

# EPO-3

## Extreme Winterslaap Interrupter Final report

14-01-2015

Projectgroep A1

Technische Universiteit Delft



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 schermje. In het verslag is te vinden hoe al deze subsystemen zijn ontworpen en uitgewerkt.

# INHOUDSOPGAVE

<b>Samenvatting</b>	<b>ii</b>
<b>1 Introductie</b>	<b>1</b>
<b>2 Ontwerp specificatie</b>	<b>2</b>
<b>3 Systeem overzicht en ontwerp</b>	<b>3</b>
<b>4 DCF controller</b>	<b>5</b>
4.1 Inleiding . . . . .	5
4.2 Specificaties . . . . .	6
4.2.1 Ingangen . . . . .	6
4.2.2 Uitgangen . . . . .	6
4.3 Gedrag . . . . .	6
<b>5 Main controller</b>	<b>7</b>
5.1 Inleiding . . . . .	7
5.2 Specificaties . . . . .	7
5.2.1 Ingangen . . . . .	7
5.2.2 Uitgangen . . . . .	7
5.2.3 Gedrag . . . . .	8
5.3 Functionaliteit . . . . .	8
5.3.1 FSM . . . . .	8
5.3.2 VHDL code . . . . .	9
5.4 Testen . . . . .	9
5.5 Simulatie . . . . .	9
5.6 Resultaten . . . . .	9
5.6.1 Conclusie en discussie . . . . .	9
<b>6 Alarm</b>	<b>11</b>
6.1 Inleiding . . . . .	11
6.2 Specificaties . . . . .	11
6.2.1 Ingangen . . . . .	11
6.2.2 Uitgangen . . . . .	11
6.2.3 Gedrag . . . . .	11
6.3 Functionaliteit . . . . .	12
6.3.1 FSM . . . . .	12
6.3.2 Code . . . . .	13
6.4 Resultaten . . . . .	13
<b>7 LCD controller</b>	<b>15</b>
7.1 Inleiding . . . . .	15
7.2 Specificaties . . . . .	15
7.2.1 Ingangen . . . . .	15
7.2.2 Uitgangen . . . . .	15
7.2.3 Gedrag . . . . .	15
7.3 Functionaliteit . . . . .	15
7.3.1 FSM . . . . .	15
7.3.2 VHDL code . . . . .	15

7.4	Testen	15
7.5	Simulatie	15
7.6	Resultaten	15
7.6.1	Conclusie en discussie	15
<b>8</b>	<b>Results for total design</b>	<b>16</b>
<b>9</b>	<b>Plan voor het testen van de chip</b>	<b>17</b>
9.1	FPGA bord	17
9.2	Logic Analyzer	17
<b>10</b>	<b>Voortgang van het project</b>	<b>18</b>
10.1	Inleiding	18
10.2	Werkverdeling	18
10.2.1	Module opdracht	18
10.2.2	Wake-up light	18
10.3	Samenwerking binnen de groep	18
10.4	Afspraken binnen de groep	19
<b>11</b>	<b>Conclusie</b>	<b>20</b>
<b>A</b>	<b>VHDL code</b>	<b>21</b>
A.1	Vhdl code van het DCF77 blok	21
A.1.1	Klokdeeler	21
A.1.2	Mod24 (uren) teller	22
A.1.3	Mod60 (minuten) teller	23
A.1.4	Mod60 (seconden) teller	25
A.1.5	Autonome synchroniseerbare klok	27
A.1.6	Edge detector	28
A.1.7	DCF counter	29
A.1.8	DCF decoder	31
A.1.9	Parity check	33
A.1.10	Synctime	35
A.1.11	DCF77 (top-level)	36
A.2	VHDL code controller	37
A.2.1	Top level entity	37
A.2.2	Behavioural VHDL code controller	37
A.2.3	Menu entity	38
A.2.4	Behavioural VHDL code menu	38
A.2.5	Memory	42
A.2.6	Behavioural VHDL memory	42
A.2.7	Entity buffer	43
A.2.8	Behavioural VHDL buffer	43
A.3	Testbenchs voor de controller	44
A.3.1	VHDL controller	44
A.3.2	Testbench VHDL menu	45
A.3.3	Testbench VHDL geheugen	46
A.3.4	Testbench VHDL buffer	47
A.4	Vhdl code van het alarm	48
A.4.1	Entity alarm-compare	48
A.4.2	Behavioural alarm-compare	48
A.4.3	Top entity alarm	49
A.4.4	Behavioural alarm	49
A.4.5	Entity alarm-counter	50
A.4.6	Behavioural alarm-counter	50
A.4.7	Entity alarm-pwm	51
A.4.8	Behavioural alarm-pwm	51

<b>B</b>	<b>Simulatie resultaten</b>	<b>53</b>
B.1	Behavioral simulatie . . . . .	53
B.2	Synthesize simulatie . . . . .	54
B.3	Extracted simulatie . . . . .	55
B.4	Timing . . . . .	56
	<b>Bibliografie</b>	<b>57</b>

# 1

## INTRODUCTIE

Epo 3 staat in het teken van het ontwerpen van een chip. Wat voor product er ontworpen gaat 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-signaal 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 gesynthetiseerd 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

Het systeem moet aan verschillende specificaties voldoen. Zo zal het een algemene reset moeten bevatten. Als gevolg van het indrukken van de resetknop zullen alle opgeslagen waarden en counters op 'nul' worden gezet. Ook zullen alle signalen 'active high' moeten zijn. 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 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 8 bits vector om het scherm aan te sturen via een microcontroller
- Clk\_out, 1 bit ter aansturing van de microcontroller

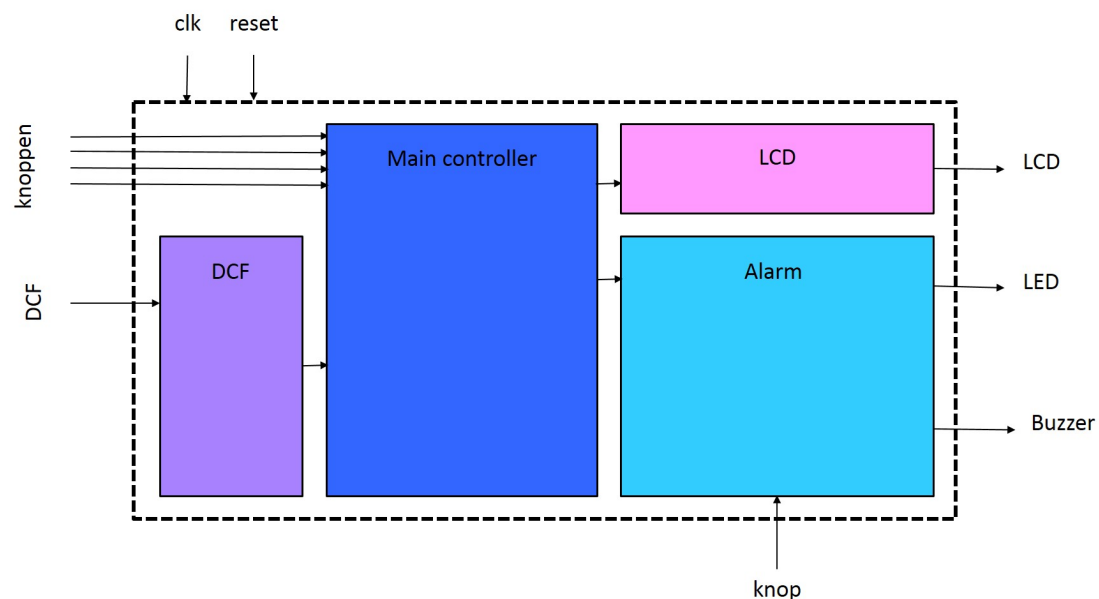
# 3

## 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

De DCF controller vangt het DCF signaal op en zet het om naar een bitvector met datum, uren en minuten. En er wordt 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 er gaat een ledje branden om aan te geven dat de chip geen DCF-signaal meer ontvangt. 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



bestuurd 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-sigitaal gegenereerd wordt wat naar een LED gaat.

