 14
53         '1' after 16500 us, '0' after 16600 us, -- bit 15
54         '1' after 17500 us, '0' after 17600 us, -- bit 16
55         '1' after 18500 us, '0' after 18600 us, -- bit 17
56         '1' after 19500 us, '0' after 19700 us, -- bit 18 1
57         '1' after 20500 us, '0' after 20600 us, -- bit 19
58         '1' after 21500 us, '0' after 21700 us, -- bit 20 1
59         '1' after 22500 us, '0' after 22600 us, -- bit 21
60         '1' after 23500 us, '0' after 23600 us, -- bit 22
61         '1' after 24500 us, '0' after 24600 us, -- bit 23
62         '1' after 25500 us, '0' after 25700 us, -- bit 24 1
63         '1' after 26500 us, '0' after 26600 us, -- bit 25
64         '1' after 27500 us, '0' after 27600 us, -- bit 26
65         '1' after 28500 us, '0' after 28700 us, -- bit 27 1
66         '1' after 29500 us, '0' after 29700 us, -- bit 28 1
67         '1' after 30500 us, '0' after 30700 us, -- bit 29 1

```

```

68      '1' after 31500 us, '0' after 31600 us, -- bit 30
69      '1' after 32500 us, '0' after 32600 us, -- bit 31
70      '1' after 33500 us, '0' after 33600 us, -- bit 32
71      '1' after 34500 us, '0' after 34700 us, -- bit 33 1
72      '1' after 35500 us, '0' after 35600 us, -- bit 34
73      '1' after 36500 us, '0' after 36700 us, -- bit 35 1
74      '1' after 37500 us, '0' after 37600 us, -- bit 36
75      '1' after 38500 us, '0' after 38600 us, -- bit 37
76      '1' after 39500 us, '0' after 39600 us, -- bit 38
77      '1' after 40500 us, '0' after 40700 us, -- bit 39 1
78      '1' after 41500 us, '0' after 41600 us, -- bit 40
79      '1' after 42500 us, '0' after 42600 us, -- bit 41
80      '1' after 43500 us, '0' after 43700 us, -- bit 42 1
81      '1' after 44500 us, '0' after 44600 us, -- bit 43
82      '1' after 45500 us, '0' after 45600 us, -- bit 44
83      '1' after 46500 us, '0' after 46600 us, -- bit 45
84      '1' after 47500 us, '0' after 47700 us, -- bit 46 1
85      '1' after 48500 us, '0' after 48600 us, -- bit 47
86      '1' after 49500 us, '0' after 49600 us, -- bit 48
87      '1' after 50500 us, '0' after 50700 us, -- bit 49 1
88      '1' after 51500 us, '0' after 51600 us, -- bit 50
89      '1' after 52500 us, '0' after 52600 us, -- bit 51
90      '1' after 53500 us, '0' after 53700 us, -- bit 52 1
91      '1' after 54500 us, '0' after 54600 us, -- bit 53
92      '1' after 55500 us, '0' after 55700 us, -- bit 54 1
93      '1' after 56500 us, '0' after 56600 us, -- bit 55
94      '1' after 57500 us, '0' after 57600 us, -- bit 56
95      '1' after 58500 us, '0' after 58600 us, -- bit 57
96      '1' after 59500 us, '0' after 59700 us, -- bit 58 1
97      '1' after 61500 us, '0' after 61600 us; -- bit 59 & bit 0
98      -- dit genereert een dcf-signaal voor het ontwerp
99      -- waarin het volgende gecodeerd is:
100     -- DD-MM-JJ: 08-12-'14 & HH:MM: 11:48 & maandag
101     -- vervolgens DD-MM-JJ: 08-12-'14 & HH:MM: 11:49 & maandag
102     -- Bits 0 t/m 17 zijn 0
103     -- Bit 18 is 1
104     -- Bit 19 is 0
105     -- Bit 20 is 1
106     -- Bits 21 t/m 27 zijn de minuten (48)
107     -- Bit 28 is parity bit v.d. minuten (1)
108     -- Bits 29 t/m 34 zijn de uren (11)
109     -- Bit 35 is parity bit v.d. uren (1)
110     -- Bits 36 t/m 41 zijn de dagen v.d. maand (8)
111     -- Bits 42 t/m 44 zijn de dagen v.d. week (ma)
112     -- Bits 45 t/m 49 vormen de maand (12)
113     -- Bits 50 t/m 57 vormen het jaar (binnen een eeuw) (14)
114     -- Bits 58 is parity bit over bits 36 t/m 57 (1)
115
116     sync_time: synctime port map(clk, reset, dcf_in, dcf_led, ready, minuten,
117                               uren, weekdag, dagen, maanden, jaren);
118
119 end behaviour;

```

## B.2.6. TESTBENCH KLOKDELER

```

1  -- Alex Oudsen, 4325494
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  entity klokdeler_tb is
7  end klokdeler_tb;


---


1  -- Alex Oudsen, 4325494
2  -- Dit is de testbench voor simulatie op schaal
3  -- Verwachte respons is een 781.25 Hz signaal
4  -- met een periodetijd van 1280 microseonden
5  -- Advies simulatietijd: 10 ms.
6
7  library ieee;
8  use ieee.std_logic_1164.all;

```

```

9
10 architecture behaviour of klokdeler_tb is
11     component klokdeler is
12         port(clk:  in  std_logic;
13             reset: in  std_logic;
14             clk_1hz: out std_logic);
15     end component klokdeler;
16
17 signal clk, reset, clk_1hz: std_logic;
18
19 begin
20     clk <= '1' after 0 ns, -- Dit genereert een 25 MHz kloksignaal
21             '0' after 20 ns when clk /= '0' else '1' after 20 ns;
22     reset    <= '1' after 0 ns, '0' after 50 ns;
23
24     divide: klokdeler port map(clk, reset, clk_1hz);
25
26 end behaviour;

```

### B.2.7. TESTBENCH MOD60 (SECONDEN) TELLER

```

1  -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity mod60_tel_tb is
7 end mod60_tel_tb;

1 -- Alex Oudsen, 4325494
2 -- Dit is de testbench voor simulatie op schaal
3 -- Verwachte respons is een count die iedere
4 -- opgaande klokflank van clk_in 1 optelt bij
5 -- de waarde van count, waarbij na 59 weer 0 komt
6 -- Bovendien wordt er tweemaal gesynchroniseerd;
7 -- eenmaal met ref = 23 en eenmaal met ref = 59
8 -- Ook wordt aan de uitgang een klok gegenereerd met
9 -- frequentie 1/60 van de frequentie van clk_in
10 -- Advies simulatietijd: 80 ms.
11
12 library ieee;
13 use ieee.std_logic_1164.all;
14
15 architecture behaviour_bcd_clk of mod60_tel_tb is
16     component mod60_clk_bcd is
17         port (clk:  in  std_logic;
18             clk_in:  in  std_logic;
19             reset:   in  std_logic;
20             sync_now: in  std_logic;
21             ref:     in  std_logic_vector(6 downto 0);
22             m_clk:   out std_logic);
23     end component mod60_clk_bcd;
24
25     signal clk, clk_in, m_clk, reset, sync_now: std_logic;
26     signal ref: std_logic_vector(6 downto 0);
27
28 begin
29     clk      <= '1' after 0 ns, -- Dit genereert een 100 kHz signaal
30             '0' after 5 us when clk /= '0' else '1' after 5 us;
31     clk_in    <= '1' after 0 ns, -- Dit genereert een 1 kHz signaal
32             '0' after 500 us when clk_in /= '0' else '1' after 500 us;
33     reset    <= '1' after 0 ns, '0' after 600 us;
34     sync_now <= '0' after 0 ns, '1' after 5 ms, '0' after 5020 us,
35             '1' after 10 ms, '0' after 10020 us;
36     ref      <= "0100011" after 0 ns, "1011001" after 6 ms;
37
38     count60: mod60_clk_bcd port map(clk, clk_in, reset, sync_now, ref, m_clk);
39
40 end behaviour_bcd_clk;

```

### B.2.8. TESTBENCH MOD60 (MINUTEN) TELLER

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity mod60_tel_tb is
7 end mod60_tel_tb;



---


1 -- Alex Oudsen, 4325494
2 -- Dit is de testbench voor simulatie op schaal
3 -- Verwachte respons is een count die iedere
4 -- opgaande klokflank van clk_in 1 optelt bij
5 -- de waarde van count, waarbij na 59 weer 0 komt
6 -- Bovendien wordt er tweemaal gesynchroniseerd;
7 -- eenmaal met ref = 23 en eenmaal met ref = 59
8 -- Ook wordt aan de uitgang een klok gegenereerd met
9 -- frequentie 1/60 van de frequentie van clk_in
10 -- Advies simulatietijd: 50 ms.
11
12 library ieee;
13 use ieee.std_logic_1164.all;
14
15 architecture behaviour_bcd of mod60_tel_tb is
16   component mod60_tel_bcd is
17     port (clk: in std_logic;
18           clk_in: in std_logic;
19           reset: in std_logic;
20           sync_now: in std_logic;
21           ref: in std_logic_vector(6 downto 0);
22           count: out std_logic_vector(6 downto 0);
23           h_clk: out std_logic);
24   end component mod60_tel_bcd;
25
26   signal clk, clk_in, h_clk, reset, sync_now: std_logic;
27   signal ref, count: std_logic_vector(6 downto 0);
28
29 begin
30   clk      <= '1' after 0 ns, -- Dit genereert een 100 kHz signaal
31             '0' after 5 us when clk /= '0' else '1' after 5 us;
32   clk_in    <= '1' after 0 ns, -- Dit genereert een 1 kHz signaal
33             '0' after 500 us when clk_in /= '0' else '1' after 500 us;
34   reset     <= '1' after 0 ns, '0' after 600 us;
35   sync_now  <= '0' after 0 ns, '1' after 5 ms, '0' after 5020 us,
36             '1' after 10 ms, '0' after 10020 us;
37   ref       <= "0100011" after 0 ns, "1011001" after 6 ms;
38
39   count60: mod60_tel_bcd port map(clk, clk_in, reset, sync_now, ref, count, h_clk);
40
41 end behaviour_bcd;

```

### B.2.9. TESTBENCH MOD24 (UREN) TELLER

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity mod24_tel_tb is
7 end mod24_tel_tb;



---


1 -- Alex Oudsen, 4325494
2 -- Dit is de testbench voor simulatie op schaal
3 -- Verwachte respons is een count die iedere
4 -- opgaande klokflank van clk_in 1 optelt bij
5 -- de bcd waarde van count, waarbij na 23 weer 0 komt
6 -- Bovendien wordt er tweemaal gesynchroniseerd;
7 -- eenmaal met ref = 10 en eenmaal met ref = 23
8 -- Advies simulatietijd: 40 ms.
9
10 library ieee;

```

```

11 use ieee.std_logic_1164.all;
12
13 architecture behaviour_bcd of mod24_tel_tb is
14   component mod24_tel_bcd is
15     port (clk: in std_logic;
16           clk_in: in std_logic;
17           reset: in std_logic;
18           sync_now: in std_logic;
19           ref: in std_logic_vector(5 downto 0);
20           count: out std_logic_vector(5 downto 0));
21   end component mod24_tel_bcd;
22
23   signal clk, clk_in, reset, sync_now: std_logic;
24   signal ref, count: std_logic_vector(5 downto 0);
25
26 begin
27   clk      <= '1' after 0 ns, -- Dit genereert een 100 kHz signaal
28             '0' after 5 us when clk /= '0' else '1' after 5 us;
29   clk_in    <= '1' after 0 ns, -- Dit genereert een 1 kHz signaal
30             '0' after 500 us when clk_in /= '0' else '1' after 500 us;
31   reset     <= '1' after 0 ns, '0' after 600 us;
32   sync_now  <= '0' after 0 ns, '1' after 5 ms, '0' after 5020 us,
33             '1' after 10 ms, '0' after 10020 us;
34   ref       <= "010000" after 0 ns, "100011" after 6 ms;
35
36   count24: mod24_tel_bcd port map(clk, clk_in, reset, sync_now, ref, count);
37
38 end behaviour_bcd;

```

## B.2.10. TESTBENCH AUTONOME SYNCHRONISEERBARE KLOK

```

1 -- Alex Oudsen, 4325494
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5
6 entity autosyncclk_tb is
7 end autosyncclk_tb;



---


1 -- Alex Oudsen, 4325494
2 -- Dit is de testbench voor simulatie op schaal
3 -- De verwachte respons geeft twee synchronisatiemomenten,
4 -- namelijk 10:10 en 23:45, waaromheen de klok autonoom
5 -- verder zal tellen op de interne klok
6 -- Advies simulatietijd: 80 ms.
7
8 library ieee;
9 use ieee.std_logic_1164.all;
10
11 architecture behaviour_bcd of autosyncclk_tb is
12   component ausy_klok_bcd is
13     port (clk: in std_logic;
14           s_clk: in std_logic;
15           reset: in std_logic;
16           sync_now: in std_logic;
17           min_ref: in std_logic_vector(6 downto 0);
18           hr_ref: in std_logic_vector(5 downto 0);
19           minutes: out std_logic_vector(6 downto 0);
20           hours: out std_logic_vector(5 downto 0));
21   end component ausy_klok_bcd;
22
23   signal clk, s_clk, reset, sync_now: std_logic;
24   signal minutes, min_ref: std_logic_vector(6 downto 0);
25   signal hours, hr_ref: std_logic_vector(5 downto 0);
26
27 begin
28   clk      <= '1' after 0 ns, -- Dit genereert een 100 kHz signaal
29             '0' after 5 us when clk /= '0' else '1' after 5 us;
30   s_clk    <= '1' after 0 ns, -- Dit genereert een 1 kHz signaal
31             '0' after 500 us when s_clk /= '0' else '1' after 500 us;
32   reset    <= '1' after 0 ns, '0' after 600 us;

```

```

33      sync_now      <= '0' after 0 ns, '1' after 5 ms, '0' after 5020 us,
34          '1' after 10 ms, '0' after 10020 us;
35      min_ref      <= "0010000" after 0 ns, "1000101" after 6 ms;
36      hr_ref       <= "010000" after 0 ns, "100011" after 6 ms;
37
38      klok: ausy_klok_bcd port map(clk, s_clk, reset, sync_now, min_ref, hr_ref, minutes, hours
39          );
40 end behaviour_bcd;

```

### B.2.11. TESTBENCH DCF77 (TOP-LEVEL)

```

1  -- Alex Oudsen, 4325494
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  entity dcf77_tb is
7  end dcf77_tb;



---


1  -- Alex Oudsen, 4325494
2  -- Dit is testbench voor simulatie met de 'echte' klok
3  -- De verwachte respons bestaat naast het 1hz signaal uit
4  -- een led-signaal wat afhankelijk is van de kwaliteit van het
5  -- DCF77 signaal, de huidige tijd (gesynchroniseerd met het DCF77
6  -- signaal indien mogelijk en de huidige datum (inclusief ready signaal)
7  -- Het DCF77 testsignaal bevat: maandag 8 december, 11:48
8  -- Advies simulatietijd: 65 ms.
9
10 library ieee;
11 use ieee.std_logic_1164.all;
12
13 architecture behaviour_bcd of dcf77_tb is
14     component dcf77_bcd is
15         port (clk:           in std_logic;
16             reset:        in std_logic;
17             dcf_in:        in std_logic;
18             dcf_led:       out std_logic;
19             clk_1hz:       out std_logic;
20             minutes:      out std_logic_vector(6 downto 0);
21             hours:        out std_logic_vector(5 downto 0);
22             weekday:      out std_logic_vector(2 downto 0);
23             day:          out std_logic_vector(5 downto 0);
24             month:        out std_logic_vector(4 downto 0);
25             year:         out std_logic_vector(7 downto 0);
26             date_ready:   out std_logic);
27     end component dcf77_bcd;
28
29     signal clk, reset, dcf_in, dcf_led, clk_1hz, date_ready: std_logic;
30     signal minutes: std_logic_vector(6 downto 0);
31     signal hours, day: std_logic_vector(5 downto 0);
32     signal month: std_logic_vector(4 downto 0);
33     signal weekday: std_logic_vector(2 downto 0);
34     signal year: std_logic_vector(7 downto 0);
35
36 begin
37     clk <= '1' after 0 ns,
38         '0' after 16 ns when clk /= '0' else '1' after 16 ns;
39     reset <= '1' after 0 ns, '0' after 50 us;
40     dcf_in <= '0' after 0 ns,
41         '1' after 1500 us, '0' after 1600 us, -- bit 59 & bit 0
42         '1' after 2500 us, '0' after 2600 us, -- bit 1
43         '1' after 3500 us, '0' after 3600 us, -- bit 2
44         '1' after 4500 us, '0' after 4600 us, -- bit 3
45         '1' after 5500 us, '0' after 5600 us, -- bit 4
46         '1' after 6500 us, '0' after 6600 us, -- bit 5
47         '1' after 7500 us, '0' after 7600 us, -- bit 6
48         '1' after 8500 us, '0' after 8600 us, -- bit 7
49         '1' after 9500 us, '0' after 9600 us, -- bit 8
50         '1' after 10500 us, '0' after 10600 us, -- bit 9
51         '1' after 11500 us, '0' after 11600 us, -- bit 10

```

```

52      '1' after 12500 us, '0' after 12600 us, -- bit 11
53      '1' after 13500 us, '0' after 13600 us, -- bit 12
54      '1' after 14500 us, '0' after 14600 us, -- bit 13
55      '1' after 15500 us, '0' after 15600 us, -- bit 14
56      '1' after 16500 us, '0' after 16600 us, -- bit 15
57      '1' after 17500 us, '0' after 17600 us, -- bit 16
58      '1' after 18500 us, '0' after 18600 us, -- bit 17
59      '1' after 19500 us, '0' after 19700 us, -- bit 18 1
60      '1' after 20500 us, '0' after 20600 us, -- bit 19
61      '1' after 21500 us, '0' after 21700 us, -- bit 20 1
62      '1' after 22500 us, '0' after 22600 us, -- bit 21
63      '1' after 23500 us, '0' after 23600 us, -- bit 22
64      '1' after 24500 us, '0' after 24600 us, -- bit 23
65      '1' after 25500 us, '0' after 25700 us, -- bit 24 1
66      '1' after 26500 us, '0' after 26600 us, -- bit 25
67      '1' after 27500 us, '0' after 27600 us, -- bit 26
68      '1' after 28500 us, '0' after 28700 us, -- bit 27 1
69      '1' after 29500 us, '0' after 29700 us, -- bit 28 1
70      '1' after 30500 us, '0' after 30700 us, -- bit 29 1
71      '1' after 31500 us, '0' after 31600 us, -- bit 30
72      '1' after 32500 us, '0' after 32600 us, -- bit 31
73      '1' after 33500 us, '0' after 33600 us, -- bit 32
74      '1' after 34500 us, '0' after 34700 us, -- bit 33 1
75      '1' after 35500 us, '0' after 35600 us, -- bit 34
76      '1' after 36500 us, '0' after 36700 us, -- bit 35 1
77      '1' after 37500 us, '0' after 37600 us, -- bit 36
78      '1' after 38500 us, '0' after 38600 us, -- bit 37
79      '1' after 39500 us, '0' after 39600 us, -- bit 38
80      '1' after 40500 us, '0' after 40700 us, -- bit 39 1
81      '1' after 41500 us, '0' after 41600 us, -- bit 40
82      '1' after 42500 us, '0' after 42600 us, -- bit 41
83      '1' after 43500 us, '0' after 43700 us, -- bit 42 1
84      '1' after 44500 us, '0' after 44600 us, -- bit 43
85      '1' after 45500 us, '0' after 45600 us, -- bit 44
86      '1' after 46500 us, '0' after 46600 us, -- bit 45
87      '1' after 47500 us, '0' after 47700 us, -- bit 46 1
88      '1' after 48500 us, '0' after 48600 us, -- bit 47
89      '1' after 49500 us, '0' after 49600 us, -- bit 48
90      '1' after 50500 us, '0' after 50700 us, -- bit 49 1
91      '1' after 51500 us, '0' after 51600 us, -- bit 50
92      '1' after 52500 us, '0' after 52600 us, -- bit 51
93      '1' after 53500 us, '0' after 53700 us, -- bit 52 1
94      '1' after 54500 us, '0' after 54600 us, -- bit 53
95      '1' after 55500 us, '0' after 55700 us, -- bit 54 1
96      '1' after 56500 us, '0' after 56600 us, -- bit 55
97      '1' after 57500 us, '0' after 57600 us, -- bit 56
98      '1' after 58500 us, '0' after 58600 us, -- bit 57
99      '1' after 59500 us, '0' after 59700 us, -- bit 58 1
100     '1' after 61500 us, '0' after 61600 us; -- bit 59 & bit 0
101     -- dit genereert een dcf-signaal voor het ontwerp
102     -- waarin het volgende gecodeerd is:
103     -- DD-MM-JJ: 08-12-'14 & HH:MM: 11:48 & maandag
104     -- vervolgens DD-MM-JJ: 08-12-'14 & HH:MM: 11:49 & maandag
105     -- Bits 0 t/m 17 zijn 0
106     -- Bit 18 is 1
107     -- Bit 19 is 0
108     -- Bit 20 is 1
109     -- Bits 21 t/m 27 zijn de minuten (48)
110     -- Bit 28 is parity bit v.d. minuten (1)
111     -- Bits 29 t/m 34 zijn de uren (11)
112     -- Bit 35 is parity bit v.d. uren (1)
113     -- Bits 36 t/m 41 zijn de dagen v.d. maand (8)
114     -- Bits 42 t/m 44 zijn de dagen v.d. week (ma)
115     -- Bits 45 t/m 49 vormen de maand (12)
116     -- Bits 50 t/m 57 vormen het jaar (binnen een eeuw) (14)
117     -- Bits 58 is parity bit over bits 36 t/m 57 (1)
118
119     synced_time: dcf77_bcd port map(clk, reset, dcf_in, dcf_led, clk_1hz, minutes, hours,
120                               weekday, day, month, year, date_ready);
121   end behaviour_bcd;

```

## B.3. FPGA HULPBESTANDEN VAN HET DCF77 BLOK

### B.3.1. DATE READY BUFFER

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity buff is
5     port(clk      : in std_logic;
6           reset   : in std_logic;
7           switch8: in std_logic;
8           date_r   : in std_logic;
9           ready    : out std_logic);
10 end buff;
11
12 library IEEE;
13 use IEEE.std_logic_1164.ALL;
14 use IEEE.numeric_std.ALL;
15
16 architecture behaviour of buff is
17
18     signal red, new_red: std_logic;
19
20 begin
21
22     ready <= red;
23
24     process(clk, reset) is
25     begin
26         if(clk'event and clk = '1') then
27             if(reset = '1') then
28                 red <= '0';
29             else
30                 red <= new_red;
31             end if;
32         end if;
33     end process;
34
35     process(date_r, red, switch8) is
36     begin
37         if(date_r = '1') then
38             new_red <= '1';
39         elsif(switch8 = '1') then
40             new_red <= '0';
41         else
42             new_red <= red;
43         end if;
44     end process;
45 end behaviour;