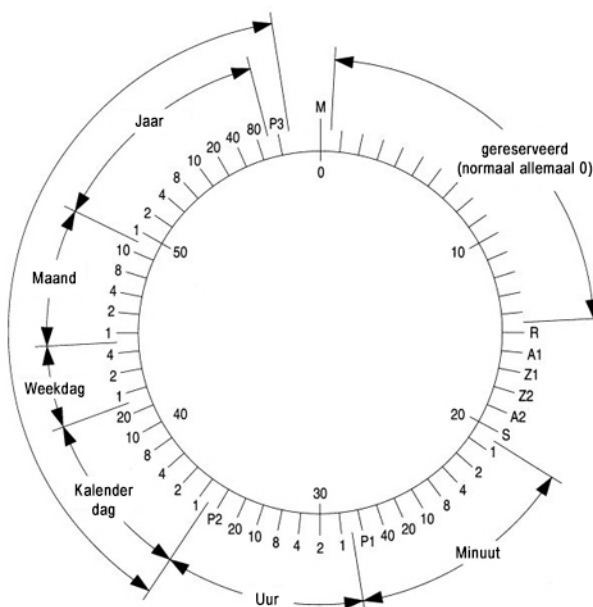
De LCD controller zorgt dat de datum, tijd, ingestelde alarmtijd en de veranderingen in het menu op de LCD zichtbaar zijn. Er wordt een LCD scherm gebruikt waar de pixels afzonderlijk van elkaar aangestuurd worden. Tussen de chip en het scherm zit nog een microcontroller, waarin de karakters zijn opgeslagen,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

# 4

## DCF CONTROLLER

### 4.1. INLEIDING

De basis van onze wekker wordt gelegd door een klok. Uit dit onderdeel, genaamd DCF-controller, komt verschillende data, als de tijd, de datum en het weeknummer. Een van de eigenschappen van deze klok zal zijn dat hij gesynchroniseerd wordt met een zogenaamd DCF-sigitaal. Dit is een signaal dat in Duitsland verzonden wordt en allerlei informatie bevat, als de actuele tijd en de datum. Er zullen meerdere van deze elementen gebruikt worden in onze wekker. Al deze data wordt verzonden door middel van een pulssigitaal. Vanuit Duitsland wordt elke seconde een puls van 100 of 200 ms verstuurd, zodat respectievelijk een 0-bit en een 1-bit doorgegeven wordt. Dit resulteert in een totaal van 59 bits, gevolgd door een seconde 'rust', dat elke minuut opnieuw verzonden wordt. In figuur 3.1 is te zien welke secondes welke informatie bevatten. In de afbeelding wordt een aantal keer P# genoemd. Dit zijn paritychecks om het ingekomen signaal te controleren. Als de voorgaande bits een even aantal logische enen bevat, zal de parity bit een logische 0 geven. [1] *## Controleren ##* De controller converteert dit gedigitaliseerde DCF77 signaal naar een tijdsreferentie, waarna deze een autonome klok synchroniseert, welke de huidige tijd genereert.



Figuur 4.1: Codering van het dcf-sigitaal [1]

## 4.2. SPECIFICATIES

In deze sectie zullen we de in- en uitgangen van de DCF-controller in een overzicht weergeven. Doordat het totale systeem met dit onderdeel start, bevat dit blok enkel standaard ingangen en een ingang van buitenaf. De uitgangen uren en minuten worden doorgestuurd naar de main-controller. De clk van 1 Hz zal in verschillende onderdelen worden gebruikt. De debug\_led zal rechtstreeks op het LCD worden weergegeven.

### 4.2.1. INGANGEN

Dit onderdeel maakt gebruik van de volgende ingangen:

- Reset, standaard input.
- Klok, standaard input.
- DCF, signaal van 'logische' pulsen.

### 4.2.2. UITGANGEN

Dit onderdeel heeft de volgende uitgangen:

- Uren, een BCD vector van 6 bits.
- minuten, een BCD vector van 7 bits.
- clk, een clk die elke seconde een puls geeft.
- debug\_led, een signaal dat een logische 1 doorgeeft zodra er een dcf signaal wordt ontvangen.

## 4.3. GEDRAG

De DCF-controller heeft als belangrijkste gedragsfunctie om de tijd en datum door te geven. Al deze data moet zo vaak mogelijk worden gesynchroniseerd met een inkoment DCF-sigitaal. Dit signaal moet vanaf een antenne gedecodeerd worden en vervolgens moet het signaal via parity bits gecontroleerd worden. Pas als hieraan voldaan is moet het signaal in een register worden geschreven en doorgestuurd naar de andere onderdelen van de chip. Mocht het DCF-sigitaal signaal niet of slecht worden ontvangen dan moet de tijd instantaan en individueel door kunnen lopen. Indien het DCF-sigitaal weer terugkomt moet de controller dit automatisch oppakken. Om aan te geven of er een DCF signaal is, moet er een zogenaamd debug lampje gaan branden als dit gebeurt. In de 60e seconde zal dit signaal laag moeten zijn, doordat dan geen bit doorgegeven wordt.

De huidige, indien mogelijk met het DCF77 signaal gesynchroniseerde, tijd moet worden gegeven in uren (in een BCD gecodeerd signaal van 5-bits) en minuten (in een BCD gecodeerd signaal van 6-bits). De datum, afkomstig uit het DCF77 signaal dient na synchronisatie te worden bewaard totdat er opnieuw met het DCF77 signaal wordt gesynchroniseerd. De datum wordt gegeven in vier vectoren:

- De dag van de week wordt gegeven door drie bits, waarbij maandag is gecodeerd als 001, dinsdag als 010, enz.
- De dag van de maand, gegeven als BCD gecodeerd signaal van 6-bits.
- Het nummer van de maand wordt gegeven als BCD gecodeerd 5-bits signaal.
- De laatste twee cijfers van het huidige jaartal worden bovendien gegeven door een 8-bits BCD gecodeerd signaal.

Daarnaast moet de DCF-controller ook een 1 Hz kloksigitaal via een klokdeler naar alle andere onderdelen kunnen sturen. Als laatste moeten alle subblokken, inclusief registers, van de DCF-controller bij een 'active high' van een resetknop gereset worden.

# 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.

Al deze informatie samengenomen is besloten dat 4 knoppen voor de interface het meest gebruiksvriendelijke resultaat oplevert. 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

#### 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 de tijd dat de wekker af moet gaan en de wekkerdata, dus of het licht en geluid aan staan, en of de wekker überhaupt aanstaat;
- Menu-state, dit is de staat in welke de FSM zich op het moment bevindt. Deze informatie wordt doorgevoerd naar het LCD-scherm om zo te kunnen zien waar in het menu men zit.

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[5 down to 0] daarin staan de minuten wekker[10 down to 6] daarin staan de uren wekker[11] geluid bit wekker[12] led bit wekker[13] wekker bit (Of de wekker uberhaupt 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 state: Rust, Wekkertijd 001 : Uren aanpassen 010 : Minuten aanpassen 011 : Led aanpassen 100 : Geluid aanpassen

### 5.2.3. GEDRAG

Om te beginnen moet de tijd waarop de wekker af moet gaan 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 minuut. Hierna wordt de huidige tijd weer weergegeven.

Daarnaast is een vereiste dat de led uitgezet moet kunnen worden. Afhankelijk van een instelling moet het wake-up-light gedeelte wel of niet aangaan. Hetzelfde geldt voor het geluid.

Dit alles moet zo gebruiksvriendelijk mogelijk gebeuren.

## 5.3. FUNCTIONALITEIT

### 5.3.1. FSM

In fig. 5.1 staat de gemaakte fsm en in tabel 5.2 staan de uitgangen per state gespecificeerd.

Rust, Reset	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 down to 12] = wekdata[13 down to 12] menu = "100"

Tijd uren	enable ='0' wekker=wekdata menu = "001"
Uren plus	enable ='1' wekker=wekdata+1 menu = "001"
Uren min	enable ='1' wekker=wekdata-1 menu = "001"
Tijd minuten	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

### 5.3.2. VHDL CODE

De code voor de controller van de wekker is te vinden in appendix A.2. 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 A.2.1 en A.2.2.
- Het menu, hierin zit de echte logica verwerkt. Deze is te vinden in appendices A.2.3 en A.2.4.
- Het gebruikte geheugen element voor de opslag van 14 bits, te vinden in appendices A.2.5 en A.2.6.
- De gebruikte buffer is te vinden in appendices A.2.7 en A.2.8. 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 A.3.1 tot A.3.4.