```

### B.3.2. KLOKDELER 50 MHZ NAAR 32 KHZ

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity gen32khz is
5     port(clk50mhz : in std_logic; -- 50 Mhz clock input
6           clk32khz : out std_logic); -- 10 Hz clock output
7 end gen32khz;
8
9 library IEEE;
10 use IEEE.std_logic_1164.ALL;
11 use IEEE.numeric_std.ALL;
12
13 architecture behaviour of gen32khz is
14     signal count, new_count : unsigned (10 downto 0);
15     signal new_clk32khz : std_logic;
16 begin
17     lbl1: process (clk50mhz)
18     begin
19         if (clk50mhz'event and clk50mhz = '1') then

```

```

20         count <= new_count;
21             clk32khz <= new_clk32khz;
22     end if;
23 end process;
24 lbl2: process (count)
25 begin
26     if (count >= 1563) then
27         new_count <= (others => '0');
28         new_clk32khz <= '0';
29     elsif (count >= 783) then
30         new_count <= count + 1;
31         new_clk32khz <= '1';
32     else
33         new_count <= count + 1;
34         new_clk32khz <= '0';
35     end if;
36 end process;
37 end behaviour;

```

### B.3.3. KLOKDELER 50 MHz NAAR 10 Hz

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity gen10hz is
5     port(clk50mhz : in std_logic;    -- 50 Mhz clock input
6           res      : in std_logic;    -- reset
7           clk10hz : out std_logic);   -- 10 Hz clock output
8 end gen10hz;
9
10 library IEEE;
11 use IEEE.std_logic_1164.ALL;
12 use IEEE.numeric_std.ALL;
13
14 architecture behaviour of gen10hz is
15     signal count, new_count : unsigned(23 downto 0);
16     signal new_clk10hz : std_logic;
17 begin
18     lbl1: process (clk50mhz)
19     begin
20         if (clk50mhz'event and clk50mhz = '1') then
21             if (res = '1') then
22                 count <= (others => '0');
23                 clk10hz <= '0';
24             else
25                 count <= new_count;
26                 clk10hz <= new_clk10hz;
27             end if;
28         end if;
29     end process;
30     lbl2: process (count)
31     begin
32         if (count >= 5000000) then
33             new_count <= (others => '0');
34             new_clk10hz <= '0';
35         elsif (count >= 2500000) then
36             new_count <= count + 1;
37             new_clk10hz <= '1';
38         else
39             new_count <= count + 1;
40             new_clk10hz <= '0';
41         end if;
42     end process;
43 end behaviour;

```

### B.3.4. DCF GENERATOR

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity dcf_gen is

```

```

5      port(clk50mhz : in std_logic; -- 50 Mhz clock input
6          res      : in std_logic; -- reset
7          dcf      : out std_logic); -- DCF output signal
8 end dcf_gen;
9
10 library IEEE;
11 use IEEE.std_logic_1164.ALL;
12 use IEEE.numeric_std.ALL;
13
14 architecture behaviour of dcf_gen is
15 component gen10hz
16     port (clk50mhz : in std_logic;
17             res      : in std_logic;
18             clk10hz  : out std_logic);
19 end component;
20
21 signal cnt100ms, new_cnt100ms : unsigned(3 downto 0);
22 signal cnt1s, new_cnt1s       : unsigned(5 downto 0);
23 signal clk10hz, cur_clk10hz   : std_logic;
24
25 signal code : std_logic_vector(59 downto 1);
26
27 begin
28
29     code <= (
30         '1', -- P3
31         '0', '0', '0', '1', '0', '1', '0', '0', -- year : 14
32         '1', '0', '0', '1', '0', -- month : dec
33         '0', '1', '0', -- week day : tuesday
34         '0', '0', '1', '0', '1', -- calender day : 9
35         '0', '1', '0', '0', '1', '0', -- P2 + hours : 22
36         '0', '0', '0', '1', '0', '0', '1', '0', -- P1 + min : 12
37         '1', '0', '1', '0', '1', '0', '1', '0', -- not used
38         '1', '0', '1', '0', '1', '0', '1', '0', -- not used
39         '0' -- bit 0
40     );
41
42     lbl0: gen10hz port map (clk50mhz, res, clk10hz);
43
44     lbl1a: process (clk50mhz)
45 begin
46     if (clk50mhz'event and clk50mhz = '1') then
47         if (res = '1') then
48             cnt100ms <= (others => '0');
49             cnt1s <= (others => '0');
50         else
51             cnt100ms <= new_cnt100ms;
52             cnt1s <= new_cnt1s;
53         end if;
54         cur_clk10hz <= clk10hz;
55     end if;
56 end process;
57
58     lbl1b: process (cnt100ms, cnt1s, clk10hz, cur_clk10hz)
59 begin
60     new_cnt1s <= cnt1s; -- default
61     new_cnt100ms <= cnt100ms; -- default
62     if (clk10hz = '1' and cur_clk10hz = '0') then
63         if (cnt100ms = 9) then
64             new_cnt100ms <= (others => '0');
65             if (cnt1s = 59) then
66                 new_cnt1s <= (others => '0');
67             else
68                 new_cnt1s <= cnt1s + 1;
69             end if;
70         else
71             new_cnt100ms <= cnt100ms + 1;
72         end if;
73     end if;
74 end process;
75

```

```

76     dcf <= '0' when cnt1s = 0 else
77         '1' when cnt100ms = 0 else
78             '1' when cnt100ms = 1 and code(to_integer(cnt1s)) = '1' else
79                 '0';
80
81 end behaviour;

```

### B.3.5. SCHAKELAAR VOOR DE DCF GENERATOR

```

1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  entity dcf_switch is
5      port(switch9: in std_logic;
6          dcf: in std_logic;
7          dcf_out: out std_logic);
8  end dcf_switch;
9
10 library IEEE;
11 use IEEE.std_logic_1164.ALL;
12 use IEEE.numeric_std.ALL;
13
14 architecture behaviour of dcf_switch is
15
16 begin
17
18     process(switch9, dcf) is
19     begin
20         if(switch9 = '1') then
21             dcf_out <= dcf;
22         else
23             dcf_out <= '0';
24         end if;
25     end process;
26 end behaviour;

```

### B.3.6. SCHAKELMOGELIJKHEID TUSSEN DAG, MAAND EN JAAR

```

1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  entity switch is
5      port(day : in std_logic_vector(5 downto 0);
6          month : in std_logic_vector(4 downto 0);
7          year : in std_logic_vector(7 downto 0);
8          switch0: in std_logic;
9          switch1: in std_logic;
10         switch2: in std_logic;
11         output : out std_logic_vector(7 downto 0));
12 end switch;
13
14 library IEEE;
15 use IEEE.std_logic_1164.ALL;
16 use IEEE.numeric_std.ALL;
17
18 architecture behaviour of switch is
19
20 begin
21
22 process(switch0, switch1, switch2, day, month, year) is
23
24 begin
25     if(switch0 = '1' and switch1 = '0' and switch2 = '0') then
26         output <= year;
27     elsif(switch0 = '0' and switch1 = '1' and switch2 = '0') then
28         output(7 downto 5) <= (others => '0');
29         output(4 downto 0) <= month;
30     elsif(switch0 = '0' and switch1 = '0' and switch2 = '1') then
31         output(7 downto 6) <= (others => '0');
32         output(5 downto 0) <= day;
33     else

```

```

34         output <= (others => '0');
35     end if;
36 end process;
37 end behaviour;
```

### B.3.7. AANSTURING SEVEN-SEGMENT DISPLAY

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_UNSIGNED.ALL;
4 use IEEE.STD_LOGIC_ARITH.ALL;
5
6 entity sevenseg is
7 port (clk      : in std_logic;
8        reset    : in std_logic;
9        minuten  : in std_logic_vector(6 downto 0); --minuten input
10       uren     : in std_logic_vector(5 downto 0); --minuten input
11       seg7_min1 : out std_logic_vector(6 downto 0); -- 7 bit decoded output.
12       seg7_min2 : out std_logic_vector(6 downto 0); -- 7 bit decoded output.
13       seg7_uur1 : out std_logic_vector(6 downto 0); -- 7 bit decoded output.
14       seg7_uur2 : out std_logic_vector(6 downto 0)); -- 7 bit decoded output.
15
16 end sevenseg;
17
18 architecture Behavioral of sevenseg is
19
20 signal minuten1: std_logic_vector(3 downto 0);
21 signal minuten2: std_logic_vector(2 downto 0);
22 signal uren1: std_logic_vector(3 downto 0);
23 signal uren2: std_logic_vector(1 downto 0);
24 begin
25 minuten1 <= minuten(3 downto 0);
26 minuten2 <= minuten(6 downto 4);
27 uren1 <= uren(3 downto 0);
28 uren2 <= uren(5 downto 4);
29
30
31 process (clk, reset)
32 begin
33 if (clk'event and clk='1') then
34     if(reset = '1') then
35         seg7_min1 <= (others => '0');
36         seg7_min2 <= (others => '0');
37         seg7_uur1 <= (others => '0');
38         seg7_uur2 <= (others => '0');
39     else
40         case minuten1 is
41             when "0000"=> seg7_min1 <="0000001"; -- '0'
42             when "0001"=> seg7_min1 <="1001111"; -- '1'
43             when "0010"=> seg7_min1 <="0010010"; -- '2'
44             when "0011"=> seg7_min1 <="0000110"; -- '3'
45             when "0100"=> seg7_min1 <="1001100"; -- '4'
46             when "0101"=> seg7_min1 <="0100100"; -- '5'
47             when "0110"=> seg7_min1 <="0100000"; -- '6'
48             when "0111"=> seg7_min1 <="0001111"; -- '7'
49             when "1000"=> seg7_min1 <="0000000"; -- '8'
50             when "1001"=> seg7_min1 <="0000100"; -- '9'
51             --nothing is displayed when a number more than 9 is given as input.
52             when others=> seg7_min1 <="1111111";
53         end case;
54
55         case minuten2 is
56             when "000"=> seg7_min2 <="0000001"; -- '0'
57             when "001"=> seg7_min2 <="1001111"; -- '1'
58             when "010"=> seg7_min2 <="0010010"; -- '2'
59             when "011"=> seg7_min2 <="0000110"; -- '3'
60             when "100"=> seg7_min2 <="1001100"; -- '4'
61             when "101"=> seg7_min2 <="0100100"; -- '5'
62             when "110"=> seg7_min2 <="0100000"; -- '6'
63             --nothing is displayed when a number more than 6 is given as input.
64             when others=> seg7_min2 <="1111111";
```

```
65      end case;
66
67      case uren1 is
68          when "0000"=> seg7_uur1 <="0000001"; -- '0'
69          when "0001"=> seg7_uur1 <="1001111"; -- '1'
70          when "0010"=> seg7_uur1 <="0010010"; -- '2'
71          when "0011"=> seg7_uur1 <="0000110"; -- '3'
72          when "0100"=> seg7_uur1 <="1001100"; -- '4'
73          when "0101"=> seg7_uur1 <="0100100"; -- '5'
74          when "0110"=> seg7_uur1 <="0100000"; -- '6'
75          when "0111"=> seg7_uur1 <="0001111"; -- '7'
76          when "1000"=> seg7_uur1 <="0000000"; -- '8'
77          when "1001"=> seg7_uur1 <="0000100"; -- '9'
78          --nothing is displayed when a number more than 9 is given as input.
79          when others=> seg7_uur1 <="1111111";
80      end case;
81
82      case uren2 is
83          when "00"=> seg7_uur2 <="0000001"; -- '0'
84          when "01"=> seg7_uur2 <="1001111"; -- '1'
85          when "10"=> seg7_uur2 <="0010010"; -- '2'
86          --nothing is displayed when a number more than 2 is given as input.
87          when others=> seg7_uur2 <="1111111";
88      end case;
89  end if;
90 end if;
91 end process;
92
93 end Behavioral;
```

## B.4. VHDL CODE CONTROLLER

### B.4.1. TOP LEVEL ENTITY

```

1 -- Rens Hamburger 4292936
2 library IEEE;
3 use IEEE.std_logic_1164.ALL;
4
5 entity controller is
6   port(clk      :in  std_logic;
7        reset    :in  std_logic;
8        knoppen :in  std_logic_vector(3 downto 0);
9        wekker   :out std_logic_vector(15 downto 0);
10       menu_state :out std_logic_vector(2 downto 0));
11 end controller;

```

### B.4.2. BEHAVIOURAL VHDL CODE CONTROLLER

```

1 --Rens Hamburger 4292936
2 --Het portmappen van gebruikte componenten voor de complete controller
3 library IEEE;
4 use IEEE.std_logic_1164.ALL;
5
6 architecture behaviour of controller is
7 component menu is
8   -- Onderdelen van de schakeling gedaan worden
9   port(clk      :in  std_logic;
10        reset    :in  std_logic;
11        knoppen :in  std_logic_vector(3 downto 0);
12        wekdata  :in  std_logic_vector(15 downto 0);
13        enable   :out std_logic;
14        wekker   :out std_logic_vector(15 downto 0);
15        menu_signal :out std_logic_vector(2 downto 0));
16 end component menu;
17
17 component geheugen is
18   -- 14 bit opslag
19   port(clk      :in  std_logic;
20        reset    :in  std_logic;
21        enable   :in  std_logic;
22        wek_in   :in  std_logic_vector(15 downto 0);
23        wek_out  :out std_logic_vector(15 downto 0));
24 end component geheugen;
25
26 component buff is
27   -- De buffer die speciaal gemaakt is voor de menu
28   -- met extra eigenschappen
29   port(clk      :in  std_logic;
30        reset    :in  std_logic;
31        knoppen_in :in  std_logic_vector(3 downto 0);
32        knoppen_out :out std_logic_vector(3 downto 0));
33 end component buff;
34
35 signal knoppen_buff : std_logic_vector(3 downto 0);
36 signal wekdata_men, wekker_men : std_logic_vector(15 downto 0);
37 signal write_enable : std_logic;
38
39 begin
40   buffer_portmap : buff port map (clk,reset,knoppen,knoppen_buff);
41   menu_portmap : menu port map (clk,reset,knoppen_buff,wekdata_men,write_enable,wekker_men,
42                                 menu_state);
43   memory_portmap: geheugen port map (clk,reset,write_enable,wekker_men,wekdata_men);
44   wekker <= wekdata_men;
45
46 end behaviour;

```

### B.4.3. MENU ENTITY

```

1 -- Kevin Hill 4287592 & Rens Hamburger 4292936
2 library IEEE;
3 use IEEE.std_logic_1164.ALL;
4 use IEEE.Numeric_Std.all;

```

```

5
6 entity menu is
7   port(clk      :in  std_logic;
8        reset    :in  std_logic;
9        knoppen  :in  std_logic_vector(3 downto 0);
10       wekdata  :in  std_logic_vector(15 downto 0);
11       enable   :out std_logic;
12       wekker   :out std_logic_vector(15 downto 0);
13       menu_signal :out std_logic_vector(2 downto 0));
14 end menu;

```

#### B.4.4. BEHAVIOURAL VHDL CODE MENU

```

1  -- Kevin Hill 4287592 & Rens Hamburger 4292936
2  -- FSM is voor het overzicht in drie processen verwerkt
3  -- 1ste process zorgt ervoor dat alles op de opgande klokflank gebeurt, en gaat naar de
   -- nieuwe state toe (of de reset-state)
4  -- 2de process voert de state uit
5  -- 3de process bepalen van de nieuwe state
6 library IEEE;
7 use IEEE.std_logic_1164.ALL;
8 use IEEE.numeric_std.all;
9
10 architecture behaviour of menu is
11 type fsm_states is (rust, wekkertijd, led, led_toggle, geluid, geluid_toggle, wekker_toggle,
12                      uren_set, uren_plus, uren_min, minuten_set, minuten_plus, minuten_min);
13 signal state, new_state : fsm_states;
14 begin
15   assign : process(clk, reset) --Daadwerkelijk alles toekennen
16   begin
17     if rising_edge(clk) then
18       if reset = '0' then
19         state <= new_state;
20       else
21         state <= rust;
22       end if;
23     end if;
24   end process assign;
25
26   actie_uitvoeren : process(knoppen, wekdata, clk, reset, state) --Voer acties uit
27   begin
28     case state is
29       when rust =>
30         enable <= '0';
31         wekker <= wekdata;
32         menu_signal <= "000";
33
34       when wekker_toggle =>
35         enable <= '1';
36         wekker(14 downto 0) <= wekdata(14 downto 0);
37         wekker(15) <= not wekdata(15);
38         menu_signal <= "000";
39
40       when wekkertijd =>
41         enable <= '0';
42         wekker <= wekdata;
43         menu_signal <= "101";
44
45       when led =>
46         enable <= '0';
47         wekker <= wekdata;
48         menu_signal <= "011";
49
50       when led_toggle =>
51         enable <= '1';
52         wekker(13 downto 0) <= wekdata(13 downto 0);
53         wekker(14) <= not wekdata(14);
54         wekker(15) <= wekdata(15);
55         menu_signal <= "011";
56
57       when geluid =>

```

```

57         enable <= '0';
58         wekker <= wekdata;
59         menu_signal <= "100";
60
61     when geluid_toggle =>
62         enable <= '1';
63         wekker(12 downto 0) <= wekdata(12 downto 0);
64         wekker(13) <= not wekdata(13);
65         wekker(15 downto 14) <= wekdata(15 downto 14);
66         menu_signal <= "100";
67
68     when uren_set =>
69         enable <= '0';
70         wekker <= wekdata;
71         menu_signal <= "001";
72
73     when uren_plus =>
74         enable <= '1';
75         menu_signal <= "101";
76         if wekdata(12 downto 7) = "100011" then --23
77             wekker(12 downto 7) <= "000000"; --Bij de 23 uur weer opnieuw beginnen
78         else
79             if (wekdata(10 downto 7) = "1001") then --Bij x9 uur 1 op tellen bij de x
80                 en enkele weer terug naar 0
81                 wekker(10 downto 7) <= "0000";
82                 wekker(12 downto 11) <= std_logic_vector(to_unsigned(to_integer(
83                     unsigned(wekdata(12 downto 11))) + 1, 2));
84             else
85                 wekker(10 downto 7) <= std_logic_vector(to_unsigned(to_integer(
86                     unsigned(wekdata(10 downto 7))) + 1, 4)); -- 1 minuut erbij
87                 optellen
88                 wekker(12 downto 11) <= wekdata(12 downto 11); -- Tientallen blijven
89                 constant
90             end if;
91         end if;
92         wekker(15 downto 13) <= wekdata(15 downto 13); -- Af
93         wekker(6 downto 0) <= wekdata(6 downto 0); --Af
94
95     when uren_min =>
96         if wekdata(12 downto 7) = "000000" then
97             wekker(12 downto 7) <= "100011"; --23
98         else
99             if wekdata(10 downto 7) = "0000" then
100                 wekker(10 downto 7) <= "1001";
101                 wekker(12 downto 11) <= std_logic_vector(to_unsigned(to_integer(
102                     unsigned(wekdata(12 downto 11))) - 1, 2));
103             else
104                 wekker(10 downto 7) <= std_logic_vector(to_unsigned(to_integer(
105                     unsigned(wekdata(10 downto 7))) - 1, 4));
106                 wekker(12 downto 11) <= wekdata(12 downto 11);
107             end if;
108         end if;
109         wekker(15 downto 13) <= wekdata(15 downto 13);
110         wekker(6 downto 0) <= wekdata(6 downto 0);
111         enable <= '1';
112         menu_signal <= "101";
113
114     when minuten_set =>
115         enable <= '0';
116         wekker <= wekdata;
117         menu_signal <= "010";
118
119     when minuten_plus =>
120         enable <= '1';
121         if wekdata(6 downto 0) = "1011001" then --59
122             wekker(6 downto 0) <= "0000000"; --Bij de 59 minuten gaan weer op nieuw
123             beginnen
124         else
125             if wekdata(3 downto 0) = "1001" then --Bij x9 minuten 1 op tellen bij de
126                 x en enkele weer terug naar 0
127                 wekker(3 downto 0) <= "0000";

```

```

119          wekker(6 downto 4) <= std_logic_vector(to_unsigned(to_integer(
120              unsigned(wekdata(6 downto 4))) + 1 , 3));
121      else
122          wekker(3 downto 0) <= std_logic_vector(to_unsigned(to_integer(
123              unsigned(wekdata(3 downto 0))) + 1 , 4)); -- 1 minuut erbij
124          optellen
125          wekker(6 downto 4) <= wekdata(6 downto 4); -- Tientallen blijven
126          constant
127      end if;
128  end if;
129  menu_signal <= "111";
130  wekker(15 downto 7) <= wekdata(15 downto 7); --Af
131
132 when minuten_min =>
133     enable <= '1';
134     if wekdata (6 downto 0) = "0000000" then
135         wekker(6 downto 0) <= "1011001"; --59
136     else
137         if wekdata(3 downto 0) = "0000" then --Bij x0 minuten 1 van de tientallen
138             afhalen en de enkele getal op 9 zetten
139             wekker(3 downto 0) <= "1001"; --9
140             wekker(6 downto 4) <= std_logic_vector(to_unsigned(to_integer(
141                 unsigned(wekdata(6 downto 4))) - 1 , 3));
142         else
143             wekker(3 downto 0) <= std_logic_vector(to_unsigned(to_integer(
144                 unsigned(wekdata(3 downto 0))) - 1 , 4));
145             wekker(6 downto 4) <= wekdata(6 downto 4);
146         end if;
147     end if;
148     menu_signal <= "111";
149     wekker(15 downto 7) <= wekdata(15 downto 7); --Af
150
151 end case;
152 end process actie_uitvoeren;
153
154 next_state : process (knoppen, wekdata, clk, reset, state) -- Bepaal nieuwe state
155 begin
156     case state is
157     when rust =>
158         if knoppen(0) = '1' then
159             new_state <= wekkertijd;
160         elsif knoppen(1) = '1' then
161             new_state <= wekker_toggle;
162         else
163             new_state <= rust;
164         end if;
165
166     when wekker_toggle =>
167         new_state <= rust;
168
169     when wekkertijd =>
170         if knoppen(0) = '1' then
171             new_state <= rust;
172         elsif knoppen(2) = '1' then
173             new_state <= geluid;
174         elsif knoppen(3) = '1' then
175             new_state <= led;
176         elsif knoppen(1) = '1' then
177             new_state <= uren_set;
178         else
179             new_state <= wekkertijd;
180         end if;
181
182     when led =>
183         if knoppen(0) = '1' then
184             new_state <= rust;
185         elsif knoppen(2) = '1' then
186             new_state <= wekkertijd;
187         elsif knoppen(3) = '1' then
188             new_state <= geluid;
189         elsif knoppen(1) = '1' then
190             new_state <= led_toggle;

```

```

183         else
184             new_state <= led;
185         end if;
186
187         when led_toggle =>
188             new_state <= led;
189
190         when geluid =>
191             if knoppen(0) = '1' then
192                 new_state <= rust;
193             elsif knoppen(2) = '1' then
194                 new_state <= led;
195             elsif knoppen(3) = '1' then
196                 new_state <= wekkertijd;
197             elsif knoppen(1) = '1' then
198                 new_state <= geluid_toggle;
199             else
200                 new_state <= geluid;
201             end if;
202
203         when geluid_toggle =>
204             new_state <= geluid;
205
206         when uren_set =>
207             if knoppen(0) = '1' then
208                 new_state <= rust;
209             elsif knoppen(2) = '1' then
210                 new_state <= uren_plus;
211             elsif knoppen(3) = '1' then
212                 new_state <= uren_min;
213             elsif knoppen(1) = '1' then
214                 new_state <= minuten_set;
215             else
216                 new_state <= uren_set;
217             end if;
218
219         when uren_plus =>
220             new_state <= uren_set;
221
222         when uren_min =>
223             new_state <= uren_set;
224
225         when minuten_set =>
226             if knoppen(0) = '1' then
227                 new_state <= rust;
228             elsif knoppen(2) = '1' then
229                 new_state <= minuten_plus;
230             elsif knoppen(3) = '1' then
231                 new_state <= minuten_min;
232             elsif knoppen(1) = '1' then
233                 new_state <= rust;
234             else
235                 new_state <= minuten_set;
236             end if;
237
238         when minuten_plus =>
239             new_state <= minuten_set;
240
241         when minuten_min =>
242             new_state <= minuten_set;
243         when others =>
244             new_state <= rust;
245     end case;
246 end process next_state;
247 end behaviour;

```

## B.4.5. MEMORY

```

1  -- Rens Hamburger 4292936
2  library IEEE;
3  use IEEE.std_logic_1164.ALL;

```

```

4
5 entity geheugen is
6   port(clk :in std_logic;
7     reset :in std_logic;
8     enable :in std_logic;
9     wek_in :in std_logic_vector(15 downto 0);
10    wek_out:out std_logic_vector(15 downto 0));
11 end geheugen;

```

#### B.4.6. BEHAVIOURAL VHDL MEMORY

```

1 -- Rens Hamburger 4292936
2 -- 16 bit geheugen element met een positief enable signaal
3 library IEEE;
4 use IEEE.std_logic_1164.ALL;
5
6 architecture behaviour of geheugen is
7   signal wek_opslag,wek_temp : std_logic_vector(15 downto 0 );
8 begin
9   assign : process(clk,reset,wek_temp,wek_in)
10  begin
11    if rising_edge(clk) then
12      if reset = '1' then
13        wek_temp<= (others => '0');
14      else
15        if enable = '1' then
16          wek_temp <= wek_in;
17        else
18          wek_temp <= wek_temp;
19        end if;
20      end if;
21    end if;
22    wek_out<=wek_temp;
23  end process assign;
24 end behaviour;

```

#### B.4.7. ENTITY BUFFER

```

1 --Rens Hamburger 4292936
2 library IEEE;
3 use IEEE.std_logic_1164.ALL;
4
5 entity buff is
6   port(clk :in std_logic;
7     reset :in std_logic;
8     knoppen_in :in std_logic_vector(3 downto 0);
9     knoppen_out :out std_logic_vector(3 downto 0));
10 end buff;

```

#### B.4.8. BEHAVIOURAL VHDL BUFFER

```

1 --Rens Hamburger 4292936
2 --Buffer die zorgt dat we maar 1 periode lang een hoog signaal ontvangen van een knop
3 --Voorkomt ook dat een gebruiker twee knoppen tegelijk kan indrukken wat mogelijk voor voor
4   onverwacht gedrag
5 library IEEE;
6 use IEEE.std_logic_1164.ALL;
7 use IEEE.numeric_std.all;
8
9 architecture behaviour of buff is
10 type fsm_states is (rust, one, zero);
11 signal state, new_state : fsm_states;
12 signal knoppen_temp : std_logic_vector(3 downto 0);
13 begin
14   assign : process(clk, reset) --Daadwerkelijk alles toekennen
15   begin
16     if rising_edge(clk) then
17       if reset = '0' then
18         state <= new_state;
19       else
20         state <= rust;
21     end if;
22   end if;
23   knoppen_out <= state;
24 end process assign;
25
26 end buff;

```

```

20      end if;
21      knoppen_out <= knoppen_temp;
22  end if;
23 end process assign;
24 actie_uitvoeren : process(knoppen_in,clk, reset, state) --Voer acties uit
25 begin
26     case state is
27     when rust =>
28         if ((knoppen_in(0) = '1' xor knoppen_in(1) = '1') xor (knoppen_in(2) = '1'
29             xor knoppen_in(3) = '1')) then
30             new_state <= one;
31             knoppen_temp <= knoppen_in;
32         else
33             new_state <= state;
34             knoppen_temp <= "0000";
35         end if;
36     when zero =>
37         knoppen_temp <= "0000";
38         if ((knoppen_in(0) = '0' and knoppen_in(1) = '0') and (knoppen_in(2) = '0'
39             and knoppen_in(3) = '0')) then
40             new_state <= rust;
41         else
42             new_state <= state;
43         end if;
44     when one =>
45         new_state <= zero;
46         knoppen_temp <= "0000";
47     when others =>
48         new_state <= rust;
49         knoppen_temp <= "0000";
50     end case;
51 end process actie_uitvoeren;
52 end behaviour;

```

## B.5. TESTBENCHS VOOR DE CONTROLLER

### B.5.1. VHDL CONTROLLER

```

1 -- Kevin Hill 4287592 & Rens Hamburger 4292936
2 -- Testbench voor de controller met gebruik van de buffer en geheugen element
3 library IEEE;
4 use IEEE.std_logic_1164.ALL;
5 use IEEE.Numeric_Std.all;
6
7 architecture behaviour of controller_tb is
8 component controller is
9     port(clk      :in  std_logic;
10        reset    :in  std_logic;
11        knoppen:in   std_logic_vector(3 downto 0);
12        wekker :out  std_logic_vector(15 downto 0);
13        menu_state :out  std_logic_vector(2 downto 0));
14 end component controller;
15
16 signal clk, reset          :  std_logic;
17 signal menu_signal         :  std_logic_vector(2 downto 0);
18 signal knoppen             :  std_logic_vector (3 downto 0);
19 signal wekker              :  std_logic_vector (15 downto 0);
20
21 begin
22     clk      <= '1' after 0 ns,
23           '0' after 40 ns when clk /= '0' else '1' after 40 ns;      --31250
24
25     reset    <= '1' after 0 ns,      --knoppen(0) = menu
26           '0' after 128 ns;      --knoppen(1) = set
27                           --knoppen(2) = up
28     knoppen <= "0000" after 0 ns,  --knoppen(3) = down
29           "0010" after 128 ns,   --rust -> wekker_toggle
30           "0000" after 208 ns,   --knoppen(3) = down
31           "0001" after 608 ns,   --rust -> wekkertijd
32           "0000" after 688 ns,   --knoppen(3) = down
33           "0001" after 848 ns,   --wekkertijd -> rust

```

```

34      "0000" after 928 ns,      --knoppen(3) = down
35      "0001" after 1088 ns,    --rust -> wekkertijd
36      "0000" after 1168 ns,    --knoppen(3) = down
37      "0010" after 1328 ns,    --wekkertijd -> uren_set
38      "0000" after 1408 ns,    --knoppen(3) = down
39      "0100" after 1568 ns,    --uren_set -> uren_plus
40      "0000" after 1648 ns,    --knoppen(3) = down
41      "1000" after 2008 ns,    --uren_set -> uren_min
42      "0000" after 2088 ns,    --uren_min -> uren_set
43      "0001" after 2248 ns,    --uren_set -> rust
44      "0000" after 2328 ns,    --knoppen(3) = down
45      "0001" after 2488 ns,    --rust -> wekkertijd
46      "0000" after 2568 ns,    --knoppen(3) = down
47      "0010" after 2768 ns,    --wekkertijd -> uren_set
48      "0000" after 2808 ns,    --knoppen(3) = down
49      "0010" after 2968 ns,    --uren_set -> minuten_set
50      "0000" after 3048 ns,    --knoppen(3) = down
51      "0100" after 3208 ns,    --minuten_set -> minuten_plus
52      "0000" after 3288 ns,    --minuten_plus -> minuten_set
53      "1000" after 3448 ns,    --minuten_set -> minuten_min
54      "0000" after 3528 ns,    --minuten_min -> minuten_set
55      "0001" after 3688 ns,    --minuten_set -> rust
56      "0000" after 3768 ns,    --knoppen(3) = down
57      "0001" after 3928 ns,    --rust -> wekkertijd
58      "0000" after 4008 ns,    --knoppen(3) = down
59      "0010" after 4168 ns,    --wekkertijd -> uren_set
60      "0000" after 4248 ns,    --knoppen(3) = down
61      "0010" after 4408 ns,    --uren_set -> minuten_set
62      "0000" after 4488 ns,    --knoppen(3) = down
63      "0010" after 4648 ns,    --minuten_set -> rust
64      "0000" after 4728 ns,    --knoppen(3) = down
65      "0001" after 4888 ns,    --rust -> wekkertijd
66      "0000" after 4968 ns,    --knoppen(3) = down
67      "1000" after 5128 ns,    --wekkertijd -> led
68      "0000" after 5208 ns,    --knoppen(3) = down
69      "0001" after 5368 ns,    --led -> rust
70      "0000" after 5448 ns,    --knoppen(3) = down
71      "0001" after 5608 ns,    --rust -> wekkertijd
72      "0000" after 5688 ns,    --knoppen(3) = down
73      "0100" after 5848 ns,    --wekkertijd -> geluid
74      "0000" after 6128 ns,    --knoppen(3) = down
75      "0100" after 6288 ns,    --geluid -> led
76      "0000" after 6368 ns,    --knoppen(3) = down
77      "0100" after 6528 ns,    --led -> wekkertijd
78      "0000" after 6608 ns,    --knoppen(3) = down
79      "1000" after 6768 ns,    --wekkertijd -> led
80      "0000" after 6848 ns,    --knoppen(3) = down
81      "1000" after 7008 ns,    --led -> geluid
82      "0000" after 7088 ns,    --knoppen(3) = down
83      "1000" after 7248 ns,    --geluid -> wekkertijd
84      "0000" after 7328 ns,    --knoppen(3) = down
85      "1000" after 7488 ns,    --wekkertijd -> led
86      "0000" after 7568 ns,    --knoppen(3) = down
87      "0010" after 7728 ns,    --led -> led_toggle
88      "0000" after 7808 ns,    --led_toggle -> led
89      "1000" after 7968 ns,    --led -> geluid
90      "0000" after 8048 ns,    --knoppen(3) = down
91      "0010" after 8208 ns,    --geluid -> geluid_toggle
92      "0000" after 8288 ns,    --geluid_toggle -> geluid
93      "0001" after 8448 ns,    --geluid -> rust
94      "0000" after 8528 ns;   --done, done, done;
95
96      controller_pm: controller port map(clk, reset, knoppen, wekker,menu_signal);
97 end architecture;

```

## B.5.2. TESTBENCH VHDL MENU

```

1  -- Kevin Hill 4287592
2  -- De testbench voor de menu loopt alle states door
3
4  library IEEE;s

```

```

5  use IEEE.std_logic_1164.ALL;
6  use IEEE.Numeric_Std.all;
7
8  architecture behaviour of menu_test is
9  component menu is          --component initialiseren, met de volgende in/uitgangen:
10   port(clk      :in      std_logic;
11       reset     :in      std_logic;
12       knoppen   :in      std_logic_vector (3 downto 0);    --dit zijn de fysieke
13           knoppen
14       wekdata   :in      std_logic_vector (15 downto 0);    --komt bij het
15           register vandaan
16       enable    :out     std_logic;
17       wekker    :out     std_logic_vector (15 downto 0);
18       menu_signal :out    std_logic_vector (2 downto 0)); --voor de LCD'
19 end component menu;
20
21 signal clk, reset, enable   :  std_logic;
22 signal menu_signal         :  std_logic_vector (2 downto 0);
23 signal knoppen,minuten_enkel,uren_enkel      :  std_logic_vector (3 downto 0);
24           --signalen voor de port map
25 signal wekdata, wekker      :  std_logic_vector (15 downto 0);
26 --signal uren            :  std_logic_vector (5 downto 0);
27 --signal minute          :  std_logic_vector (6 downto 0);
28 signal uren_dubble        :  std_logic_vector (1 downto 0);
29 signal minuten_duble      :  std_logic_vector (2 downto 0);
30
31 begin
32   clk      <=  '1' after 0 ns,
33           '0' after 20 ns when clk /= '0' else '1' after 20 ns;
34
35   reset    <=  '1' after 0 ns,           --knoppen(0) = menu;
36           '0' after 62 ns;           --knoppen(1) = set;
37           --knoppen(2) = up;
38
39   knoppen <= "0000" after 0 ns, --knoppen(3) = down
40           "0010" after 68 ns, --rust -> wekker_toggle
41           "0010" after 108 ns, --wekker_toggle -> rust
42           "0001" after 148 ns, --rust -> wekkertijd
43           "0001" after 188 ns, --wekkertijd -> rust
44           "0001" after 228 ns, --rust -> wekkertijd
45           "0010" after 268 ns, --wekkertijd -> uren_set
46           "0100" after 308 ns, --uren_set -> uren_plus
47           "0000" after 348 ns, --uren_plus -> uren_set
48           "1000" after 388 ns, --uren_set -> uren_min
49           "0000" after 428 ns, --uren_min -> uren_set
50           "0001" after 468 ns, --uren_set -> rust
51           "0001" after 508 ns, --rust -> wekkertijd
52           "0010" after 548 ns, --wekkertijd -> uren_set
53           "0010" after 588 ns, --uren_set -> minuten_set
54           "0100" after 628 ns, --minuten_set -> minuten_plus
55           "0000" after 668 ns, --minuten_plus -> minuten_set
56           "1000" after 708 ns, --minuten_set -> minuten_min
57           "0000" after 748 ns, --minuten_min -> minuten_set
58           "0001" after 788 ns, --minuten_set -> rust
59           "0001" after 828 ns, --rust -> wekkertijd
60           "0010" after 868 ns, --wekkertijd -> uren_set
61           "0010" after 908 ns, --uren_set -> minuten_set
62           "0010" after 948 ns, --minuten_set -> wekkertijd EIGENLIJK GAAT DIT NAAR RUST TOE
63           "0001" after 988 ns, --rust -> wekkertijd
64           "1000" after 1028 ns, --wekkertijd -> led
65           "0001" after 1068 ns, --led -> rust
66           "0001" after 1108 ns, --rust -> wekkertijd
67           "0100" after 1148 ns, --wekkertijd -> geluid
68           "0100" after 1188 ns, --geluid -> led
69           "0100" after 1228 ns, --led -> wekkertijd
70           "1000" after 1268 ns, --wekkertijd -> led
71           "1000" after 1308 ns, --led -> geluid
72           "1000" after 1348 ns, --geluid -> wekkertijd
73           "1000" after 1388 ns, --wekkertijd -> led
74           "0010" after 1428 ns, --led -> led_toggle
75           "0000" after 1468 ns, --led_toggle -> led

```

```

73      "1000" after 1508 ns, --led -> geluid
74      "0010" after 1548 ns, --geluid -> geluid_toggle
75      "0000" after 1588 ns, --geluid_toggle -> geluid
76      "0001" after 1628 ns, --geluid -> rust
77      "0000" after 1668 ns; --done, done, done;
78
79
80 uren <= wekker(12 downto 7);
81 minuten <= wekker(6 downto 0);
82 wekdata <= "000010001000000" after 20 ns;
83
84
85      menu_pm: menu port map(clk, reset, knoppen, wekdata, enable, wekker, menu_signal); --de
86      daadwerkelijke port map
86 end architecture;