## 5.4. TESTEN

Om zeker te zijn dat alles goed werkt worden er drie verschillende testen uitgevoerd. De eerste is op behavioural niveau. Hier wordt getest of de basis van de code werkt zoals verwacht. Na een goed geslaagd resultaat kan de code worden gesynthetiseerd, en deze gesynthetiseerde code worden gesimuleerd. Als er geen fouten optreden kan het ontwerp gemaakt worden, daarna geextraheerd en nogmaals getest worden. De testen worden uitgevoerd met behulp van *Modelsim*.

## 5.5. SIMULATIE

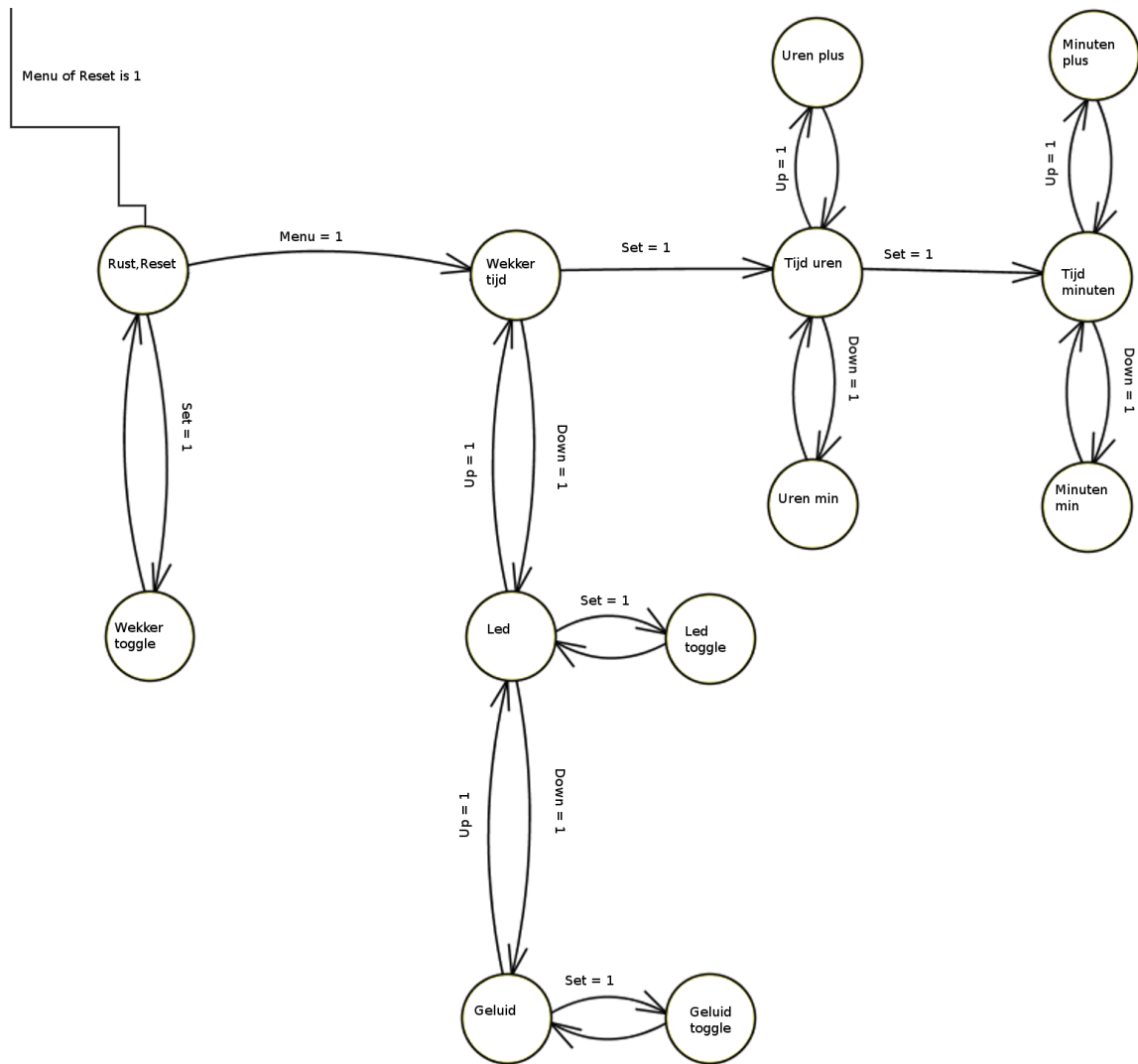
De resultaten van de simulatie staan in appendix B. De testbench is te lang om in een keer weer te geven daarom is deze op geknipt in vier stukken. De testbench die gemaakt is voor de simulatie staat in appendix A.3.1

## 5.6. RESULTATEN

Van fig. B.1 tot en met fig. B.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. B.13. Hieruit is op te maken dat deze 60ns is.

### 5.6.1. CONCLUSIE EN DISCUSSIE

De controller werkt op alle gesimuleerde niveau's naar verwachting. De minimale klok periode bedraagt 60ns om gliches te voorkomen bij het optellen en aftrekken van uren en minuten. De controller maakt op dit moment gebruik van 9088 transistoren waarvan er voor de daadwerkelijke schakelingen slechts 2914 worden gebruikt. De controller maakt op dit moment nog gebruik van het binaire telsysteem (dat gebruik maakt van machten van 2), er bleek echter dat voor de lcd scherm BCD veel beter werkt. Dit moet nog worden geïmplementeerd. Vlak



Figuur 5.1: FSM diagramma van de menu

nadat het inputbuffer gemaakt was kwam men er achter dat in plaats van een buffer ook de rising\_edge functie gebruikt had kunnen 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-sigitaal, signaal om de led aan te sturen.
- Geluid, signaal om een geluid af te laten gaan.

#### 6.2.3. GEDRAG

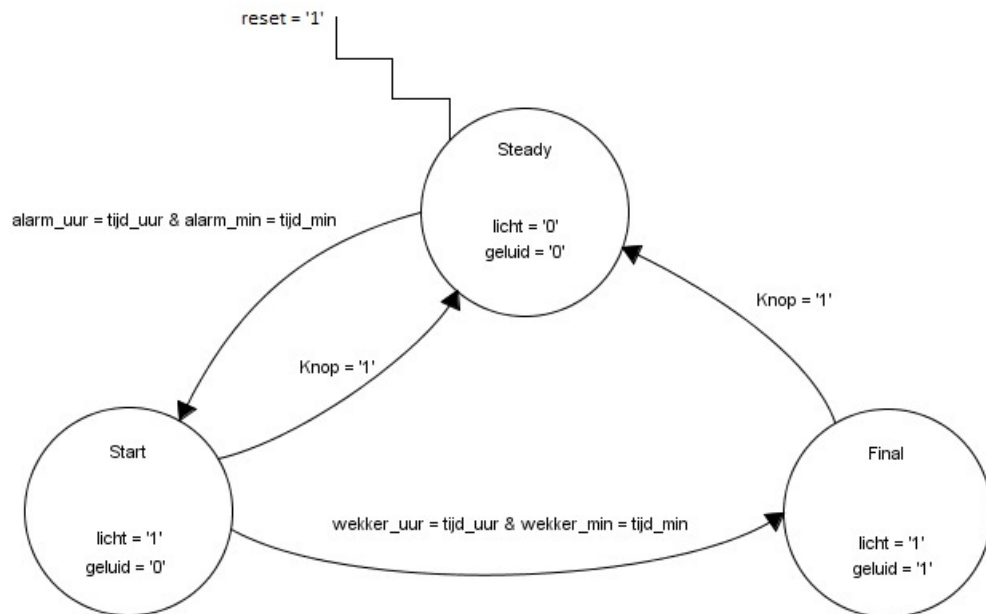
Het alarm moet een bepaalde tijd voordat de wekker is ingesteld aangaan, nu gekozen voor 15 minuten. Er wordt 15 minuten van de ingestelde tijd afgetrokken. Zodra die tijd gelijk is aan de huidige tijd komt er een signaal (licht) aan bij het gedeelte wat voor een pwm signaal zorgt. In dat gedeelte wordt 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-sigitaal 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.