```

### B.5.3. TESTBENCH VHDL GEHEUGEN

```

1  -- Rens Hamburger 4292936
2  -- Testbench voor de 16 bit geheugen element met een positief enable signaal
3  library IEEE;
4  use IEEE.std_logic_1164.ALL;
5
6  architecture behaviour of geheugen_tb is
7  component geheugen is
8      port(clk      :in    std_logic;
9          reset   :in    std_logic;
10         enable  :in    std_logic;
11         wek_in :in    std_logic_vector(15 downto 0);
12         wek_out:out   std_logic_vector(15 downto 0));
13 end component geheugen;
14
15 signal clk,enable,reset      :    std_logic;
16 signal wek_in,wek_out       :    std_logic_vector(15 downto 0);
17
18
19 begin
20     clk      <= '0' after 0 ns,
21             '1' after 20 ns when clk /= '1' else '0' after 20 ns;
22
23     reset    <= '1' after 0 ns,
24             '0' after 85 ns;
25
26
27     enable <= '0' after 0 ns,
28             '1' after 150 ns,
29             '0' after 290 ns,
30             '1' after 590 ns;
31
32     wek_in <= "0000000000000001" after 0 ns,
33             "0000000000000010" after 70 ns,
34             "0000000000000011" after 110 ns,
35             "0000000000000100" after 150 ns,
36             "0000000000000101" after 190 ns,
37             "0000000000000110" after 230 ns,
38             "0000000000000111" after 270 ns,
39             "0000000000001000" after 310 ns,
40             "0000000000001001" after 350 ns,
41             "0000000000001010" after 390 ns,
42             "0000000000001011" after 430 ns,
43             "0000000000001100" after 470 ns,
44             "0000000000001101" after 510 ns,
45             "0000000000001110" after 550 ns,
46             "0000000000001111" after 590 ns,
47             "00000000000010000" after 630 ns,
48             "00000000000010001" after 680 ns,
49             "00000000000010010" after 735 ns,
50             "00000000000010111" after 779 ns;
51
52     geheugen_pm: geheugen port map(clk,reset,enable,wek_in,wek_out);
53 end behaviour;

```

### B.5.4. TESTBENCH VHDL BUFFER

```

1 -- Rens Hamburger 4292936
2 -- Testbench om de buffer te testen
3 library IEEE;
4 use IEEE.std_logic_1164.ALL;
5
6 architecture behaviour of buff_tb is
7 component buff is
8     port(clk           :in  std_logic;
9           reset        :in  std_logic;
10          knoppen_in   :in  std_logic_vector(3 downto 0);
11          knoppen_out  :out  std_logic_vector(3 downto 0));
12 end component buff;
13
14 signal clk,enable,reset      :  std_logic;
15 signal knoppen,knoppjes      :  std_logic_vector(3 downto 0);
16
17
18 begin
19     clk      <= '0' after 0 ns,
20             '1' after 20 ns when clk /= '1' else '0' after 20 ns;
21
22     reset    <= '1' after 0 ns,
23             '0' after 85 ns;
24
25     knoppen <= "0000" after 0 ns,
26                 "1111" after 100 ns,
27                 "0000" after 150 ns,
28                 "1000" after 190 ns,
29                 "0000" after 240 ns,
30                 "0001" after 290 ns;
31
32     buff_pm: buff port map(clk,reset,knoppen,knoppjes);
33 end behaviour;
```

## B.6. VHDL CODE VAN HET ALARM

### B.6.1. ENTITY ALARM-COMPARE

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity compare is
5     port(clk       :in  std_logic;
6           reset     :in  std_logic;
7           tijd_uur  :in  std_logic_vector(4 downto 0);
8           tijd_min  :in  std_logic_vector(5 downto 0);
9           wekker_uur:in  std_logic_vector(4 downto 0);
10          wekker_min:in  std_logic_vector(5 downto 0);
11          stop_alarm:in  std_logic;
12          geluid    :out  std_logic;
13          licht     :out  std_logic);
14 end compare;
```

### B.6.2. BEHAVIOURAL ALARM-COMPARE

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3 use IEEE.numeric_std.ALL;
4
5 architecture behaviour of compare is
6     type comp_state is (steady, start, final);
7     signal state, new_state: comp_state;
8     signal alarm_uur: std_logic_vector(4 downto 0);
9     signal alarm_min: std_logic_vector(5 downto 0);
10
11 begin
12     l1l1: process (clk)
13     begin
14         if (clk'event and clk = '1') then
15             if (reset = '1') or (stop_alarm = '1') then
```

```

15      state <= steady;
16      alarm_min <= std_logic_vector(to_unsigned(0, 6));
17      alarm_uur <= std_logic_vector(to_unsigned(0,5));
18  else
19      if (to_integer(unsigned(wekker_min)) > 14) then
20          alarm_min <= std_logic_vector(to_unsigned(to_integer(unsigned(wekker_min)
21                                         ) - 15, 6));
22          alarm_uur <= wekker_uur;
23      else
24          alarm_min <= std_logic_vector(to_unsigned(60 - (15-to_integer(unsigned(
25                                         wekker_min))),6));
26          if (to_integer(unsigned(wekker_uur)) = 0) then
27              alarm_uur <= std_logic_vector(to_unsigned(23, 5));
28          else
29              alarm_uur <= std_logic_vector(to_unsigned(to_integer(unsigned(
30                                         wekker_uur)) - 1, 5));
31          end if;
32      end if;
33  end process;
34 begin
35     lbl2: process (state, alarm_min, alarm_uur, wekker_uur, wekker_min, tijd_min, tijd_uur)
36 begin
37     case state is
38         when steady =>
39             geluid <= '0';
40             licht <= '0';
41             if (alarm_min = tijd_min) and (alarm_uur = tijd_uur) then
42                 new_state <= start;
43             else
44                 new_state <= steady;
45             end if;
46         when start =>
47             geluid <= '0';
48             licht <= '1';
49             if (wekker_uur = tijd_uur) and (wekker_min = tijd_min) then
50                 new_state <= final;
51             else
52                 new_state <= start;
53             end if;
54         when final =>
55             geluid <= '1';
56             licht <= '1';
57             new_state <= final;
58         when others =>
59             geluid <= '0';
60             licht <= '1';
61             new_state <= state;
62     end case;
63 end process;
64 end behaviour;

```

### B.6.3. TOP ENTITY ALARM

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity alarm is
5     port(clk      :in  std_logic;
6           reset    :in  std_logic;
7           sec      :in  std_logic;
8           licht    :in  std_logic;
9           pwm_signal:out std_logic);
10 end alarm;

```

### B.6.4. BEHAVIOURAL ALARM

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3

```

```

4  architecture behaviour of alarm is
5  component counter
6    port( clk   :in  std_logic;
7          reset :in  std_logic;
8          sec   :in  std_logic;
9          licht :in  std_logic;
10         length:out std_logic_vector(5 downto 0));
11 end component;
12 component pwm
13   port( clk   :in  std_logic;
14         reset :in  std_logic;
15         length:in  std_logic_vector(5 downto 0);
16         pwm_signal :out  std_logic);
17 end component;
18 signal length : std_logic_vector (5 downto 0);
19 begin
20   counter_1 : counter port map (clk => clk, reset => reset, sec => sec, licht => licht,
21                                     length => length);
22   pwm_1 : pwm port map (clk => clk, reset => reset, length => length, pwm_signal =>
23                         pwm_signal);
22 end behaviour;

```

### B.6.5. ENTITY ALARM-COUNTER

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity counter is
5   port(clk   :in  std_logic;
6        reset :in  std_logic;
7        sec   :in  std_logic;
8        licht :in  std_logic;
9        length:out std_logic_vector(5 downto 0));
10 end counter;

```

### B.6.6. BEHAVIOURAL ALARM-COUNTER

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3 use IEEE.numeric_std.ALL;
4
5 architecture behaviour of counter is
6   type counter_state is (init, laag, hoog);
7   signal count, new_count: unsigned(3 downto 0);
8   signal length2, new_length2: unsigned(5 downto 0);
9   signal state, new_state: counter_state;
10 begin
11   length <= std_logic_vector(new_length2);
12   lbl1: process(clk)
13   begin
14     if (clk'event and clk = '1') then
15       if (reset = '1') or (licht = '0') then
16         state <= init;
17         count <= (others => '0');
18       else
19         state <= new_state;
20         count <= new_count;
21       end if;
22       length2 <= new_length2;
23     end if;
24   end process;
25   lbl2: process(sec, count, length2)
26   begin
27     case state is
28       when init =>
29         new_length2 <= (others => '1');
30         new_count <= (others => '0');
31         new_state <= laag;
32       when laag =>
33         if (sec = '1') then
34           if (count = "1111") then

```

```

35          new_count <= "0001";
36          if (length2 /= 0) then
37              new_length2 <= length2 -1;
38          else
39              new_length2 <= length2;
40          end if;
41      else
42          new_count <= count + 1;
43          new_length2 <= length2;
44      end if;
45      new_state <= hoog;
46  else
47      new_count <= count;
48      new_length2 <= length2;
49      new_state <= laag;
50  end if;
51 when hoog =>
52     if (sec = '0') then
53         new_state <= laag;
54     else
55         new_state <= hoog;
56     end if;
57     new_count <= count;
58     new_length2 <= length2;
59 when others =>
60     new_count <= count;
61     new_length2 <= length2;
62     new_state <= hoog;
63 end case;
64 end process;
65 end behaviour;

```

### B.6.7. ENTITY ALARM-PWM

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity pwm is
5     port(clk    :in  std_logic;
6           reset :in  std_logic;
7           length:in  std_logic_vector(5 downto 0);
8           pwm_signal :out std_logic);
9 end pwm;

```

### B.6.8. BEHAVIOURAL ALARM-PWM

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3 use IEEE.numeric_std.ALL;
4
5 architecture behaviour of pwm is
6     type pwm_state is (hoog, laag, res_state);
7     signal counter, new_counter: unsigned(5 downto 0);
8     signal state, new_state: pwm_state;
9 begin
10     lbl1: process(clk)
11     begin
12         if(clk'event and clk = '1') then
13             if (reset = '1') then
14                 state <= res_state;
15                 counter <= (others => '0');
16             else
17                 state <= new_state;
18                 counter <= new_counter;
19             end if;
20         end if;
21     end process;
22     lbl2: process(counter, length, state)
23     begin
24         case state is
25             when res_state =>

```

```

26          pwm_signal <= '0';
27          new_counter <= (others => '0');
28          new_state <= laag;
29      when laag =>
30          pwm_signal <= '0';
31          new_counter <= counter + 1;
32          if (unsigned(length) <= counter) then
33              new_state <= hoog;
34          else
35              new_state <= laag;
36          end if;
37      when hoog =>
38          pwm_signal <= '1';
39          new_counter <= counter + 1;
40          if (unsigned(length) <= counter) then
41              new_state <= hoog;
42          else
43              new_state <= laag;
44          end if;
45      when others =>
46          pwm_signal <= '0';
47          new_counter <= counter;
48          new_state <= laag;
49      end case;
50  end process;
51 end behaviour;

```

## B.7. VHDL CODE LCD AANSTURING

### B.7.1. ENTITY TIJD

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity tijd is
5     port(clk :in  std_logic;
6           reset:in  std_logic;
7           uren: in std_logic_vector(5 downto 0);
8           minuten : in std_logic_vector(6 downto 0);
9           x      :out  std_logic_vector(6 downto 0);
10          y      :out  std_logic_vector(5 downto 0);
11          c      :out  std_logic_vector(6 downto 0);
12          ready:in  std_logic;
13          hz_sig:in std_logic
14      );
15 end tijd;

```

### B.7.2. BEHAVIOURAL TIJD

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3 use IEEE.std_logic_arith.all;
4
5 architecture behaviour of tijd is
--Constants declaration
7 constant dubbele_punt_x : std_logic_vector(6 downto 0) := "0110101";
8 constant dubbele_punt_y : std_logic_vector(5 downto 0) := "000110";
9 constant dubbele_punt_c_aan : std_logic_vector(6 downto 0) := "00000001";
10 constant dubbele_punt_c_uit : std_logic_vector(6 downto 0) := "00000010";
11
12 --Posities
13 constant char_0_x : std_logic_vector(6 downto 0) := "0010101";
14 constant char_0_y : std_logic_vector(5 downto 0) := "000011";
15
16 constant char_1_x : std_logic_vector(6 downto 0) := "0100101";
17 constant char_1_y : std_logic_vector(5 downto 0) := "000011";
18
19 constant char_2_x : std_logic_vector(6 downto 0) := "0111000";
20 constant char_2_y : std_logic_vector(5 downto 0) := "000011";
21
22 constant char_3_x : std_logic_vector(6 downto 0) := "1001000";

```

```

23 constant char_3_y : std_logic_vector(5 downto 0) := "000011";
24
25 --Characters
26 constant char_0 : std_logic_vector(6 downto 0) := "0001100";
27 constant char_1 : std_logic_vector(6 downto 0) := "0000011";
28 constant char_2 : std_logic_vector(6 downto 0) := "0000100";
29 constant char_3 : std_logic_vector(6 downto 0) := "0000101";
30 constant char_4 : std_logic_vector(6 downto 0) := "0000110";
31 constant char_5 : std_logic_vector(6 downto 0) := "0000111";
32 constant char_6 : std_logic_vector(6 downto 0) := "0001000";
33 constant char_7 : std_logic_vector(6 downto 0) := "0001001";
34 constant char_8 : std_logic_vector(6 downto 0) := "0001010";
35 constant char_9 : std_logic_vector(6 downto 0) := "0001011";
36
37 --Signals
38 type states is (rust, char_0_state,char_1_state, char_2_state, char_3_state, dubbele_punt);
39 signal state, new_state : states;
40 signal hz_state, new_hz_state : std_logic;
41 signal ready_sig, new_ready_sig : std_logic;
42 --signal minuten_tijd, new_minuten_tijd : unsigned(5 downto 0);
43 --signal uren_tijd, new_uren_tijd : unsigned(4 downto 0);
44 --signal minner : unsigned(5 downto 0);
45 signal lsb_minuten, new_lsb_minuten : std_logic;
46 signal punt, new_punt: std_logic;
47
48 begin
49 process(clk, state, new_state, reset)--process om van state te veranderen en voor de reset
50 begin
51
52     if(rising_edge(clk)) then
53         if(reset = '1') then
54             state <= rust;
55             hz_state <= '0';
56             ready_sig <= '0';
57             punt <= '0';
58             lsb_minuten <= '0';
59         else
60             state <= new_state;
61             lsb_minuten <= new_lsb_minuten;
62             hz_state <= new_hz_state;
63             punt <= new_punt;
64             ready_sig <= new_ready_sig;
65         end if;
66     end if;
67 end process;
68
69 process(ready, hz_sig, uren, minuten, state)--process voor rekenen
70 begin
71     case(state) is
72         when rust =>
73             x <= "0000000";
74             y <= "000000";
75             c <= "0000000";
76             new_ready_sig <= ready;
77             new_hz_state <= hz_sig;
78             new_punt <= punt;
79             if (minuten(0) /= lsb_minuten) then
80                 new_lsb_minuten <= minuten(0);
81                 new_state <= char_0_state;
82             elsif(hz_state /= hz_sig) then
83                 new_lsb_minuten <= minuten(0);
84                 if(hz_sig = '1') then
85                     new_state <= dubbele_punt;
86                 else
87                     new_state <= rust;
88                 end if;
89             else
90                 new_lsb_minuten <= minuten(0);
91                 new_state <= rust;
92             end if;
93         when dubbele_punt =>

```

```

94      --new_minuten_tijd <= minuten_tijd;
95      --new_uren_tijd <= uren_tijd;
96      new_ready_sig <= ready;
97      new_lsb_minuten <= lsb_minuten;
98      new_hz_state <= hz_sig;
99      --minner <= "000000";
100     if(punt = '1') then
101         x <= dubbele_punt_x;
102         y <= dubbele_punt_y;
103         c <= dubbele_punt_c_aan;
104         if(ready = '0') then
105             if(ready_sig = '1') then
106                 --ready_sig <= '0';
107                 new_punt <= '0';
108                 new_state <= rust;
109             else
110                 --ready_sig <= ready;
111                 new_punt <= punt;
112                 new_state <= dubbele_punt;
113             end if;
114         else
115             new_punt <= punt;
116             new_state <= dubbele_punt;
117             --ready_sig <= ready;
118         end if;
119     else
120         x <= dubbele_punt_x;
121         y <= dubbele_punt_y;
122         c <= dubbele_punt_c_uit;
123         if(ready = '0') then
124             if(ready_sig = '1') then
125                 --ready_sig<= '0';
126                 new_punt <= '1';
127                 new_state <= rust;
128             else
129                 --ready_sig <= ready;
130                 new_punt <= punt;
131                 new_state <= dubbele_punt;
132             end if;
133         else
134             new_punt <= punt;
135             new_state <= dubbele_punt;
136             --ready_sig <= ready;
137         end if;
138     end if;
139     when char_0_state =>
140         new_lsb_minuten <= lsb_minuten;
141         new_ready_sig <= ready;
142         new_hz_state <= hz_sig;
143         new_punt <= punt;
144         x <= char_0_x;
145         y <= char_0_y;
146         case(uren(5 downto 4)) is
147             when "00" =>
148                 c <= char_0;
149             when "01" =>
150                 c <= char_1;
151             when "10" =>
152                 c <= char_2;
153             when others =>
154                 c <= char_0;
155         end case;
156         if(ready = '0') then
157             if(ready_sig = '1') then
158                 new_state <= char_1_state;
159             else
160                 new_state <= char_0_state;
161             end if;
162         else
163             new_state <= char_0_state;
164         end if;

```

```

165      when char_1_state =>
166          new_lsb_minuten <= lsb_minuten;
167          new_ready_sig <= ready;
168          new_hz_state <= hz_sig;
169          x <= char_1_x;
170          y <= char_1_y;
171          new_punt <= punt;
172      if(ready = '0') then
173          if(ready_sig = '1') then
174              new_state <= char_2_state;
175          else
176              new_state <= char_1_state;
177          end if;
178      else
179          new_state <= char_1_state;
180      end if;
181      case (uren(3 downto 0)) is
182          when "0001" =>
183              c <= char_1;
184          when "0010" =>
185              c <= char_2;
186          when "0011" =>
187              c <= char_3;
188          when "0100" =>
189              c <= char_4;
190          when "0101" =>
191              c <= char_5;
192          when "0110" =>
193              c <= char_6;
194          when "0111" =>
195              c <= char_7;
196          when "1000" =>
197              c <= char_8;
198          when "1001" =>
199              c <= char_9;
200          when others =>
201              c <= char_0;
202      end case;
203      when char_2_state =>
204          new_hz_state <= hz_sig;
205          new_lsb_minuten <= lsb_minuten;
206          new_ready_sig <= ready;
207          x <= char_2_x;
208          y <= char_2_y;
209          new_punt <= punt;
210      case(minuten(6 downto 4)) is
211          when "000" =>
212              c <= char_0;
213          when "001" =>
214              c <= char_1;
215          when "010" =>
216              c <= char_2;
217          when "011" =>
218              c <= char_3;
219          when "100" =>
220              c <= char_4;
221          when "101" =>
222              c <= char_5;
223          when others =>
224              c <= char_0;
225      end case;
226      if(ready = '0') then
227          if(ready_sig = '1') then
228              new_state <= char_3_state;
229          else
230              new_state <= char_2_state;
231          end if;
232      else
233          new_state <= char_2_state;
234      end if;
235

```

```

236      when char_3_state =>
237          new_lsb_minuten <= lsb_minuten;
238          new_hz_state <= hz_sig;
239          new_ready_sig <= ready;
240          x <= char_3_x;
241          y <= char_3_y;
242          new_punt <= punt;
243      case(minuten(3 downto 0)) is
244          when "0001" =>
245              c <= char_1;
246          when "0010" =>
247              c <= char_2;
248          when "0011" =>
249              c <= char_3;
250          when "0100" =>
251              c <= char_4;
252          when "0101" =>
253              c <= char_5;
254          when "0110" =>
255              c <= char_6;
256          when "0111" =>
257              c <= char_7;
258          when "1000" =>
259              c <= char_8;
260          when "1001" =>
261              c <= char_9;
262          when others =>
263              c <= char_0;
264      end case;
265      if(ready = '0') then
266          if(ready_sig = '1') then
267              new_state <= rust;
268          else
269              new_state <= char_3_state;
270          end if;
271      else
272          new_state <= char_3_state;
273      end if;
274  when others =>
275      new_lsb_minuten <= lsb_minuten;
276      new_hz_state <= hz_sig;
277      new_ready_sig <= ready;
278      new_state <= rust;
279      new_punt <= punt;
280      x <= "0000000";
281      y <= "000000";
282      c <= "0000000";
283  end case;
284 end process;
285 end behaviour;

```

### B.7.3. ENTITY DATUM

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity datum is
5     port(
6         clk           : in std_logic;
7         reset         : in std_logic;
8         ready         : in std_logic;
9         tijd_uren    : in std_logic_vector (5 downto 0);
10        dagvdweek   : in std_logic_vector (2 downto 0);
11        dagvdmaand  : in std_logic_vector (5 downto 0);
12        maand        : in std_logic_vector (4 downto 0);
13        jaar         : in std_logic_vector (7 downto 0);
14
15        x            : out std_logic_vector (6 downto 0);
16        y            : out std_logic_vector (5 downto 0);
17        c            : out std_logic_vector (6 downto 0)
18    );

```

```
19 end datum;
```

#### B.7.4. BEHAVIOURAL DATUM

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4
5 architecture behaviour of datum is
6
7 --component f_edge_detect is
8 --  port (
9 --    clk      : in std_logic;
10 --   signal_in : in std_logic;
11 --   signal_out : out std_logic
12 --  );
13 --end component;
14
15 --Constantes voor posities X-Y
16 constant x_rust : std_logic_vector (6 downto 0) := "0000000";
17 constant y_rust : std_logic_vector (5 downto 0) := "00000";
18
19 constant y_1      : std_logic_vector (5 downto 0) := "011011";
20
21 constant x_1 : std_logic_vector (6 downto 0) := "0010101";
22 constant x_2 : std_logic_vector (6 downto 0) := "0101100";
23 constant x_3 : std_logic_vector (6 downto 0) := "0110010";
24 constant x_4 : std_logic_vector (6 downto 0) := "0111100";
25 constant x_5 : std_logic_vector (6 downto 0) := "1000010";
26 constant x_6 : std_logic_vector (6 downto 0) := "1001100";
27 constant x_7 : std_logic_vector (6 downto 0) := "1010010";
28
29 -- Constantes voor getallen 0 tot 9
30 constant char_rust : std_logic_vector(6 downto 0) := "0000000";
31 constant char_0 : std_logic_vector(6 downto 0) := "0010111";
32 constant char_1 : std_logic_vector(6 downto 0) := "0001110";
33 constant char_2 : std_logic_vector(6 downto 0) := "0001111";
34 constant char_3 : std_logic_vector(6 downto 0) := "0010000";
35 constant char_4 : std_logic_vector(6 downto 0) := "0010001";
36 constant char_5 : std_logic_vector(6 downto 0) := "0010010";
37 constant char_6 : std_logic_vector(6 downto 0) := "0010011";
38 constant char_7 : std_logic_vector(6 downto 0) := "0010100";
39 constant char_8 : std_logic_vector(6 downto 0) := "0010101";
40 constant char_9 : std_logic_vector(6 downto 0) := "0010110";
41
42 --Constantes voor de dagen MA t/m ZO
43 constant char_ma : std_logic_vector(6 downto 0) := "0011000";
44 constant char_di : std_logic_vector(6 downto 0) := "0011001";
45 constant char_wo : std_logic_vector(6 downto 0) := "0011010";
46 constant char_do : std_logic_vector(6 downto 0) := "0011011";
47 constant char_vr : std_logic_vector(6 downto 0) := "0011100";
48 constant char_zu : std_logic_vector(6 downto 0) := "0011101";
49 constant char_zo : std_logic_vector(6 downto 0) := "0011110";
50
51
52
53
54 type states is (rust, selectdata, cdvdw, cgetal);
55
56 signal state, next_state      : states;
57 signal positie, new_positie   : unsigned ( 2 downto 0);
58 signal data_buffer            : std_logic_vector(3 downto 0);
59 signal start, finish, ready_buf1, ready_buf2 : std_logic;
60
61
62 begin
63 --proces FSM
64
65 process (clk, reset, ready_buf1, new_positie, next_state)
66 begin
67     if (rising_edge (clk)) then

```

```

68         ready_buf2<=ready_buf1;
69
70     if (reset='1') then
71         positie <= (others => '0');
72         state<= selectdata;
73     else
74         state<=next_state;
75         positie <= new_positie;
76     end if;
77 end if;
78 end process;
79
80 process (state, ready_buf2, ready, tijd_uren, positie, data_buffer)
81 begin
82     case(state) is
83
84         when rust =>
85             ready_buf1<=ready;
86             c <= char_rust;
87             new_positie <= (others => '0');
88
89         if (tijd_uren = "000000" ) then
90             next_state <= selectdata;
91         else
92             next_state <= rust;
93         end if;
94
95         when sreset =>
96             c <= char_rust;
97
98         new_positie <= (others => '0');
99         --indien er is gereset, of de uren zijn veranderd naar "00000", start
100        verzenden
101        if (reset_buffer = '1') then
102            next_state <= selectdata;
103        elsif (start = '1') then
104            next_state <= selectdata;
105        else
106            next_state <= sreset;
107        end if;
108
109        when selectdata =>
110            ready_buf1<=ready;
111            c <= char_rust; -- maak uitgang C in rust stand,
112            new_positie <= positie;
113
114            -- indien positie tot 7 is geteld, ga naar sreset (rust) state.
115            if (positie = 7 ) then
116                if (tijd_uren = "000000") then
117                    next_state <= selectdata;
118                else
119                    next_state <= rust;
120                end if;
121            -- indien positie is 0, ga character voor dag van de week schrijven
122            elsif (positie = 0) then
123                next_state<= cdvwd;
124                -- in alle andere gevallen, ga character(s) voor de datum schrijven
125            else
126                next_state<= cgetal;
127            end if;
128
129            -- state om character voor dag van de week te schrijven
130            when cdvwd =>
131                ready_buf1<=ready;
132                case (data_buffer) is
133                    when "0001" =>
134                        c <= char_ma;
135                    when "0010" =>
136                        c <= char_di;
137                    when "0011" =>

```

```
138          c <= char_wo;
139      when "0100" =>
140          c <= char_do;
141      when "0101" =>
142          c <= char_vr;
143      when "0110" =>
144          c <= char_za;
145      when "0111" =>
146          c <= char_zo;
147      when others =>
148          c <= char_rust;
149  end case;
150
151 -- indien de slave-select laag wordt, dan positie+1 en terug naar state
152 select data
153 if (ready_buf2 = '1') then
154     if(ready = '0') then
155         new_positie <= positie + 1;
156         next_state <= selectdata;
157     else
158         next_state <= cdvwd;
159         new_positie <= positie;
160     end if;
161 else
162     next_state <= cgetal;
163     new_positie <= positie;
164 end if;
165
166 -- state voor character om getallen te schrijven voor datum
167 when cgetal =>
168     ready_buf1<=ready;
169     case (data_buffer) is
170         when "0000" =>
171             c <= char_0;
172         when "0001" =>
173             c <= char_1;
174         when "0010" =>
175             c <= char_2;
176         when "0011" =>
177             c <= char_3;
178         when "0100" =>
179             c <= char_4;
180         when "0101" =>
181             c <= char_5;
182         when "0110" =>
183             c <= char_6;
184         when "0111" =>
185             c <= char_7;
186         when "1000" =>
187             c <= char_8;
188         when "1001" =>
189             c <= char_9;
190         when others =>
191             c <= char_rust;
192     end case;
193
194     if (ready_buf2 = '1') then
195         if(ready = '0') then
196             new_positie <= positie + 1;
197             next_state <= selectdata;
198         else
199             next_state <= cgetal;
200             new_positie <= positie;
201         end if;
202     else
203         next_state <= cgetal;
204         new_positie <= positie;
205     end if;
206 end case;
```

```

208      end process;
209
210  process (positie, dagvdmaand, maand, jaar, dagvdweek)
211 begin
212     case positie is
213
214        when "001" =>
215            data_buffer(0) <= dagvdmaand(4);
216            data_buffer(1) <= dagvdmaand(5);
217            data_buffer(2) <= '0';
218            data_buffer(3) <= '0';
219            x <= x_2;
220            y <= y_1;
221        when "010" =>
222            data_buffer(0) <= dagvdmaand(0);
223            data_buffer(1) <= dagvdmaand(1);
224            data_buffer(2) <= dagvdmaand(2);
225            data_buffer(3) <= dagvdmaand(3);
226            x <= x_3;
227            y <= y_1;
228        when "011" =>
229            data_buffer(0) <= maand(4);
230            data_buffer(1) <= '0';
231            data_buffer(2) <= '0';
232            data_buffer(3) <= '0';
233            x <= x_4;
234            y <= y_1;
235
236        when "100" =>
237            data_buffer(0) <= maand(0);
238            data_buffer(1) <= maand(1);
239            data_buffer(2) <= maand(2);
240            data_buffer(3) <= maand(3);
241            x <= x_5;
242            y <= y_1;
243        when "101" =>
244            data_buffer(0) <= jaar(4);
245            data_buffer(1) <= jaar(5);
246            data_buffer(2) <= jaar(6);
247            data_buffer(3) <= jaar(7);
248            x <= x_6;
249            y <= y_1;
250        when "110" =>
251            data_buffer(0) <= jaar(0);
252            data_buffer(1) <= jaar(1);
253            data_buffer(2) <= jaar(2);
254            data_buffer(3) <= jaar(3);
255            x <= x_7;
256            y <= y_1;
257
258        when "000" =>
259            data_buffer(0) <= dagvdweek(0);
260            data_buffer(1) <= dagvdweek(1);
261            data_buffer(2) <= dagvdweek(2);
262            data_buffer(3) <= '0';
263            x <= x_1;
264            y <= y_1;
265        when others =>
266            data_buffer(0) <= '0';
267            data_buffer(1) <= '0';
268            data_buffer(2) <= '0';
269            data_buffer(3) <= '0';
270            x <= x_rust;
271            y <= y_rust;
272     end case;
273 end process;
274
275
276 end behaviour;

```

### B.7.5. ENTITY DCF LCD

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity dcf_lcd is
5   port(clk      :in  std_logic;
6        reset     :in  std_logic;
7        ready     :in  std_logic;
8        dcf_debug:in  std_logic;
9        x         :out std_logic_vector(6 downto 0);
10       y         :out std_logic_vector(5 downto 0);
11       c         :out std_logic_vector(6 downto 0));
12 end dcf_lcd;

```

### B.7.6. BEHAVIOURAL DCF LCD

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 architecture behaviour of dcf_lcd is
5 type dcf_state is (steady, aan, aan_ready, uit, uit_ready);
6 signal state, new_state: dcf_state;
7 signal buf: std_logic;
8
9 constant x_pos: std_logic_vector (6 downto 0) := "0000001";
10 constant y_pos: std_logic_vector (5 downto 0) := "000001";
11 constant c_aan: std_logic_vector (6 downto 0) := "0100100";
12 constant c_uit: std_logic_vector (6 downto 0) := "0100101";
13
14 begin
15
16 lbl1: process(clk)
17 begin
18   if (clk'event and clk='1') then
19     if reset = '1' then
20       state <= steady;
21       buf <= dcf_debug;
22     else
23       state <= new_state;
24       buf <= dcf_debug;
25     end if;
26   end if;
27 end process;
28
29 lbl2: process (state, dcf_debug, ready)
30 begin
31   case state is
32     when steady =>
33       x <= "0000000";
34       y <= "00000";
35       c <= "0000000";
36     if (dcf_debug = '0') and (buf = '1') then
37       new_state <= uit;
38     elsif (dcf_debug = '1') and (buf ='0') then
39       new_state <= aan;
40     else
41       new_state <= state;
42     end if;
43   when aan =>
44     x <= x_pos;
45     y <= y_pos;
46     c <= c_aan;
47     if (ready = '1') then
48       new_state <= aan_ready;
49     elsif (dcf_debug ='0') then
50       new_state <= uit;
51     else
52       new_state <= aan;
53     end if;
54   when aan_ready =>

```

```

55          x <= x_pos;
56          y <= y_pos;
57          c <= c_aan;
58      if (ready = '0') then
59          new_state <= steady;
60      elsif (dcf_debug = '0') then
61          new_state <= uit;
62      else
63          new_state <= aan_ready;
64      end if;
65  when uit =>
66      x <= x_pos;
67      y <= y_pos;
68      c <= c_uit;
69      if (ready = '1') then
70          new_state <= uit_ready;
71      elsif (dcf_debug = '1') then
72          new_state <= aan;
73      else
74          new_state <= uit;
75      end if;
76  when uit_ready =>
77      x <= x_pos;
78      y <= y_pos;
79      c <= c_uit;
80      if (ready = '0') then
81          new_state <= steady;
82      elsif (dcf_debug = '1') then
83          new_state <= aan;
84      else
85          new_state <= uit_ready;
86      end if;
87  end case;
88 end process;
89 end behaviour;

```

### B.7.7. ENTITY GELUID

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity geluid is
5     port(clk: in std_logic;
6           reset: in std_logic;
7           ready: in std_logic;
8           menu: in std_logic_vector (2 downto 0);
9           geluid_signaal: in    std_logic;
10          x     : out   std_logic_vector(6 downto 0);
11          y     : out   std_logic_vector(5 downto 0);
12          c : out std_logic_vector(6 downto 0));
13 end geluid;

```

### B.7.8. BEHAVIOURAL GELUID

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 architecture behaviour of geluid is
5 type geluid_state is (steady, aan, uit, uit_ready, aan_ready);
6 signal state, new_state: geluid_state;
7
8 constant x_pos: std_logic_vector (6 downto 0) := "0000001";
9 constant y_pos: std_logic_vector (5 downto 0) := "010111";
10 constant c_aan: std_logic_vector (6 downto 0) := "0100000";
11 constant c_uit: std_logic_vector (6 downto 0) := "0100001";
12
13 begin
14
15 l11: process(clk)
16 begin
17     if (clk'event and clk='1') then

```

```
18      if reset = '1' then
19          state <= steady;
20      else
21          state <= new_state;
22      end if;
23  end if;
24 end process;
25
26 lbl2: process (state, menu, geluid_signaal, ready)
27 begin
28     case state is
29         when steady =>
30             x <= "0000000";
31             y <= "000000";
32             c <= "0000000";
33             if (geluid_signaal = '0') and (menu = "100") then
34                 new_state <= uit;
35             elsif (geluid_signaal = '1') and (menu = "100") then
36                 new_state <= aan;
37             else
38                 new_state <= steady;
39             end if;
40         when aan =>
41             x <= x_pos;
42             y <= y_pos;
43             c <= c_aan;
44             if (ready = '1') then
45                 new_state <= aan_ready;
46             elsif (menu = "100") and (geluid_signaal = '0') then
47                 new_state <= uit;
48             else
49                 new_state <= aan;
50             end if;
51         when aan_ready =>
52             x <= x_pos;
53             y <= y_pos;
54             c <= c_aan;
55             if (ready = '0') then
56                 new_state <= steady;
57             elsif (menu = "100") and (geluid_signaal = '0') then
58                 new_state <= uit;
59             else
60                 new_state <= aan_ready;
61             end if;
62         when uit =>
63             x <= x_pos;
64             y <= y_pos;
65             c <= c_uit;
66             if (ready = '1') then
67                 new_state <= uit_ready;
68             elsif (menu = "100") and (geluid_signaal = '1') then
69                 new_state <= aan;
70             else
71                 new_state <= uit;
72             end if;
73         when uit_ready =>
74             x <= x_pos;
75             y <= y_pos;
76             c <= c_uit;
77             if (ready = '0') then
78                 new_state <= steady;
79             elsif (menu = "100") and (geluid_signaal = '1') then
80                 new_state <= aan;
81             else
82                 new_state <= uit_ready;
83             end if;
84     end case;
85 end process;
86 end behaviour;
```

### B.7.9. ENTITY LICHT

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity licht is
5   port(clk      :in  std_logic;
6        reset    :in  std_logic;
7        ready    :in  std_logic;
8        menu     :in  std_logic_vector(2 downto 0);
9        licht_signaal:in  std_logic;
10       x        :out std_logic_vector(6 downto 0);
11       y        :out std_logic_vector(5 downto 0);
12       c        :out std_logic_vector(6 downto 0));
13 end licht;
```

### B.7.10. BEHAVIOURAL LICHT

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 architecture behaviour of licht is
5 type licht_state is (steady, aan, uit, uit_ready, aan_ready);
6 signal state, new_state: licht_state;
7
8 constant x_pos: std_logic_vector (6 downto 0) := "0000001";
9 constant y_pos: std_logic_vector (5 downto 0) := "001100";
10 constant c_aan: std_logic_vector (6 downto 0) := "0100010";
11 constant c_uit: std_logic_vector (6 downto 0) := "0100011";
12
13 begin
14
15 lbl1: process(clk)
16 begin
17   if (clk'event and clk='1') then
18     if reset = '1' then
19       state <= steady;
20     else
21       state <= new_state;
22     end if;
23   end if;
24 end process;
25
26 lbl2: process (state, menu, licht_signaal, ready)
27 begin
28   case state is
29     when steady =>
30       x <= "0000000";
31       y <= "000000";
32       c <= "0000000";
33       if (licht_signaal = '0') and (menu = "011") then
34         new_state <= uit;
35       elsif (licht_signaal = '1') and (menu = "011") then
36         new_state <= aan;
37       else
38         new_state <= steady;
39       end if;
40     when aan =>
41       x <= x_pos;
42       y <= y_pos;
43       c <= c_aan;
44       if (ready = '1') then
45         new_state <= aan_ready;
46       elsif (menu = "011") and (licht_signaal = '0') then
47         new_state <= uit;
48       else
49         new_state <= aan;
50       end if;
51     when aan_ready =>
52       x <= x_pos;
53       y <= y_pos;
```

```

54      c <= c_aan;
55      if (ready = '0') then
56          new_state <= steady;
57      elsif (menu = "011") and (licht_signaal = '0') then
58          new_state <= uit;
59      else
60          new_state <= aan_ready;
61      end if;
62  when uit =>
63      x <= x_pos;
64      y <= y_pos;
65      c <= c_uit;
66      if (ready = '1') then
67          new_state <= uit_ready;
68      elsif (menu = "011") and (licht_signaal = '1') then
69          new_state <= aan;
70      else
71          new_state <= uit;
72      end if;
73  when uit_ready =>
74      x <= x_pos;
75      y <= y_pos;
76      c <= c_uit;
77      if (ready = '0') then
78          new_state <= steady;
79      elsif (menu = "011") and (licht_signaal = '1') then
80          new_state <= aan;
81      else
82          new_state <= uit_ready;
83      end if;
84  end case;
85 end process;
86 end behaviour;

```

### B.7.11. ENTITY MENU SCHERM

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity menu_scherm is
5     port(clk      :in  std_logic;
6           reset    :in  std_logic;
7           ready    :in  std_logic;
8           menu     :in  std_logic_vector(2 downto 0);
9           alarm    :in  std_logic;
10          x_menu   :out std_logic_vector(6 downto 0);
11          y_menu   :out std_logic_vector(5 downto 0);
12          c_menu   :out std_logic_vector(6 downto 0));
13 end menu_scherm;

```

### B.7.12. BEHAVIOURAL MENU SCHERM

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 architecture behaviour of menu_scherm is
5
6 type menu_state is (steady, schrijven, ready_state);
7 signal state, new_state: menu_state;
8 signal buf, new_buf: std_logic_vector (2 downto 0);
9 signal alarm_buf: std_logic;
10 signal ready_sig, new_ready_sig : std_logic;
11
12 constant x_pos_menu: std_logic_vector (6 downto 0) := "0000011";      -- x-positie voor menu
13 constant y_pos_menu: std_logic_vector (5 downto 0) := "100011";      -- y-positie voor menu
14 constant c_0: std_logic_vector (6 downto 0) := "0101010";      -- leeg, alarm aan
15 constant c_1: std_logic_vector (6 downto 0) := "0100110";      -- uren aanpassen
16 constant c_2: std_logic_vector (6 downto 0) := "0100111";      -- minuten aanpassen
17 constant c_3: std_logic_vector (6 downto 0) := "0101001";      -- licht aanpassen
18 constant c_4: std_logic_vector (6 downto 0) := "0101000";      -- geluid aanpassen
19 constant c_5: std_logic_vector (6 downto 0) := "0101011";      -- tijd aanpassen