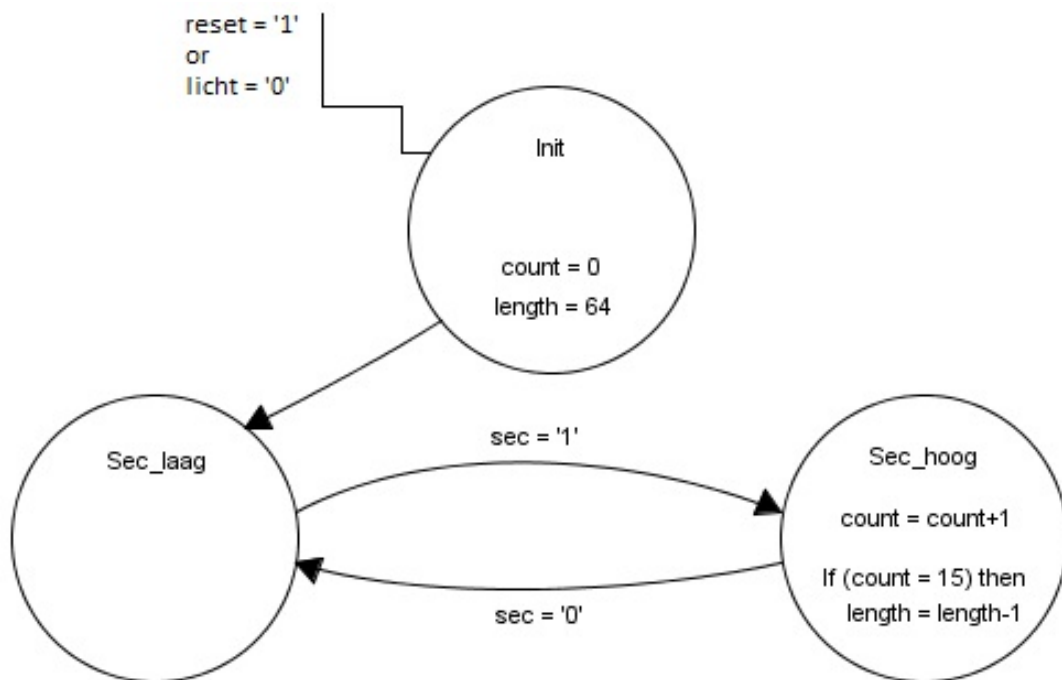
## 6.3. FUNCTIONALITEIT

### 6.3.1. FSM

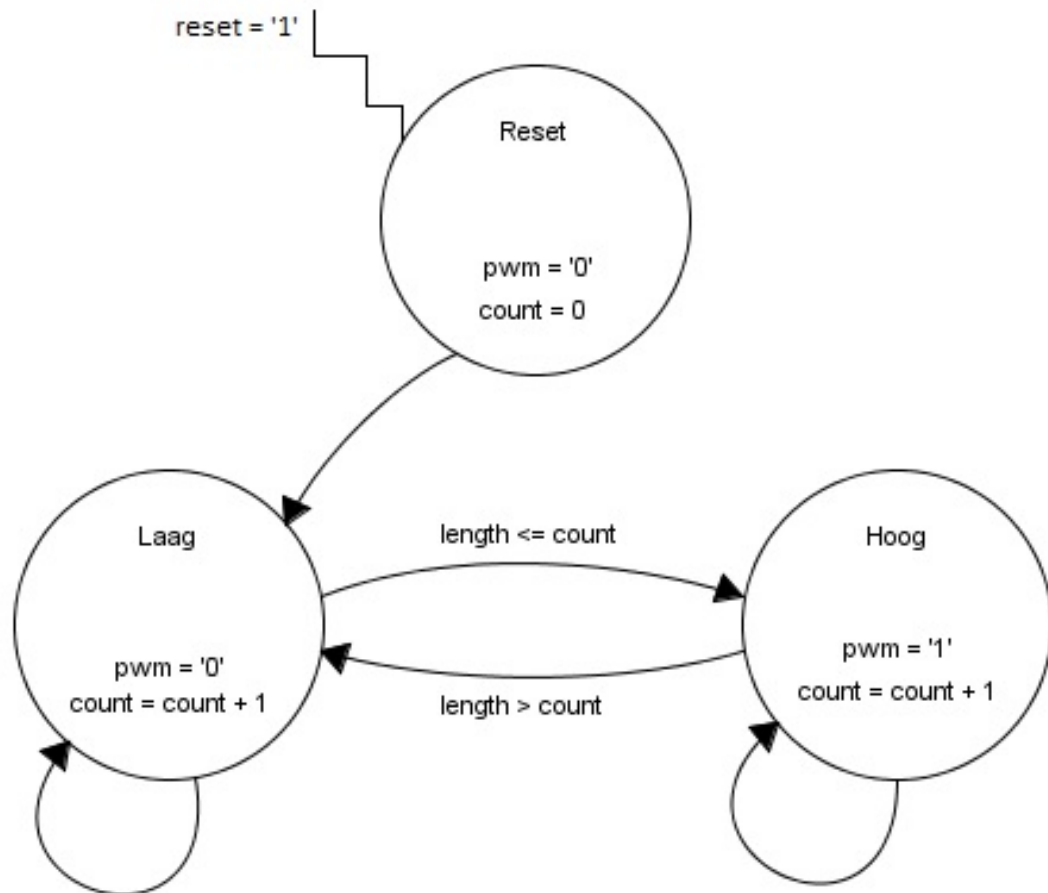
FSM van “alarm-compare”, het gedeelte waar de tijd vergeleken wordt met de ingestelde wekker tijd.



FSM van “alarm-counter”, hier wordt de lengte van het PWM signaal berekend.



FSM van “alarm-pwm”, hier wordt het pwm signaal gegenereerd wat de led aanstuurt.



### 6.3.2. CODE

De code voor het alarm is opgedeeld in 3 stukken:

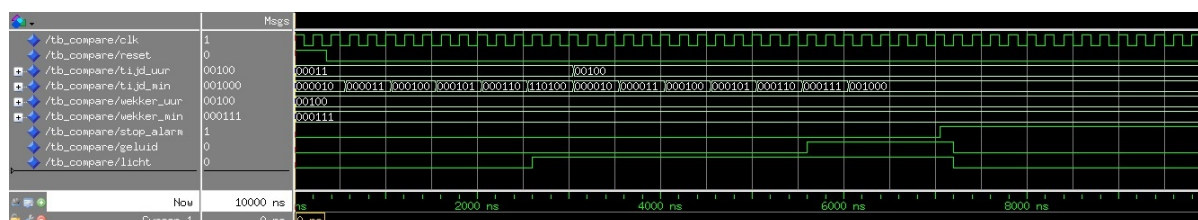
- alarm-compare
- alarm-counter
- alarm-pwm

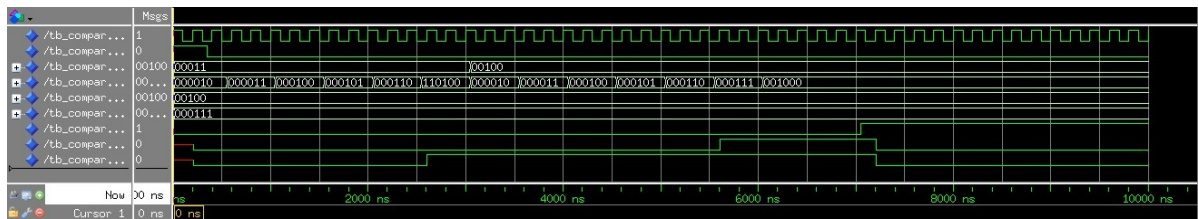
Alarm-counter en alarm-pwm zijn onderdeel van een top entity, alarm. Omdat het nog niet zeker is waar alarm-compare geplaatst gaat worden op de chip is die daar niet bij inbegrepen. De code voor het alarm is te vinden in appendix A.4.

- De entity en behavioural van alarm compare is te vinden in appendices A.4.1 en A.4.2.
- De top entity en port map van alarm is te vinden in appendices A.4.3 en A.4.4.
- De entity en behavioural van alarm-counter is te vinden in appendices A.4.5 en A.4.6.
- De entity en behavioural van alarm-pwm is te vinden in appendices A.4.7 en A.4.8.