```

```

20 constant c_6: std_logic_vector (6 downto 0) := "0101100"; -- leeg, alarm uit
21
22
23 begin
24
25 lbl1: process(clk, reset, new_state, alarm, new_ready_sig, alarm_buf)
26 begin
27     if (clk'event and clk='1') then
28         if reset = '1' then
29             state <= steady;
30             buf <= "000";
31             alarm_buf <= '0';
32             ready_sig <= '0';
33         else
34             state <= new_state;
35             buf <= new_buf;
36             alarm_buf <= alarm;
37             ready_sig <= new_ready_sig;
38         end if;
39     end if;
40 end process;
41
42
43 lbl2: process (state, menu, ready, buf, alarm, alarm_buf)
44 begin
45     case state is
46         when steady =>
47             x_menu <= x_pos_menu;
48             y_menu <= y_pos_menu;
49             c_menu <= "0000000";
50             new_buf <= buf;
51             new_ready_sig <= ready;
52             if (alarm /= alarm_buf) then
53                 new_state <= schrijven;
54             else
55                 new_state <= steady;
56             end if;
57             if (menu /= buf) then
58                 new_state <= schrijven;
59             else
60                 new_state <= steady;
61             end if;
62         when schrijven =>
63             x_menu <= x_pos_menu;
64             y_menu <= y_pos_menu;
65             new_ready_sig <= ready;
66             case menu is
67                 when "001" =>
68                     c_menu <= c_1;
69                     new_buf <= "001";
70                 when "010" =>
71                     c_menu <= c_2;
72                     new_buf <= "010";
73                 when "011" =>
74                     c_menu <= c_3;
75                     new_buf <= "011";
76                 when "100" =>
77                     c_menu <= c_4;
78                     new_buf <= "100";
79                 when "000" =>
80                     c_menu <= c_0;
81                     new_buf <= "000";
82                 when "101" =>
83                     c_menu <= c_5;
84                     new_buf <= "101";
85                 when others =>
86                     if (alarm = '1') then
87                         c_menu <= c_0;
88                     else
89                         c_menu <= c_6;
90                     end if;

```

```

91           new_buf <= menu;
92       end case;
93       if(ready = '0') then
94           if(ready_sig = '1') then
95               new_state <= steady;
96           else
97               new_state <= schrijven;
98           end if;
99       else
100           new_state <= schrijven;
101       end if;
102   when others =>
103       x_menu <= x_pos_menu;
104       y_menu <= y_pos_menu;
105       c_menu <= "0000000";
106       new_buf <= buf;
107       new_ready_sig <= ready;
108
109   end case;
110 end process;
111
112 end behaviour;

```

### B.7.13. ENTITY WEKTIJD

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity wektijd is
5     port(clk          :in  std_logic;
6           reset        :in  std_logic;
7           ready         :in  std_logic;
8           menu         :in  std_logic_vector(2 downto 0);
9           wektijd_uren:in  std_logic_vector(5 downto 0);
10          wektijd_min :in  std_logic_vector(6 downto 0);
11          x           :out std_logic_vector(6 downto 0);
12          y           :out std_logic_vector(5 downto 0);
13          c           :out std_logic_vector(6 downto 0));
14 end wektijd;

```

### B.7.14. BEHAVIOURAL WEKTIJD

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3 use IEEE.std_logic_arith.all;
4
5 architecture behaviour of wektijd is
6 constant char_0_x : std_logic_vector(6 downto 0) := "1011100";
7 constant char_0_y : std_logic_vector(5 downto 0) := "010010";
8
9 constant char_1_x : std_logic_vector(6 downto 0) := "1100010";
10 constant char_1_y : std_logic_vector(5 downto 0) := "010010";
11
12 constant char_2_x : std_logic_vector(6 downto 0) := "1101010";
13 constant char_2_y : std_logic_vector(5 downto 0) := "010010";
14
15 constant char_3_x : std_logic_vector(6 downto 0) := "1110000";
16 constant char_3_y : std_logic_vector(5 downto 0) := "010010";
17
18 --Characters
19 constant char_0 : std_logic_vector(6 downto 0) := "0000000";
20 constant char_1 : std_logic_vector(6 downto 0) := "0000001";
21 constant char_2 : std_logic_vector(6 downto 0) := "0000010";
22 constant char_3 : std_logic_vector(6 downto 0) := "0000011";
23 constant char_4 : std_logic_vector(6 downto 0) := "0000100";
24 constant char_5 : std_logic_vector(6 downto 0) := "0000101";
25 constant char_6 : std_logic_vector(6 downto 0) := "0000110";
26 constant char_7 : std_logic_vector(6 downto 0) := "0000111";
27 constant char_8 : std_logic_vector(6 downto 0) := "0001000";
28 constant char_9 : std_logic_vector(6 downto 0) := "0001001";
29

```

```

30  --Signals
31  type states is (rust, char_0_state,char_1_state, char_2_state, char_3_state);
32  signal state, new_state : states;
33
34  signal ready_sig, new_ready_sig : std_logic;
35  --signal minuten_tijd, new_minuten_tijd : unsigned(6 downto 0);
36  --signal uren_tijd, new_uren_tijd : unsigned(5 downto 0);
37
38
39 begin
40 process(clk, state, new_state, reset)--process om van state te veranderen en voor de reset
41 begin
42
43     if(rising_edge(clk)) then
44         if(reset = '1') then
45             state <= rust;
46             ready_sig <= '0';
47         else
48             state <= new_state;
49             ready_sig <= new_ready_sig;
50         end if;
51     end if;
52 end process;
53
54 process(ready, wektijd_uren, wektijd_min, state)--process voor rekenen
55 begin
56     case(state) is
57         when rust =>
58             new_ready_sig <= ready;
59             x <= "0000000";
60             y <= "000000";
61             c <= "0000000";
62             if (menu = "010") then
63                 new_state <= char_0_state;
64             else
65                 new_state <= rust;
66             end if;
67         when char_0_state =>
68             new_ready_sig <= ready;
69             x <= char_0_x;
70             y <= char_0_y;
71             case(wektijd_uren(5 downto 4)) is
72                 when "00" =>
73                     c <= char_0;
74                 when "01" =>
75                     c <= char_1;
76                 when "10" =>
77                     c <= char_2;
78                 when others =>
79                     c <= char_0;
80             end case;
81             if(ready = '0') then
82                 if(ready_sig = '1') then
83                     new_state <= char_1_state;
84                 else
85                     new_state <= char_0_state;
86                 end if;
87             else
88                 new_state <= char_0_state;
89             end if;
90         when char_1_state =>
91             new_ready_sig <= ready;
92             x <= char_1_x;
93             y <= char_1_y;
94             if(ready = '0') then
95                 if(ready_sig = '1') then
96                     new_state <= char_2_state;
97                 else
98                     new_state <= char_1_state;
99                 end if;
100            else

```

```

101      new_state <= char_1_state;
102  end if;
103  case (wektijd_uren(3 downto 0)) is
104    when "0001" =>
105      c <= char_1;
106    when "0010" =>
107      c <= char_2;
108    when "0011" =>
109      c <= char_3;
110    when "0100" =>
111      c <= char_4;
112    when "0101" =>
113      c <= char_5;
114    when "0110" =>
115      c <= char_6;
116    when "0111" =>
117      c <= char_7;
118    when "1000" =>
119      c <= char_8;
120    when "1001" =>
121      c <= char_9;
122    when others =>
123      c <= char_0;
124  end case;
125  when char_2_state =>
126    new_ready_sig <= ready;
127  x <= char_2_x;
128  y <= char_2_y;
129  case(wektijd_min(6 downto 4)) is
130    when "000" =>
131      c <= char_0;
132    when "001" =>
133      c <= char_1;
134    when "010" =>
135      c <= char_2;
136    when "011" =>
137      c <= char_3;
138    when "100" =>
139      c <= char_4;
140    when "101" =>
141      c <= char_5;
142    when others =>
143      c <= char_0;
144  end case;
145  if(ready = '0') then
146    if(ready_sig = '1') then
147      new_state <= char_3_state;
148    else
149      new_state <= char_2_state;
150    end if;
151  else
152    new_state <= char_2_state;
153  end if;
154
155  when char_3_state =>
156    new_ready_sig <= ready;
157  x <= char_3_x;
158  y <= char_3_y;
159  case(wektijd_min(3 downto 0)) is
160    when "0001" =>
161      c <= char_1;
162    when "0010" =>
163      c <= char_2;
164    when "0011" =>
165      c <= char_3;
166    when "0100" =>
167      c <= char_4;
168    when "0101" =>
169      c <= char_5;
170    when "0110" =>
171      c <= char_6;

```

```

172      when "0111" =>
173          c <= char_7;
174      when "1000" =>
175          c <= char_8;
176      when "1001" =>
177          c <= char_9;
178      when others =>
179          c <= char_0;
180  end case;
181  if(ready = '0') then
182      if(ready_sig = '1') then
183          new_state <= rust;
184      else
185          new_state <= char_3_state;
186      end if;
187  else
188      new_state <= char_3_state;
189  end if;
190  when others =>
191      new_ready_sig <= ready;
192      new_state <= rust;
193      x <= "0000000";
194      y <= "000000";
195      c <= "0000000";
196  end case;
197 end process;
198 end behaviour;

```

### B.7.15. ENTITY SEND CONTROL

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity send_control is
5     port(clk           :in  std_logic;
6           reset         :in  std_logic;
7           data_out      :out  std_logic_vector(6 downto 0);
8           clk_out       :out  std_logic;
9           selector      :out  std_logic_vector(2 downto 0);
10          x            :in   std_logic_vector(6 downto 0);
11          y            :in   std_logic_vector(5 downto 0);
12          c            :in   std_logic_vector(6 downto 0)
13      );
14 end send_control;

```

### B.7.16. BEHAVIOURAL SEND CONTROL

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 architecture behaviour of send_control is
5
6 type states is (rust, send_x, send_x_clock, send_y, send_y_clock, send_c, send_c_clock);
7
8 signal next_state, state : states;
9
10 signal selectort, new_selectort, next_selectort : std_logic_vector(2 downto 0);
11 signal data : std_logic_vector(6 downto 0);
12
13
14 begin
15
16 process(clk, reset, next_state)
17 begin
18 if(rising_edge(clk)) then
19     if(reset = '1') then
20         state <= rust;
21         selectort <= "000";
22     else
23         state <= next_state;
24         selectort <= new_selectort;

```

```

25      end if;
26  end if;
27  end process;
28
29 process(state, next_selectort, x, y, c)
30 begin
31   case(state) is
32     when rust =>
33       data <= "0000000";
34       clk_out <= '1';
35     if(c = "0000000") then
36       next_state <= rust;
37       new_selectort <= next_selectort;
38     else
39       next_state <= send_x;
40       new_selectort <= selectort;
41     end if;
42     when send_x =>
43       data <= x;
44       clk_out <= '0';
45       new_selectort <= selectort;
46       next_state <= send_x_clock;
47     when send_x_clock =>
48       data <= x;
49       clk_out <= '1';
50       new_selectort <= selectort;
51       next_state <= send_y;
52     when send_y =>
53       data(6) <= '0';
54       data(5 downto 0) <= y;
55       clk_out <= '0';
56       new_selectort <= selectort;
57       next_state <= send_y_clock;
58     when send_y_clock =>
59       data(6) <= '0';
60       data(5 downto 0) <= y;
61       clk_out <= '1';
62       new_selectort <= selectort;
63       next_state <= send_c;
64     when send_c =>
65       data <= c;
66       clk_out <= '0';
67       new_selectort <= selectort;
68       next_state <= send_c_clock;
69     when send_c_clock =>
70       data <= c;
71       clk_out <= '1';
72       new_selectort <= next_selectort;
73       next_state <= rust;
74   end case;
75 end process;
76
77
78
79
80
81
82 process(selectort)
83 begin
84   case(selectort) is
85     when "000" =>
86       next_selectort <= "001";
87     when "001" =>
88       next_selectort <= "010";
89     when "010" =>
90       next_selectort <= "011";
91     when "011" =>
92       next_selectort <= "100";
93     when "100" =>
94       next_selectort <= "101";
95     when "101" =>

```

```

96         next_selectort <= "110";
97     when "110" =>
98         next_selectort <= "000";
99     when others =>
100        next_selectort <= "000";
101    end case;
102 end process;
103
104 data_out <= data;
105 selector <= selectort;
106
107 end behaviour;
```

### B.7.17. ENTITY SEND BUS

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 entity send_bus is
5   port(
6     clk : in std_logic;
7     reset : in std_logic;
8     selector:in std_logic_vector(2 downto 0);
9     x_out:out std_logic_vector(6 downto 0);
10    y_out:out std_logic_vector(5 downto 0);
11    c_out:out std_logic_vector(6 downto 0);
12    x_in_0:in std_logic_vector(6 downto 0);
13    y_in_0:in std_logic_vector(5 downto 0);
14    c_in_0:in std_logic_vector(6 downto 0);
15    ready_0:out std_logic;
16    x_in_1:in std_logic_vector(6 downto 0);
17    y_in_1:in std_logic_vector(5 downto 0);
18    c_in_1:in std_logic_vector(6 downto 0);
19    ready_1:out std_logic;
20    x_in_2:in std_logic_vector(6 downto 0);
21    y_in_2:in std_logic_vector(5 downto 0);
22    c_in_2:in std_logic_vector(6 downto 0);
23    ready_2:out std_logic;
24    x_in_3:in std_logic_vector(6 downto 0);
25    y_in_3:in std_logic_vector(5 downto 0);
26    c_in_3:in std_logic_vector(6 downto 0);
27    ready_3:out std_logic;
28    x_in_4:in std_logic_vector(6 downto 0);
29    y_in_4:in std_logic_vector(5 downto 0);
30    c_in_4:in std_logic_vector(6 downto 0);
31    ready_4:out std_logic;
32    x_in_5:in std_logic_vector(6 downto 0);
33    y_in_5:in std_logic_vector(5 downto 0);
34    c_in_5:in std_logic_vector(6 downto 0);
35    ready_5:out std_logic;
36    x_in_6:in std_logic_vector(6 downto 0);
37    y_in_6:in std_logic_vector(5 downto 0);
38    c_in_6:in std_logic_vector(6 downto 0);
39    ready_6:out std_logic);
40 end send_bus;
```

### B.7.18. BEHAVIOURAL SEND BUS

```

1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 architecture behaviour of send_bus is
5
6
7
8 begin
9 process(selector)
10 begin
11   case (selector) is
12     when "000" =>
13       x_out <= x_in_0;
```

```
14      y_out <= y_in_0;
15      c_out <= c_in_0;
16      ready_0 <= '1';
17      ready_1 <= '0';
18      ready_2 <= '0';
19      ready_3 <= '0';
20      ready_4 <= '0';
21      ready_5 <= '0';
22      ready_6 <= '0';
23  when "001" =>
24      x_out <= x_in_1;
25      y_out <= y_in_1;
26      c_out <= c_in_1;
27      ready_0 <= '0';
28      ready_1 <= '1';
29      ready_2 <= '0';
30      ready_3 <= '0';
31      ready_4 <= '0';
32      ready_5 <= '0';
33      ready_6 <= '0';
34  when "010" =>
35      x_out <= x_in_2;
36      y_out <= y_in_2;
37      c_out <= c_in_2;
38      ready_0 <= '0';
39      ready_1 <= '0';
40      ready_2 <= '1';
41      ready_3 <= '0';
42      ready_4 <= '0';
43      ready_5 <= '0';
44      ready_6 <= '0';
45  when "011" =>
46      x_out <= x_in_3;
47      y_out <= y_in_3;
48      c_out <= c_in_3;
49      ready_0 <= '0';
50      ready_1 <= '0';
51      ready_2 <= '0';
52      ready_3 <= '1';
53      ready_4 <= '0';
54      ready_5 <= '0';
55      ready_6 <= '0';
56  when "100" =>
57      x_out <= x_in_4;
58      y_out <= y_in_4;
59      c_out <= c_in_4;
60      ready_0 <= '0';
61      ready_1 <= '0';
62      ready_2 <= '0';
63      ready_3 <= '0';
64      ready_4 <= '1';
65      ready_5 <= '0';
66      ready_6 <= '0';
67  when "101" =>
68      x_out <= x_in_5;
69      y_out <= y_in_5;
70      c_out <= c_in_5;
71      ready_0 <= '0';
72      ready_1 <= '0';
73      ready_2 <= '0';
74      ready_3 <= '0';
75      ready_4 <= '0';
76      ready_5 <= '1';
77      ready_6 <= '0';
78  when "110" =>
79      x_out <= x_in_6;
80      y_out <= y_in_6;
81      c_out <= c_in_6;
82      ready_0 <= '0';
83      ready_1 <= '0';
84      ready_2 <= '0';
```

```

85      ready_3 <= '0';
86      ready_4 <= '0';
87      ready_5 <= '0';
88      ready_6 <= '1';
89      when others =>
90          x_out <= x_in_0;
91          y_out <= y_in_0;
92          c_out <= c_in_0;
93          ready_0 <= '1';
94          ready_1 <= '0';
95          ready_2 <= '0';
96          ready_3 <= '0';
97          ready_4 <= '0';
98          ready_5 <= '0';
99          ready_6 <= '0';
100     end case;
101 end process;
102 end behaviour;

```

### B.7.19. ENTITY SEND TOP

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity send_top is
6 port (
7     clk : in std_logic;
8     reset : in std_logic;
9     x_in_0 : in std_logic_vector(6 downto 0);
10    y_in_0 : in std_logic_vector(5 downto 0);
11    c_in_0 : in std_logic_vector(6 downto 0);
12    ready_0 : out std_logic;
13
14    x_in_1 : in std_logic_vector(6 downto 0);
15    y_in_1 : in std_logic_vector(5 downto 0);
16    c_in_1 : in std_logic_vector(6 downto 0);
17    ready_1 : out std_logic;
18
19    x_in_2 : in std_logic_vector(6 downto 0);
20    y_in_2 : in std_logic_vector(5 downto 0);
21    c_in_2 : in std_logic_vector(6 downto 0);
22    ready_2 : out std_logic;
23
24    x_in_3 : in std_logic_vector(6 downto 0);
25    y_in_3 : in std_logic_vector(5 downto 0);
26    c_in_3 : in std_logic_vector(6 downto 0);
27    ready_3 : out std_logic;
28
29    x_in_4 : in std_logic_vector(6 downto 0);
30    y_in_4 : in std_logic_vector(5 downto 0);
31    c_in_4 : in std_logic_vector(6 downto 0);
32    ready_4 : out std_logic;
33
34    x_in_5 : in std_logic_vector(6 downto 0);
35    y_in_5 : in std_logic_vector(5 downto 0);
36    c_in_5 : in std_logic_vector(6 downto 0);
37    ready_5 : out std_logic;
38
39    x_in_6 : in std_logic_vector(6 downto 0);
40    y_in_6 : in std_logic_vector(5 downto 0);
41    c_in_6 : in std_logic_vector(6 downto 0);
42    ready_6 : out std_logic;
43    data_out : out std_logic_vector(6 downto 0);
44    clk_out : out std_logic);
45 end send_top;