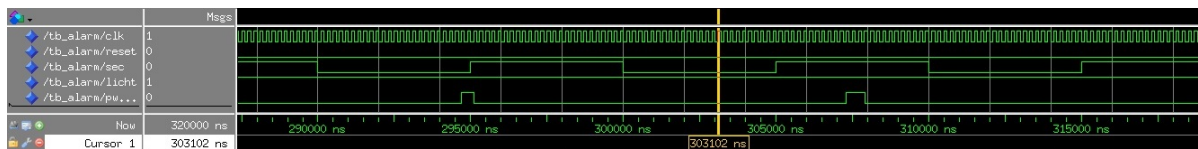
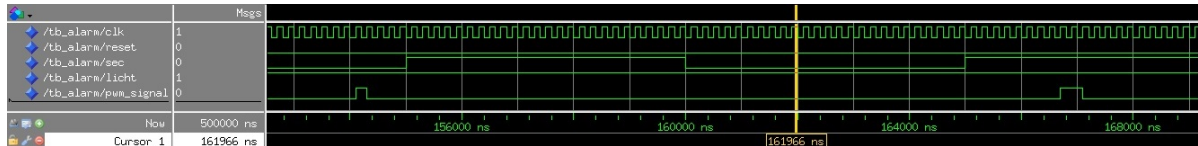
## 6.4. RESULTATEN

Alle onderdelen werken in de simulaties. Zowel de simulatie van de behaviour als van de extracted vhdl. Onderstaande afbeeldingen zijn de resultaten van de simulaties, eerst die van de behaviour daarna van de extracted. De eerste 2 afbeeldingen zijn van “alarm-compare”, de andere 2 van “alarm-pwm”.





Te zien is in de 2 afbeeldingen hier boven 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.



Bovenstaande 2 afbeeldingen laten zien dat op het moment dat er 15 seconde zijn verstreken het pwm-signaal breder wordt.

# 7

## LCD CONTROLLER

### 7.1. INLEIDING

### 7.2. SPECIFICATIES

#### 7.2.1. INGANGEN

- Klok, dit is een standaard input;
- Reset, ook dit is een standaard input;
- Menu vanaf de main controller;
- Tijd en datum vanaf de DCF controller;
- Wektijd vanaf de main controller;

#### 7.2.2. UITGANGEN

- Data, dit is een lijn voor het versturen van de x en y coördinaten naar het LCD scherm;
- SCK, is een klok. Werkt in combinatie met de data lijn. Werkt als een soort spi;

#### 7.2.3. GEDRAG

### 7.3. FUNCTIONALITEIT

#### 7.3.1. FSM

#### 7.3.2. VHDL CODE

### 7.4. TESTEN

### 7.5. SIMULATIE

### 7.6. RESULTATEN

#### 7.6.1. CONCLUSIE EN DISCUSSIE

# 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

Voor het testen zijn een aantal momenten in het proces waarop getest wordt. Zo wordt elk module getest in een simulatie in Modelsim. Hieruit kan opgemaakt worden wat het verwachte gedrag is. Maar een simulatie is niet alles. Daarom kan een module ook nog getest worden door middel van een FPGA te programmeren. De uiteindelijke chip zal getest worden met een logic analyzer en natuurlijk door te kijken of de chip de gewenste output geeft.

### 9.1. FPGA BORD

Het bord dat gebruikt kan worden is een Altera FPGA bord. Dit bord komt met eigen software genaamt Quartus. Deze software kan gebruikt worden om de gemaakte VHDL code om te zetten in een bitstream file en vervolgens het FPGA bord te programmeren. Door de VHDL code op een FPGA te programmeren kan worden geverifieerd of de code het gedrag vertoont wat verwacht wordt. Door simulatie is dit namelijk niet altijd helemaal te zien. Mocht op de FPGA een fout ontdekt worden, dan zal de code hierop aangepast worden en zal de code opnieuw gesimuleerd worden.

### 9.2. LOGIC ANALYZER

De gemaakte chip zal in Q4 worden getest. De chip zal eerst op een logic analyzer worden aangesloten. De analyzer die gebruikt zal worden is een LA-5580.

# 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

#### 10.2.1. MODULE OPDRACHT

De eerste twee weken werd er gewerkt aan een module-opdracht, dit bereide iedereen voor op de echte opdracht. De module-opdracht was vergelijkbaar met de uiteindelijke opdracht, alleen veel kleiner en het onderwerp was anders. De module-opdracht werd in tweetallen voltooid. De onderdelen die gemaakt werden zijn:

- Een ALU
- Een SRAM-module
- Een FIFO-module
- Een SPI-interface

Deze zijn allen ter voorbereiding op de grote opdracht. Deze opdrachten zijn in 2 weken tijd voltooid. Daarnaast heeft elk tweetal de specificaties voor een ander groepje opgesteld, zodat ook hierin ervaring opgedaan zou worden. Dit is nodig aangezien voor de grote opdracht zelf de specificaties opgesteld moesten worden.

#### 10.2.2. WAKE-UP LIGHT

Zodra vastgesteld was wat de grote opdracht zou worden zijn eerst precieze specificaties 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 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 per blok.

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

5 weken is een zeer korte tijd om mensen te leren kennen. Het merendeel van de samenwerking verliep goed, dit onderdeel zal uitgebreid worden in de loop van de komende 5 weken.

**10.4. AFSPRAKEN BINNEN DE GROEP**

Afspraken binnen de groep verliepen soepel. De enkele keer dat dit niet gebeurde was hier een goede reden voor. 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

Alle onderdelen zijn in theorie nu klaar. Individueel zijn ze via *Modelsim* getest en goed bevonden. De onderlinge signalen zijn zo veel mogelijk op elkaar afgestemd. De onderdelen voldoen samen aan de specificaties die eerder gesteld zijn. Echter in de praktijk zullen de onderdelen nog niet feilloos met elkaar samenwerken. Hiervoor zal er meer meer getest worden, bijvoorbeeld op een FPGA bord en een logic analyzer.



## VHDL CODE

### A.1. VHDL CODE VAN HET DCF77 BLOK

#### A.1.1. 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      -- gebruikte states
13      signal state, new_state: kd_state;
14
15      signal count, new_count: std_logic_vector(13 downto 0); -- Voor het tellen tot 32000
16      signal temp, new_temp: std_logic;                     -- Interne versie van het 1 Hz
17      -- signaal
18
19  begin
20      clk_1hz <= temp;                                       -- Uitvoeren van het interne 1 Hz signaal
21
22      process(clk) is
23      begin
24          if (clk'event and clk = '1') then
25              if (reset = '1') then                         -- Systeemreset
26                  temp <= '0';
27                  count <= (others => '0');
28                  state <= clear;
29              else
30                  temp <= new_temp;
31                  count <= new_count;
32                  state <= new_state;
33              end if;
34          end if;
35      end process;
```

```

34
35 process(state, count, temp) is
36 begin
37     case state is
38         when clear => -- Dit is de reset state
39             new_temp <= '0';
40             new_count <= (others => '0');
41             new_state <= counting;
42         when counting => -- Er wordt geteld
43             new_temp <= temp;
44             new_count <= count + 1;
45             if(count < 15998) then
46                 new_state <= counting;
47             else
48                 new_state <= switch;
49             end if;
50         when switch => -- Inverteer de uitgangswaarde
51             new_temp <= not temp;
52             new_count <= (others => '0');
53             new_state <= counting;
54         when others =>
55             new_temp <= '1';
56             new_count <= (others => '0');
57             new_state <= clear;
58     end case;
59 end process;
60 end behaviour;

```

### A.1.2. 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;

```

---

```

1 -- Joran Out, 4331958 & Alex Oudsen, 4325494
2 -- Deze mod24 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 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;

```

### A.1.3. 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;

```

#### A.1.4. 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                                -- Systeemreset
26             s_count <= (others => '0');
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;

```

### A.1.5. AUTONOME SYNCHRONISEERBARE KLOK

```

1  -- Alex Oudsen, 4325494
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  entity asy_klok_bcd is
7      port (clk:          in std_logic; -- 32 kHz systeemklok
8            s_clk:        in std_logic; -- 1 Hz klok uit klokdeleer
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 asy_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 asy_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;