```

### B.7.20. STRUCTURAL SEND TOP

```

1
2 library ieee;
3 use ieee.std_logic_1164.all;

```

```

4  use ieee.numeric_std.all;
5
6  architecture structure of send_top is
7
8  component send_bus is
9    port(  clk           : in  std_logic;
10         reset        : in  std_logic;
11         selector     : in  std_logic_vector(2 downto 0);
12         x_out        : out std_logic_vector(6 downto 0);
13         y_out        : out std_logic_vector(5 downto 0);
14         c_out        : out std_logic_vector(6 downto 0);
15
16         x_in_0       : in  std_logic_vector(6 downto 0);
17         y_in_0       : in  std_logic_vector(5 downto 0);
18         c_in_0       : in  std_logic_vector(6 downto 0);
19         ready_0      : out std_logic;
20
21         x_in_1       : in  std_logic_vector(6 downto 0);
22         y_in_1       : in  std_logic_vector(5 downto 0);
23         c_in_1       : in  std_logic_vector(6 downto 0);
24         ready_1      : out std_logic;
25
26         x_in_2       : in  std_logic_vector(6 downto 0);
27         y_in_2       : in  std_logic_vector(5 downto 0);
28         c_in_2       : in  std_logic_vector(6 downto 0);
29         ready_2      : out std_logic;
30
31         x_in_3       : in  std_logic_vector(6 downto 0);
32         y_in_3       : in  std_logic_vector(5 downto 0);
33         c_in_3       : in  std_logic_vector(6 downto 0);
34         ready_3      : out std_logic;
35
36         x_in_4       : in  std_logic_vector(6 downto 0);
37         y_in_4       : in  std_logic_vector(5 downto 0);
38         c_in_4       : in  std_logic_vector(6 downto 0);
39         ready_4      : out std_logic;
40
41         x_in_5       : in  std_logic_vector(6 downto 0);
42         y_in_5       : in  std_logic_vector(5 downto 0);
43         c_in_5       : in  std_logic_vector(6 downto 0);
44         ready_5      : out std_logic;
45
46         x_in_6       : in  std_logic_vector(6 downto 0);
47         y_in_6       : in  std_logic_vector(5 downto 0);
48         c_in_6       : in  std_logic_vector(6 downto 0);
49         ready_6      : out std_logic;
50     );
51   end component send_bus;
52
53  component send_control is
54    port(  clk           : in  std_logic;
55         reset        : in  std_logic;
56         x            : in  std_logic_vector(6 downto 0);
57         y            : in  std_logic_vector(5 downto 0);
58         c            : in  std_logic_vector(6 downto 0);
59         data_out     : out std_logic_vector(6 downto 0);
60         clk_out      : out std_logic;
61         selector     : out std_logic_vector(2 downto 0)
62     );
63   end component send_control;
64   --signal ready_tijd, ready_menu, ready_geluid, ready_def, ready_dcf, ready_datum,
65   --      ready_wek, ready_licht   : std_logic;
66   signal sel : std_logic_vector(2 downto 0);
67   --signal x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_f : std_logic_vector(6 downto 0);
68   --signal y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_f : std_logic_vector(5 downto 0);
69   --signal c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_f : std_logic_vector(6 downto 0);
70   signal x_f,c_f : std_logic_vector(6 downto 0);
71   signal y_f : std_logic_vector(5 downto 0);
72 begin
73   sbus: send_bus      port map(clk, reset, sel, x_f, y_f, c_f, x_in_0, y_in_0, c_in_0,
74                                 ready_0, x_in_1, y_in_1, c_in_1, ready_1, x_in_2, y_in_2, c_in_2, ready_2, x_in_3,

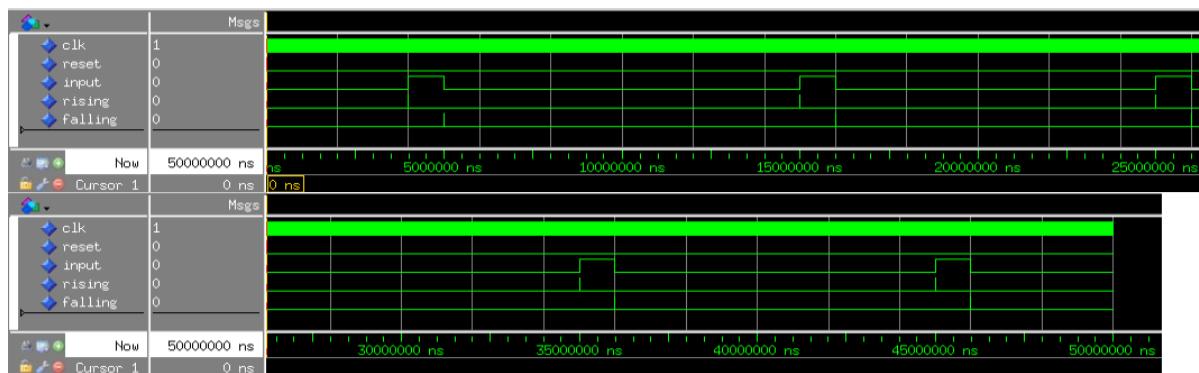
```

```
    y_in_3, c_in_3, ready_3, x_in_4, y_in_4, c_in_4, ready_4, x_in_5, y_in_5, c_in_5,
    ready_5, x_in_6, y_in_6, c_in_6, ready_6);
73  scontrol: send_control      port map(clk, reset, x_f, y_f, c_f, data_out, clk_out, sel) ;
74  end structure;
```

# C

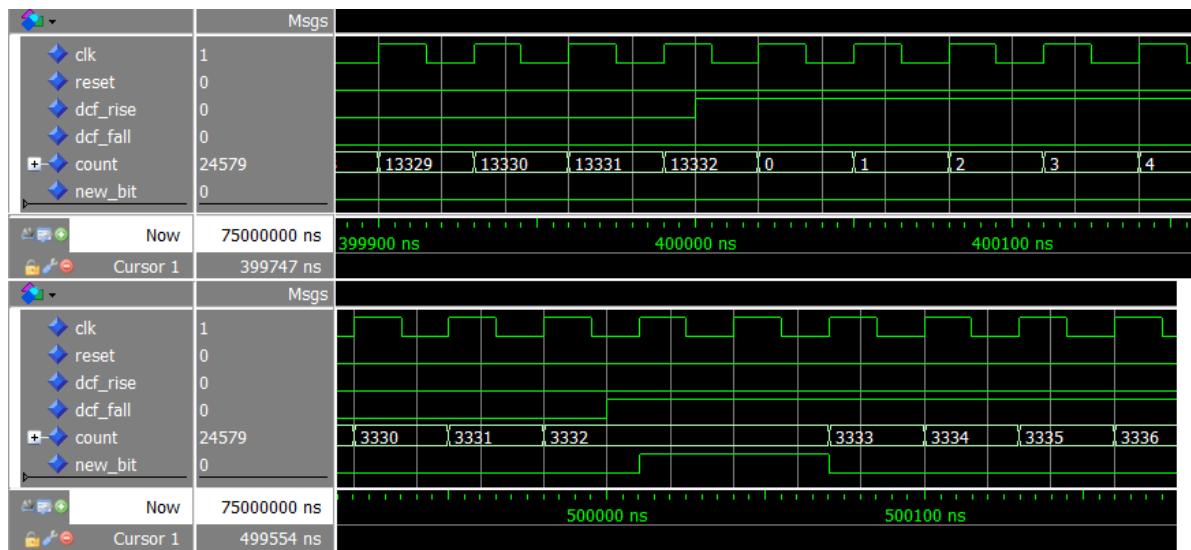
## SIMULATIERESULTATEN VAN HET DCF77 BLOK

### C.0.21. BEHAVIOUR EDGE DETECTOR

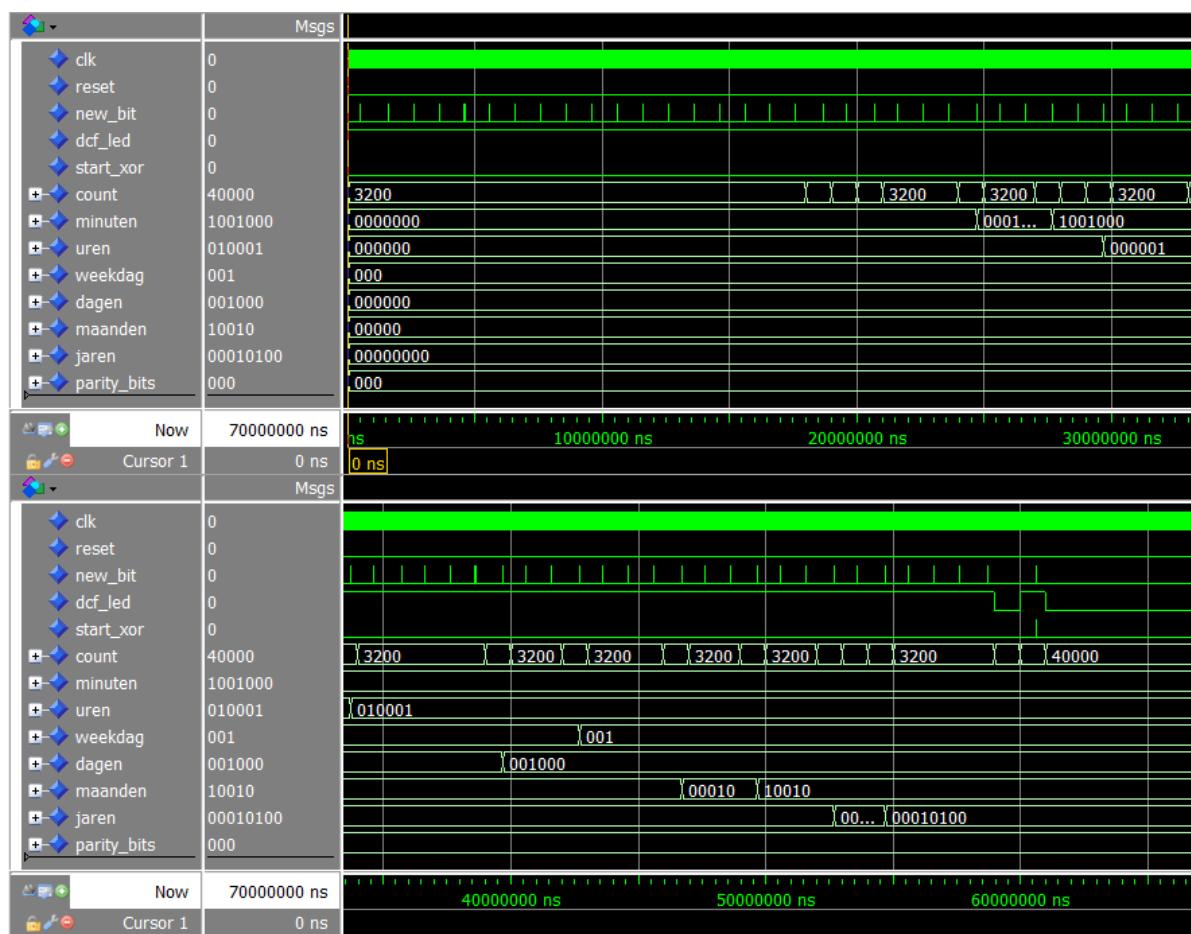


Figuur C.1: Simulatie van de edge detector (50 ms op schaal)

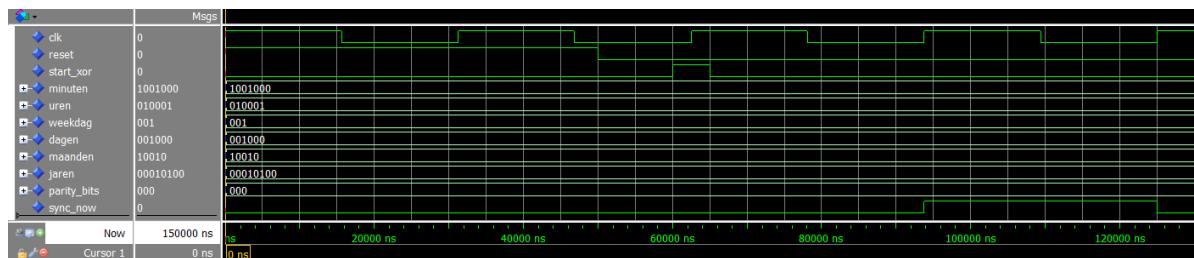
- C.0.22. BEHAVIOUR DCF COUNTER
- C.0.23. BEHAVIOUR DCF DECODER
- C.0.24. BEHAVIOUR PARITY CHECK
- C.0.25. BEHAVIOUR SYNCTIME
- C.0.26. BEHAVIOUR KLOKDELER
- C.0.27. BEHAVIOUR MOD24 TELLER
- C.0.28. BEHAVIOUR MOD60 TELLER
- C.0.29. BEHAVIOUR MOD60 CLOCK
- C.0.30. BEHAVIOUR KLOK
- C.0.31. BEHAVIOUR DCF77 BLOK
- C.0.32. SWITCH-LEVEL DCF77 BLOK



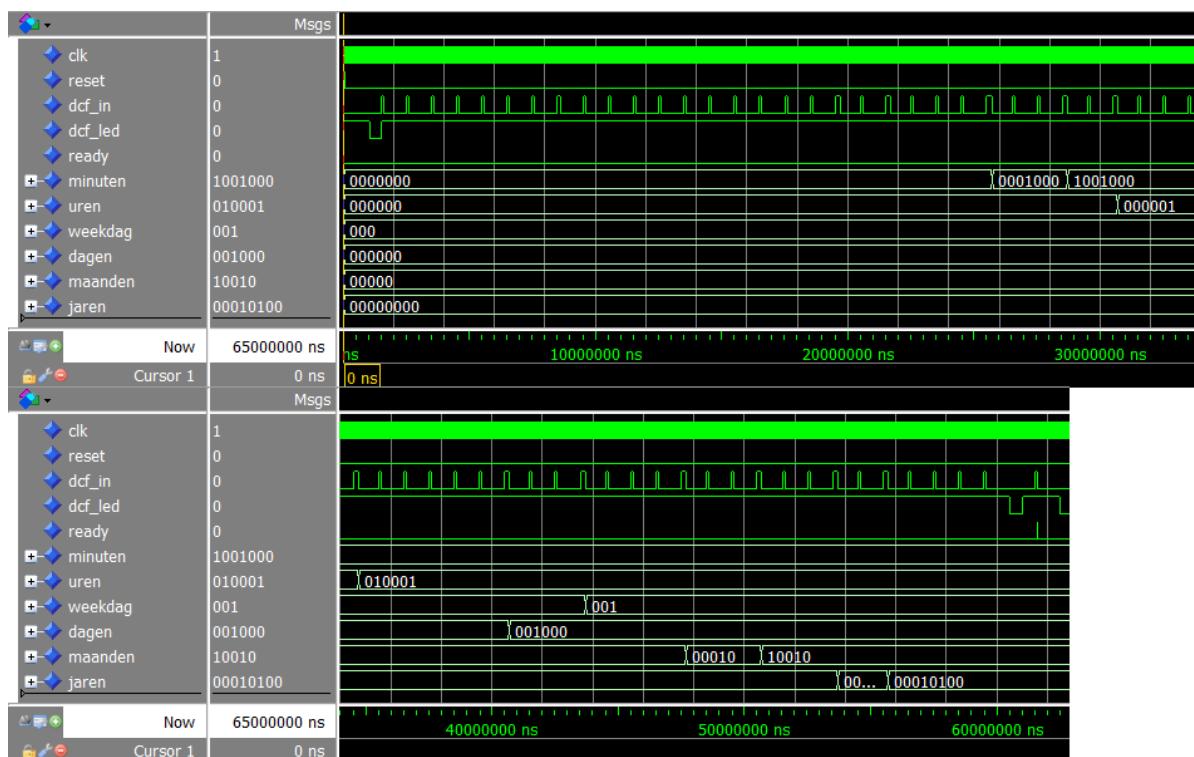
Figuur C.2: Simulatie van de DCF counter (details van 75 ms op schaal)



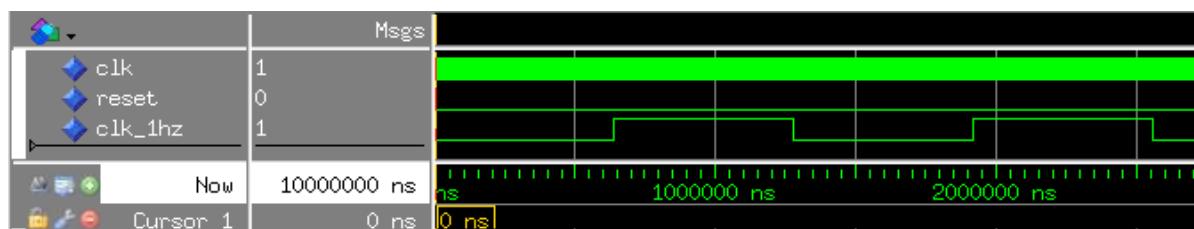
Figuur C.3: Simulatie van de DCF decoder (70 ms op schaal)



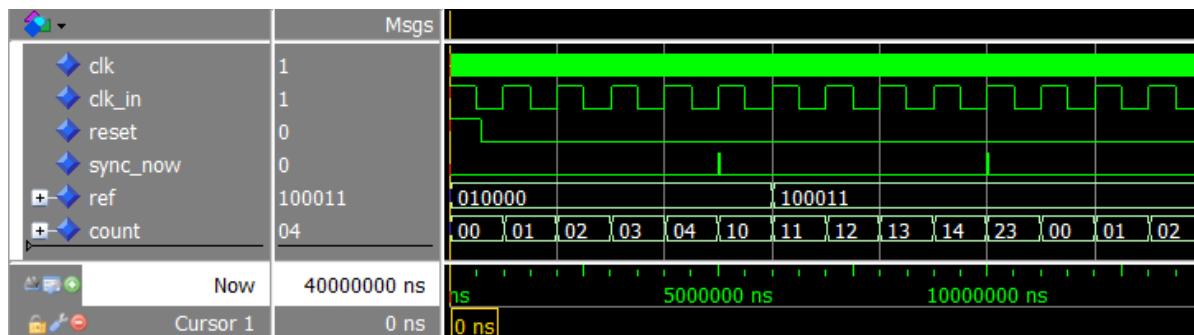
Figuur C.4: Simulatie van de parity check (150 microseonden)



Figuur C.5: Simulatie van synctime (65 ms op schaal)



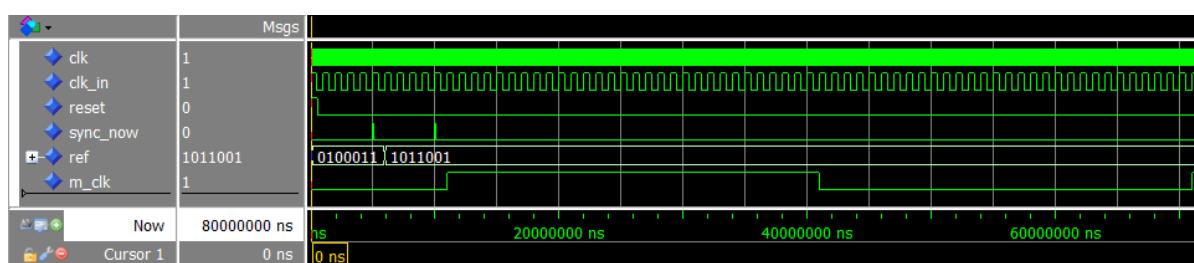
Figuur C.6: Simulatie van de klokdeler (10 ms op schaal)



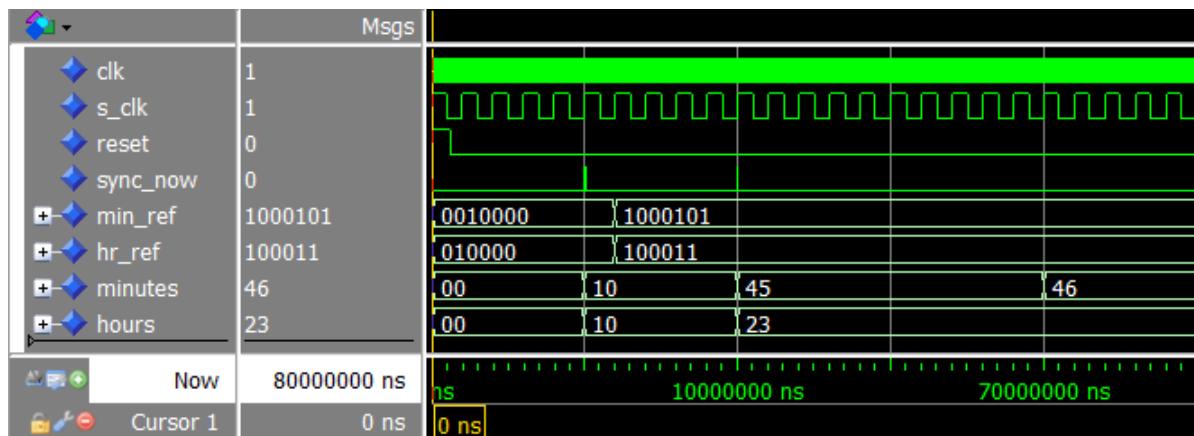
Figuur C.7: Simulatie van de mod24 teller (40 ms op schaal)



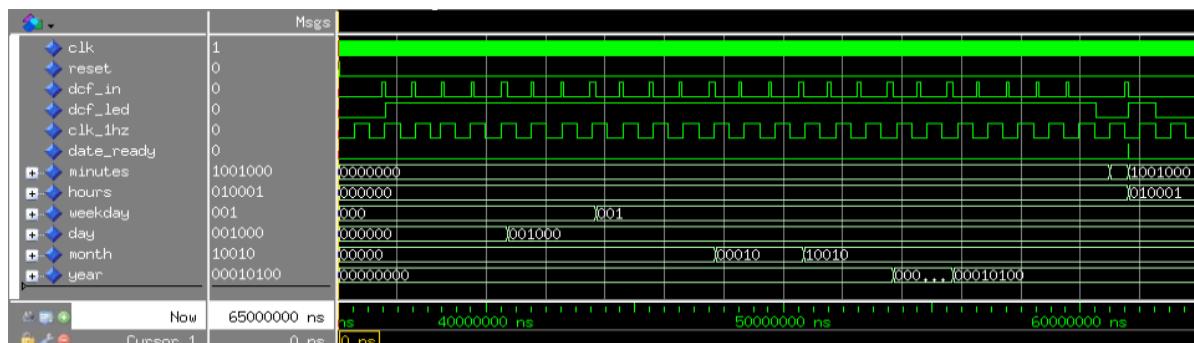
Figuur C.8: Simulatie van de mod60 teller (50 ms op schaal)