```

### A.1.6. 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;

```

---

```

1  -- Alex Oudsen, 4325494
2  -- De edge detector genereert uit het dcf77 signaal
3  -- aparte pulsen voor de rising en falling edges
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  architecture behaviour of edge_detector is
9
10     type edged is (clear, find_edge, output);
11     signal state, new_state: edged;
12
13     signal temp, new_temp: std_logic;
14     signal rise, new_rise: std_logic;
15     signal fall, new_fall: std_logic;
16
17     signal r_wait, new_r_wait: std_logic;
18     signal f_wait, new_f_wait: std_logic;
19 begin
20     rising <= rise; -- rising en falling zijn de daadwerkelijke uitgangen
21     falling <= fall;
22
23     process(clk)
24     begin
25         if(clk'event and clk = '1') then
26             if(reset = '1') then -- Systeemreset
27                 rise <= '0';
28                 fall <= '0';
29                 temp <= '0';
30
31                 r_wait <= '0';
32                 f_wait <= '0';
33
34                 state <= clear;
35             else
36                 rise <= new_rise;
37                 fall <= new_fall;
38                 temp <= new_temp;
39
40                 r_wait <= new_r_wait;
41                 f_wait <= new_f_wait;
42
43                 state <= new_state;
44             end if;
45         end if;
46     end process;
47
48     process(state, input, temp, rise, fall, r_wait, f_wait) is
49     begin
50

```

```

51     case state is
52         when clear => -- Reset state
53             new_rise <= '0';
54             new_fall <= '0';
55             new_temp <= input;
56
57             new_r_wait <= '0';
58             new_f_wait <= '0';
59
60             new_state <= find_edge;
61         when find_edge => -- Maakt gebruik van vertragingstijd van de not
62             new_rise <= (not temp) and input;
63             new_fall <= (not input) and temp;
64             new_temp <= temp;
65
66             new_r_wait <= '0';
67             new_f_wait <= '0';
68
69             new_state <= output;
70         when output => -- Hier wordt gezorgd voor een bruikbare uitgangspuls
71             new_rise <= rise;
72             new_fall <= fall;
73             new_temp <= temp;
74
75             if (rise = '1' or fall = '1') then
76                 if (rise <= '1' and f_wait = '0') then
77                     new_r_wait <= '0';
78                     new_f_wait <= '1';
79                     new_state <= clear;
80                 elsif (fall <= '1' and r_wait = '0') then
81                     new_r_wait <= '1';
82                     new_f_wait <= '0';
83                     new_state <= clear;
84                 else
85                     new_r_wait <= '0';
86                     new_f_wait <= '0';
87                     new_state <= find_edge;
88                 end if;
89             else
90                 new_r_wait <= '0';
91                 new_f_wait <= '0';
92                 new_state <= find_edge;
93             end if;
94         when others => -- Zou nooit mogen voorkomen
95             new_rise <= '0';
96             new_fall <= '0';
97             new_temp <= '0';
98
99             new_r_wait <= '0';
100            new_f_wait <= '0';
101
102            new_state <= clear;
103     end case;
104 end process;
105 end behaviour;

```

### A.1.7. 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;

```

### A.1.8. 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
                        BCD
19           parity_bits   :out std_logic_vector(2 downto 0)); -- De 3 parity bits uit het DCF77
                        signaal
20 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     variable location: natural range 0 to 59;
58
59     begin
60         location := to_integer(unsigned(bits_received));
61         new_bit_storage <= bit_storage;
62
63         case state is
64             when idle =>
65                 if(new_bit = '1') then
66                     new_bits_received <= bits_received;
67                     new_minute <= minute;
68                     new_led <= led;
69                     start_xor <= '0';
70                     new_state <= newbit;
71                     -- Controleer of er een nieuwe minuut gaat beginnen
72                 elsif(count > 32000 and count < 48000) then
73                     -- Initialiseer voor een nieuwe minuut
74                     new_bits_received <= (others => '0');
75
76                     -- Controle of er niet toevallig een seconde is gemist
77                 if(bits_received = 59) then
78                     new_minute <= '1';
79                     new_led <= '1';
80                 else
81                     new_minute <= minute;
82                     new_led <= '0';
83                 end if;
84
85                 start_xor <= '0';
86                 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;

```

### A.1.9. 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, weekday, 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 <= weekday(2) xor weekday(1) xor weekday(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;
76
77                 if (m_par = '0' and h_par = '0' and d_par = '0') then
78                     sync_now <= '1'; -- Parity is correct
79                 else
80                     sync_now <= '0'; -- Parity is niet correct
81                 end if;
82                 new_state <= clear;
83             when others => -- Zou nooit mogen voorkomen
84                 new_m_par <= m_par;
85                 new_h_par <= h_par;
86                 new_d_par <= d_par;
87
88                 sync_now <= '0';
89                 new_state <= clear;

```

```

89         end case;
90     end process;
91 end behaviour;

```

### A.1.10. 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);
81     parity: parity_check port map(clk, reset, start_xor, minuut, uur, weekday, dag, maand,
82                                jaar, par, ready);
83 end structure;

```

### A.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_lhz:         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 gemaakt van de volgende subblokken:
6  -- synctime, klokdeleer 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;
15              reset:         in  std_logic;
16              dcf_in:        in  std_logic;
17              dcf_led:       out std_logic;
18              ready:         out std_logic;
19              minuten:       out std_logic_vector(6 downto 0);
20              uren:          out std_logic_vector(5 downto 0);
21              weekdag:       out std_logic_vector(2 downto 0);
22              dagen:         out std_logic_vector(5 downto 0);
23              maanden:       out std_logic_vector(4 downto 0);
24              jaren:         out std_logic_vector(7 downto 0));
25     end component synctime;
26
27     component klokdelers is
28         port (clk:          in  std_logic;
29              reset:         in  std_logic;
30              clk_lhz:       out std_logic);
31     end component klokdelers;
32
33     component ausy_klok_bcd is
34         port (clk:          in  std_logic;
35              s_clk:         in  std_logic;
36              reset:         in  std_logic;
37              sync_now:      in  std_logic;
38              min_ref:       in  std_logic_vector(6 downto 0);
39              hr_ref:        in  std_logic_vector(5 downto 0);
40              minutes:       out std_logic_vector(6 downto 0);
41              hours:         out std_logic_vector(5 downto 0));
42     end component ausy_klok_bcd;
43
44     signal sync, s_clk: std_logic;
45     signal minuut: std_logic_vector(6 downto 0);
46     signal uur: std_logic_vector(5 downto 0);
47
48 begin
49     date_ready <= sync;
50     clk_lhz <= s_clk;
51
52     sytime: synctime      port map (clk, reset, dcf_in, dcf_led, sync, minuut, uur,
53                                     weekday, day, month, year);
54     divide: klokdelers    port map (clk, reset, s_clk);
55     r_time: ausy_klok_bcd port map (clk, s_clk, reset, sync, minuut, uur, minutes,
56                                     hours);
57
58 end structure;

```

## A.2. VHDL CODE CONTROLLER

### A.2.1. 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;

```

### A.2.2. 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 -- Het blok waar het mooie en slimmen
    onderdelen van de schakeling gedaan worden
8     port (clk          :in    std_logic;
9           reset        :in    std_logic;
10          knoppen      :in    std_logic_vector(3 downto 0);
11          wekdata       :in    std_logic_vector(15 downto 0);
12          enable        :out    std_logic;
13          wekker        :out    std_logic_vector(15 downto 0);
14          menu_signal    :out    std_logic_vector(2 downto 0));
15 end component menu;
16
17 component geheugen is -- 14 bit opslag
18     port (clk          :in    std_logic;
19           reset        :in    std_logic;
20           enable       :in    std_logic;
21           wek_in       :in    std_logic_vector(15 downto 0);
22           wek_out      :out    std_logic_vector(15 downto 0));
23 end component geheugen;
24
25
26 component buff is --De buffer die speciaal gemaakt is voor de menu
    met extra eigenschappen
27     port (clk          :in    std_logic;
28           reset        :in    std_logic;
29           knoppen_in   :in    std_logic_vector(3 downto 0);
30           knoppen_out  :out    std_logic_vector(3 downto 0));
31 end component buff;
32
33 signal knoppen_buff : std_logic_vector(3 downto 0);
34 signal wekdata_men, wekker_men : std_logic_vector(15 downto 0);
35 signal write_enable : std_logic;
36
37 begin
38 buffer_portmap : buff port map (clk, reset, knoppen, knoppen_buff);
39 menu_portmap : menu port map (clk, reset, knoppen_buff, wekdata_men, write_enable, wekker_men,
    menu_state);
40 memory_portmap : geheugen port map (clk, reset, write_enable, wekker_men, wekdata_men);
41 wekker <= wekdata_men;
42
43 end behaviour;

```

### A.2.3. 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;

```

### A.2.4. 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 opgaande 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,
    uren_set, uren_plus, uren_min, minuten_set, minuten_plus, minuten_min);
12 signal state, new_state : fsm_states;
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     end process assign;
24
25     actie_uitvoeren : process(knoppen, wekdata, clk, reset, state) --Voer acties uit
26     begin
27         case state is
28             when rust =>
29                 enable <= '0';
30                 wekker <= wekdata;
31                 menu_signal <= "000";
32
33             when wekker_toggle =>
34                 enable <= '1';
35                 wekker(14 downto 0) <= wekdata(14 downto 0);
36                 wekker(15) <= not wekdata(15);
37                 menu_signal <= "000";
38
39             when wekkertijd =>
40                 enable <= '0';
41                 wekker <= wekdata;
42                 menu_signal <= "101";
43
44             when led =>
45                 enable <= '0';
46                 wekker <= wekdata;
47                 menu_signal <= "011";
48
49             when led_toggle =>
50                 enable <= '1';
51                 wekker(13 downto 0) <= wekdata(13 downto 0);
52                 wekker(14) <= not wekdata(14);
53                 wekker(15) <= wekdata(15);
54                 menu_signal <= "011";
55
56             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
85         else
86             wekker(10 downto 7) <= std_logic_vector(to_unsigned(to_integer(
87                 unsigned(wekdata(10 downto 7))) + 1 , 4)); -- 1 minuut erbij
88             optellen
89             wekker(12 downto 11) <= wekdata(12 downto 11); -- Tientallen blijven
90             constant
91         end if;
92     end if;
93     wekker(15 downto 13) <= wekdata(15 downto 13); -- Af
94     wekker(6 downto 0) <= wekdata(6 downto 0); --Af
95
96 when uren_min =>
97     if wekdata(12 downto 7) = "000000" then
98         wekker(12 downto 7) <= "100011"; --23
99     else
100         if wekdata(10 downto 7) = "0000" then
101             wekker(10 downto 7) <= "1001";
102             wekker(12 downto 11) <= std_logic_vector(to_unsigned(to_integer(
103                 unsigned(wekdata(12 downto 11))) - 1 , 2));
104         else
105             wekker(10 downto 7) <= std_logic_vector(to_unsigned(to_integer(
106                 unsigned(wekdata(10 downto 7))) - 1 , 4));
107             wekker(12 downto 11) <= wekdata(12 downto 11);
108         end if;
109     end if;
110     wekker(15 downto 13) <= wekdata(15 downto 13);
111     wekker(6 downto 0) <= wekdata(6 downto 0);
112     enable <= '1';
113     menu_signal <= "101";
114
115 when minuten_set =>
116     enable <= '0';
117     wekker <= wekdata;
118     menu_signal <= "010";
119
120 when minuten_plus =>
121     enable <= '1';
122     if wekdata(6 downto 0) = "1011001" then --59
123         wekker(6 downto 0) <= "0000000"; --Bij de 59 minuten gaan weer op nieuw
124         beginnen
125     else
126         if wekdata(3 downto 0) = "1001" then --Bij x9 minuten 1 op tellen bij de
127             x en enkele weer terug naar 0
128             wekker(3 downto 0) <= "0000";
129             wekker(6 downto 4) <= std_logic_vector(to_unsigned(to_integer(
130                 unsigned(wekdata(6 downto 4))) + 1 , 3));
131         else
132             wekker(3 downto 0) <= std_logic_vector(to_unsigned(to_integer(
133                 unsigned(wekdata(3 downto 0))) + 1 , 4)); -- 1 minuut erbij
134             optellen
135             wekker(6 downto 4) <= wekdata(6 downto 4); -- Tientallen blijven
136             constant
137         end if;
138     end if;
139     menu_signal <= "111";
140     wekker(15 downto 7) <= wekdata(15 downto 7); --Af
141
142 when minuten_min =>
143     enable <= '1';
144     if wekdata (6 downto 0) = "0000000" then
145         wekker(6 downto 0) <= "1011001"; --59
146     else
147         if wekdata(3 downto 0) = "0000" then --Bij x0 minuten 1 van de tientallen
148             afhalen en de enkele getal op 9 zetten
149             wekker(3 downto 0) <= "1001"; --9

```

```

135         wekker(6 downto 4) <= std_logic_vector(to_unsigned(to_integer(
136             unsigned(wekdata(6 downto 4))) - 1 , 3));
137     else
138         wekker(3 downto 0) <= std_logic_vector(to_unsigned(to_integer(
139             unsigned(wekdata(3 downto 0))) - 1 , 4));
140         wekker(6 downto 4) <= wekdata(6 downto 4);
141     end if;
142     menu_signal <= "111";
143     wekker(15 downto 7) <= wekdata(15 downto 7); --Af
144 end case;
145 end process actie_uitvoeren;
146
147 next_state : process (knoppen, wekdata, clk, reset, state) -- Bepaal nieuwe state
148 begin
149     case state is
150     when rust =>
151         if knoppen(0) = '1' then
152             new_state <= wekkertijd;
153         elsif knoppen(1) = '1' then
154             new_state <= wekker_toggle;
155         else
156             new_state <= rust;
157         end if;
158     when wekker_toggle =>
159         new_state <= rust;
160     when wekkertijd =>
161         if knoppen(0) = '1' then
162             new_state <= rust;
163         elsif knoppen(2) = '1' then
164             new_state <= geluid;
165         elsif knoppen(3) = '1' then
166             new_state <= led;
167         elsif knoppen(1) = '1' then
168             new_state <= uren_set;
169         else
170             new_state <= wekkertijd;
171         end if;
172     when led =>
173         if knoppen(0) = '1' then
174             new_state <= rust;
175         elsif knoppen(2) = '1' then
176             new_state <= wekkertijd;
177         elsif knoppen(3) = '1' then
178             new_state <= geluid;
179         elsif knoppen(1) = '1' then
180             new_state <= led_toggle;
181         else
182             new_state <= led;
183         end if;
184     when led_toggle =>
185         new_state <= led;
186     when geluid =>
187         if knoppen(0) = '1' then
188             new_state <= rust;
189         elsif knoppen(2) = '1' then
190             new_state <= led;
191         elsif knoppen(3) = '1' then
192             new_state <= wekkertijd;
193         elsif knoppen(1) = '1' then
194             new_state <= geluid_toggle;
195         else
196             new_state <= geluid;
197         end if;
198     when geluid_toggle =>
199         new_state <= geluid;
200     end if;
201 end process;
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;

```

### A.2.5. 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 entity;

```

### A.2.6. 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;

```

### A.2.7. 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;

```

### A.2.8. 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
   onverwacht gedrag
4  library IEEE;
5  use IEEE.std_logic_1164.ALL;
6  use IEEE.numeric_std.all;
7
8  architecture behaviour of buff is
9      type fsm_states is (rust, one, zero);
10     signal state, new_state : fsm_states;
11     signal knoppen_temp : std_logic_vector(3 downto 0);
12 begin
13     assign : process (clk, reset) --Daadwerkelijk alles toekennen
14     begin
15         if rising_edge(clk) then
16             if reset = '0' then
17                 state <= new_state;
18             else
19                 state <= rust;
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

```



```

39         else
40             new_state <= state;
41         end if;
42     when one =>
43         new_state <= zero;
44         knoppen_temp <= "0000";
45     when others =>
46         new_state <= rust;
47         knoppen_temp <= "0000";
48     end case;
49 end process actie_uitvoeren;
50 end behaviour;