Figuur C.9: Simulatie van de mod60 clock (80 ms op schaal)



Figuur C.10: Simulatie van de klok (80 ms op schaal)



Figuur C.11: Simulatie van het DCF77 blok (65 ms op schaal)

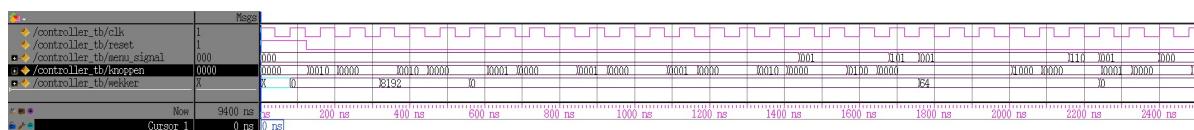


Figuur C.12: Switch-level simulatie van het DCF77 blok (65 ms op schaal)

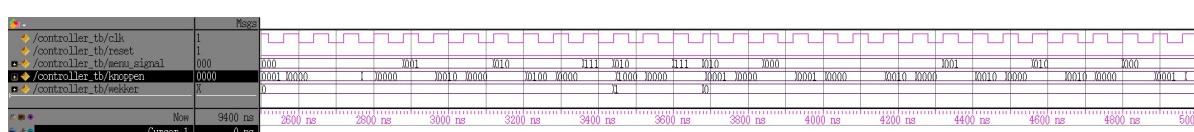
D

# SIMULATIES RESULTATEN VAN DE CONTROLLER

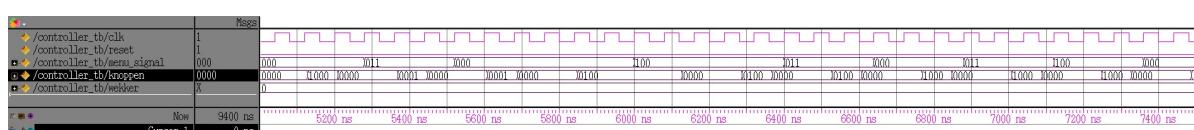
## D.1. BEHAVIORAL SIMULATIE



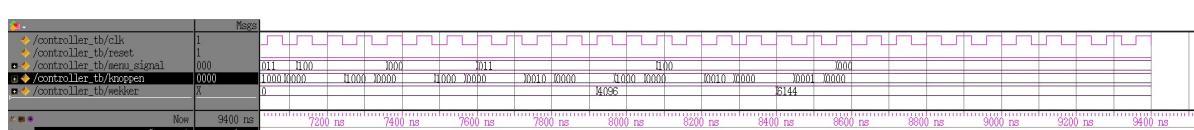
Figuur D.1: Simulatie van 0 tot 2500ns



Figuur D.2: Simulatie van 2500ns tot 5000ns

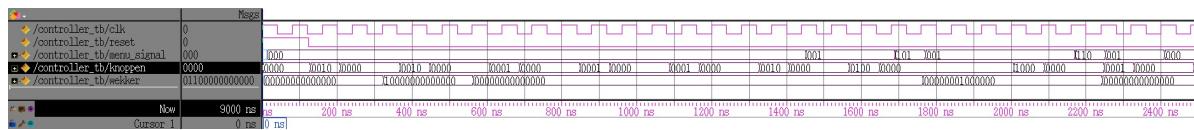


Eig 3: Simulatie van 5000ns tot 7500ns

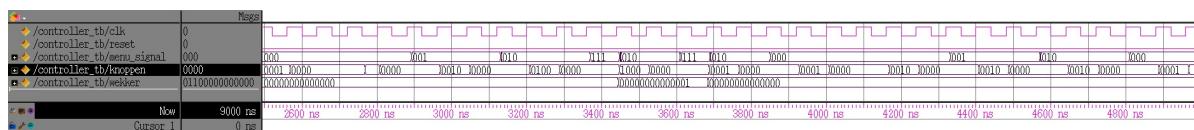


Figuur D.4: Simulatie van 7500ns tot het einde

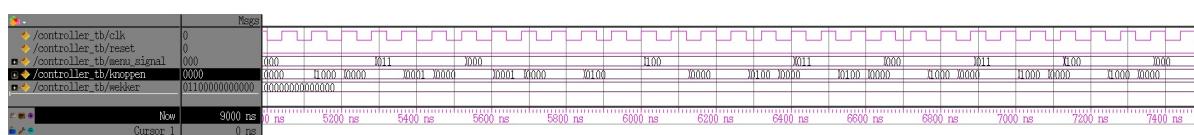
## D.2. SYNTHESIZE SIMULATIE



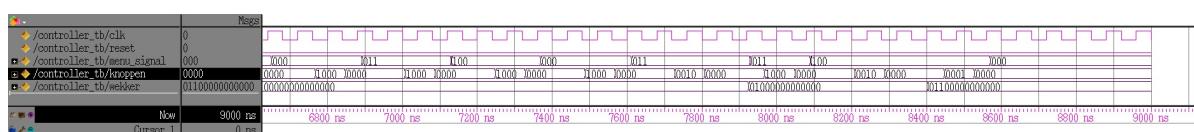
Figuur D.5: Simulatie van 0 tot 2500ns



Figuur D.6: Simulatie van 2500ns tot 5000ns

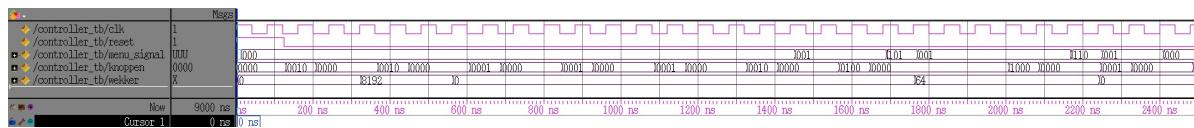


Figuur D.7: Simulatie van 5000ns tot 7500ns

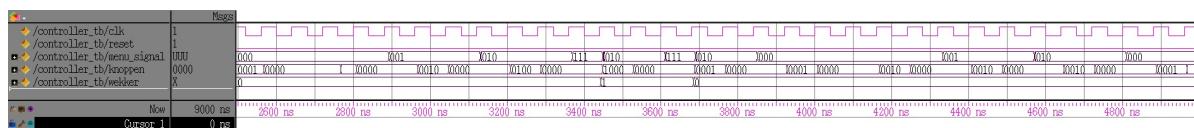


Figuur D.8: Simulatie van 7500ns tot het einde

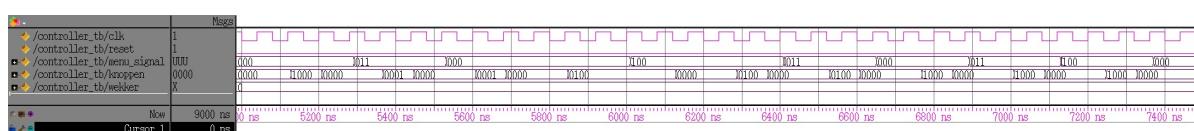
### D.3. EXTRACTED SIMULATIE



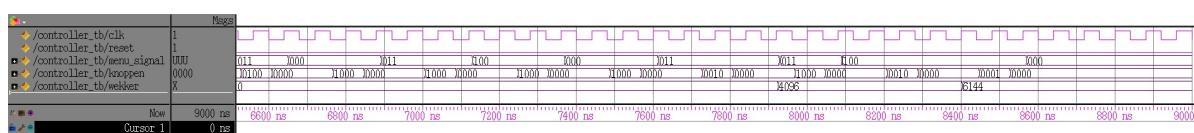
Figuur D.9: Simulatie van 0 tot 2500ns



Figuur D.10: Simulatie van 2500ns tot 5000ns

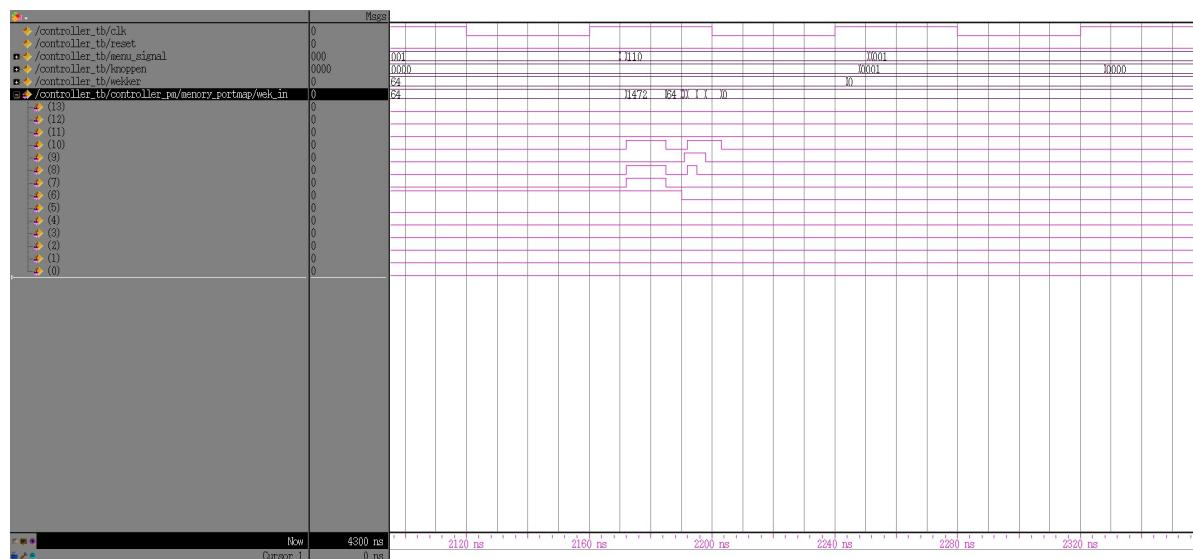


Figuur D.11: Simulatie van 5000ns tot 7500ns



Figuur D.12: Simulatie van 7500ns tot het einde

## D.4. TIMING

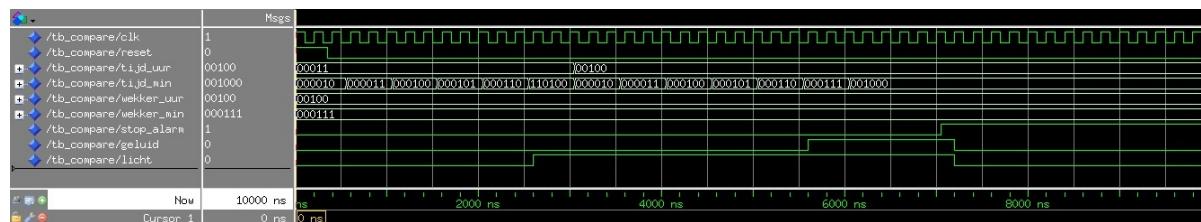


Figuur D.13: Timing problemen

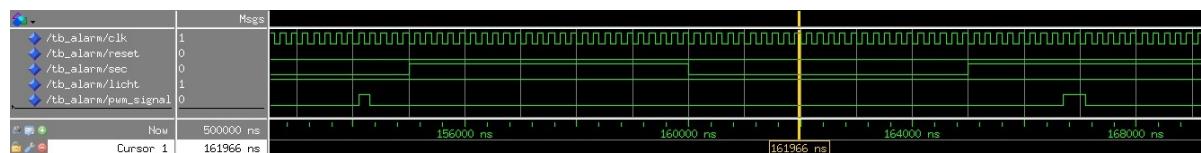
# E

## SIMULATIES RESULTATEN VAN HET ALARM

### E.1. BEHAVIORAL SIMULATIE

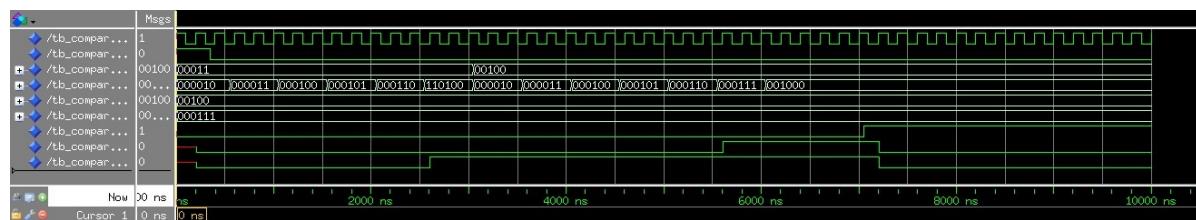


Figuur E.1: Simulatie van 0 tot 10000 ns van compare

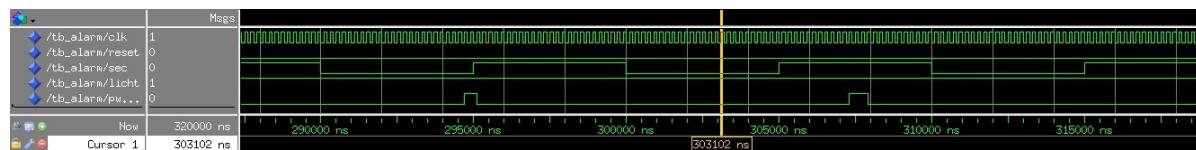


Figuur E.2: Simulatie van 151000 tot 170000 ns van alarm

## E.2. EXTRACTED SIMULATIE



Figuur E.3: Simulatie van 0 tot 10000 ns van alarm

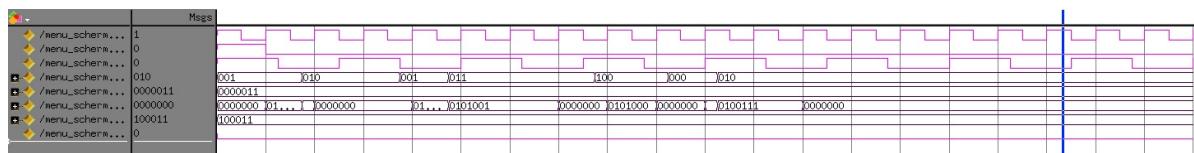


Figuur E.4: Simulatie van 290000 tot 320000 ns van alarm

# F

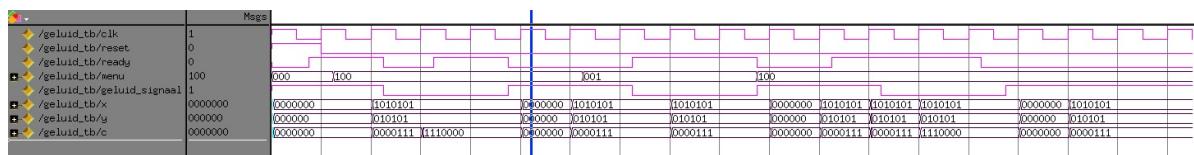
## SIMULATIES RESULTATEN VAN HET LCD

### F.1. MENU



Figuur F.1: Simulatie van het subblok menu

### F.2. GELUID



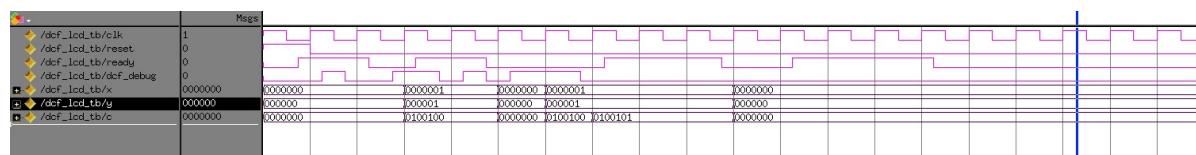
Figuur F.2: Simulatie van het subblok geluid

### F.3. LICHT

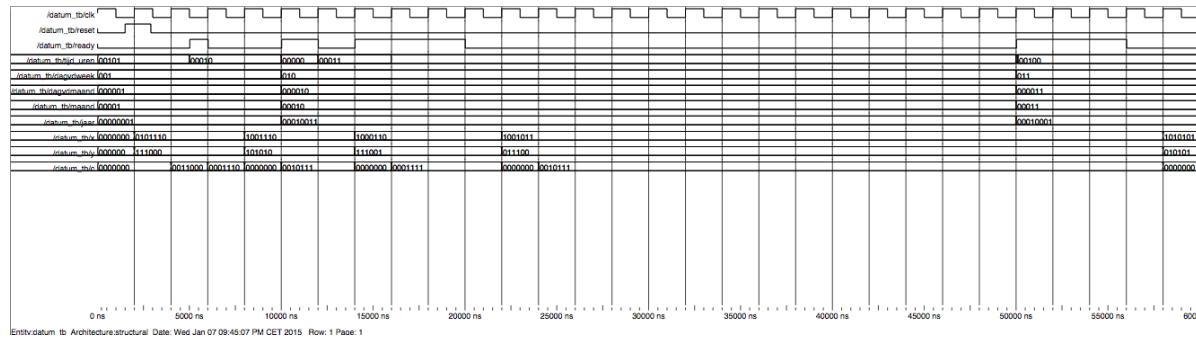


Figuur F.3: Simulatie van het subblok licht

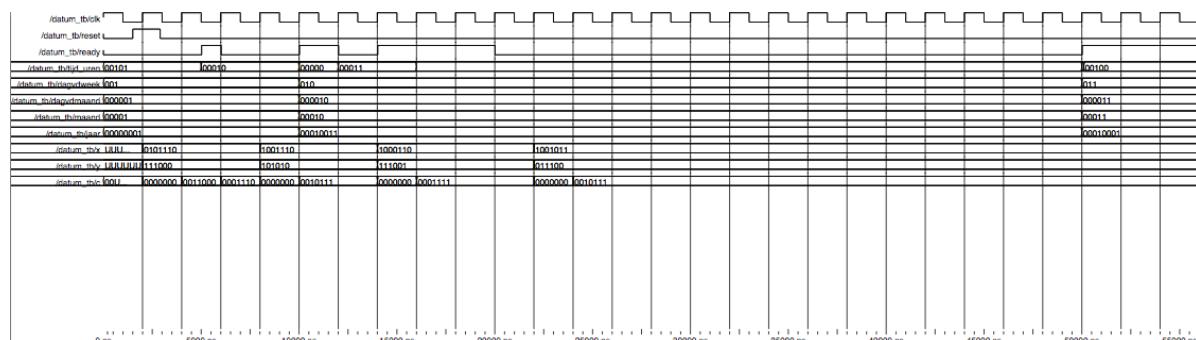
### F.4. DCF



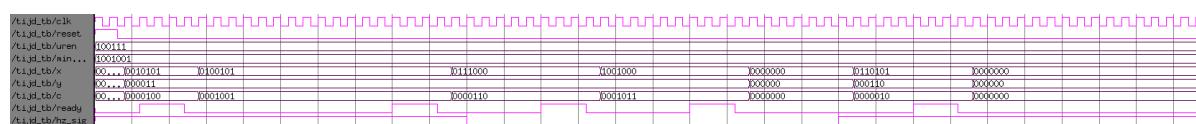
Figuur F.4: Simulatie van het subblok dcf



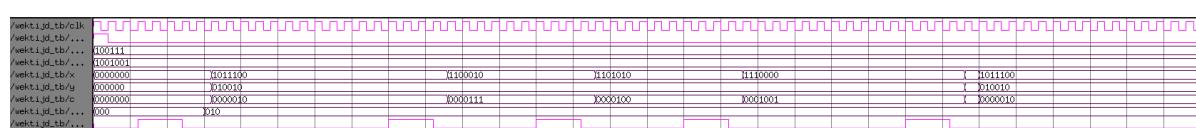
Figuur F.5: Simulatie behavioural van het subblok datum



Figuur F.6: Simulatie circuit van het subblok datum



Figuur F.7: Simulatie van het subblok tijd



Figuur F.8: Simulatie van het subblok wektijd



Figuur F.9: Simulatie van de subblokken send controller en de send bus

## BIBLIOGRAFIE

- [1] Wikipedia, [Dcf77](#), Geraadpleegd op 08-12-2014, url: en.wikipedia.org/wiki/DCF77.
- [2] Physikalisch-Technische Bundesanstalt (PTB), [Dcf77 zeitkode](#), Geraadpleegd op 08-12-2014, url: www.ptb.de/cms/fachabteilungen/abt4/fb-44/ag-442/verbreitung-der-gesetzlichen-zeit/dcf77/zeitcode.html.
- [3] C. J. Wells, [Mathematics - number systems - binary-coded-decimal](#), Geraadpleegd op 12-12-2014, url: www.technologyuk.net/mathematics/number\_systems/binary\_coded\_decimal.shtml.