```

## A.3. TESTBENCHS VOOR DE CONTROLLER

### A.3.1. 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;

```

### A.3.2. TESTBENCH VHDL MENU

```

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

```

```

23 --signal uren          :    std_logic_vector (5 downto 0);
24 --signal minute       :    std_logic_vector (6 downto 0);
25 signal uren_dubdle     :    std_logic_vector (1 downto 0);
26 signal minuten_duble   :    std_logic_vector (2 downto 0);
27
28 begin
29     clk      <=    '1' after 0 ns,
30               '0' after 20 ns when clk /= '0' else '1' after 20 ns;
31
32     reset    <=    '1' after 0 ns,          --knoppen(0) = menu;
33               '0' after 62 ns;             --knoppen(1) = set;
34                                           --knoppen(2) = up;
35
36     knoppen <=  "0000" after 0 ns, --knoppen(3) = down
37               "0010" after 68 ns, --rust -> wekker_toggle
38               "0010" after 108 ns, --wekker_toggle -> rust
39               "0001" after 148 ns, --rust -> wekkertijd
40               "0001" after 188 ns, --wekkertijd -> rust
41               "0001" after 228 ns, --rust -> wekkertijd
42               "0010" after 268 ns, --wekkertijd -> uren_set
43               "0100" after 308 ns, --uren_set -> uren_plus
44               "0000" after 348 ns, --uren_plus -> uren_set
45               "1000" after 388 ns, --uren_set -> uren_min
46               "0000" after 428 ns, --uren_min -> uren_set
47               "0001" after 468 ns, --uren_set -> rust
48               "0001" after 508 ns, --rust -> wekkertijd
49               "0010" after 548 ns, --wekkertijd -> uren_set
50               "0010" after 588 ns, --uren_set -> minuten_set
51               "0100" after 628 ns, --minuten_set -> minuten_plus
52               "0000" after 668 ns, --minuten_plus -> minuten_set
53               "1000" after 708 ns, --minuten_set -> minuten_min
54               "0000" after 748 ns, --minuten_min -> minuten_set
55               "0001" after 788 ns, --minuten_set -> rust
56               "0001" after 828 ns, --rust -> wekkertijd
57               "0010" after 868 ns, --wekkertijd -> uren_set
58               "0010" after 908 ns, --uren_set -> minuten_set
59               "0010" after 948 ns, --minuten_set -> wekkertijd EIGENLIJK GAAT DIT NAAR RUST TOE
60               "0001" after 988 ns, --rust -> wekkertijd
61               "1000" after 1028 ns, --wekkertijd -> led
62               "0001" after 1068 ns, --led -> rust
63               "0001" after 1108 ns, --rust -> wekkertijd
64               "0100" after 1148 ns, --wekkertijd -> geluid
65               "0100" after 1188 ns, --geluid -> led
66               "0100" after 1228 ns, --led -> wekkertijd
67               "1000" after 1268 ns, --wekkertijd -> led
68               "1000" after 1308 ns, --led -> geluid
69               "1000" after 1348 ns, --geluid -> wekkertijd
70               "1000" after 1388 ns, --wekkertijd -> led
71               "0010" after 1428 ns, --led -> led_toggle
72               "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 <= "0000100001000000" after 20 ns;
83
84
85     menu_pm: menu port map(clk, reset, knoppen, wekdata, enable, wekker, menu_signal); --de
86     daadwerkelijke port map
87 end architecture;

```

### A.3.3. 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                  "0000000000010000" after 630 ns,
48                  "0000000000010001" after 680 ns,
49                  "0000000000010010" after 735 ns,
50                  "0000000000010111" after 779 ns;
51
52      geheugen_pm: geheugen port map (clk,reset,enable,wek_in,wek_out);
53  end behaviour;

```

### A.3.4. 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,knopjes      :   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;

```

## A.4. VHDL CODE VAN HET ALARM

### A.4.1. 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;

```

### A.4.2. 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 begin
11     lbl1: process (clk)
12     begin
13         if (clk'event and clk = '1') then
14             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                 state <= new_state;
34             end if;
35         end if;
36     end process;
37 end behaviour;

```

```

32     end if;
33 end process;
34 lbl2: process (state, alarm_min, alarm_uur, wekker_uur, wekker_min, tijd_min, tijd_uur)
35 begin
36     case state is
37     when steady =>
38         geluid <= '0';
39         licht <= '0';
40         if (alarm_min = tijd_min) and (alarm_uur = tijd_uur) then
41             new_state <= start;
42         else
43             new_state <= steady;
44         end if;
45     when start =>
46         geluid <= '0';
47         licht <= '1';
48         if (wekker_uur = tijd_uur) and (wekker_min = tijd_min) then
49             new_state <= final;
50         else
51             new_state <= start;
52         end if;
53     when final =>
54         geluid <= '1';
55         licht <= '1';
56         new_state <= final;
57     when others =>
58         geluid <= '0';
59         licht <= '1';
60         new_state <= state;
61     end case;
62 end process;
63 end behaviour;

```

#### A.4.3. 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;

```

#### A.4.4. 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);
21     pwm_1 : pwm port map (clk => clk, reset => reset, length => length, pwm_signal =>
22                           pwm_signal);

```

22 **end** behaviour;

#### A.4.5. 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;
```

#### A.4.6. 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;

```

#### A.4.7. 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;

```

#### A.4.8. 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 =>

```

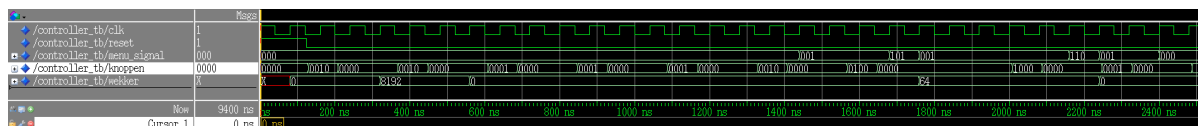


```
46         pwm_signal <= '0';
47         new_counter <= counter;
48         new_state <= laag;
49     end case;
50 end process;
51 end behaviour;
```

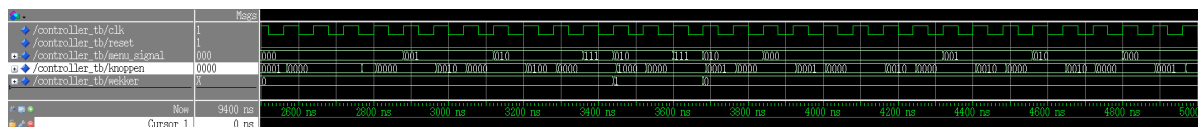
# B

## SIMULATIES RESULTATEN VAN DE CONTROLLER

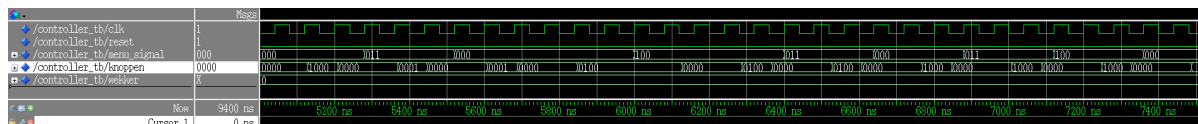
### B.1. BEHAVIORAL SIMULATIE



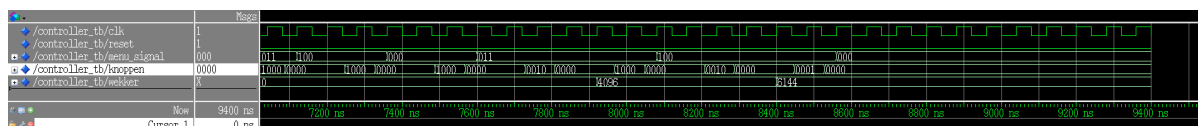
Figuur B.1: Simulatie van 0 tot 2500ns



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

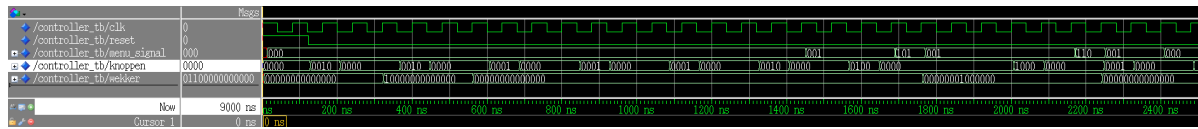


Figuur B.3: Simulatie van 5000ns tot 7500ns

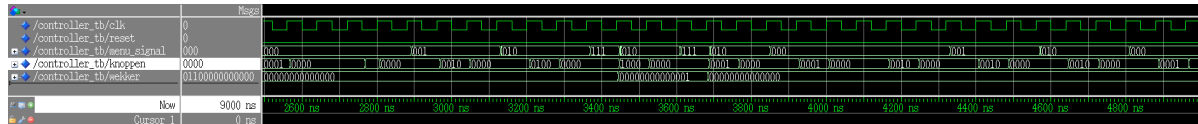


Figuur B.4: Simulatie van 7500ns tot het einde

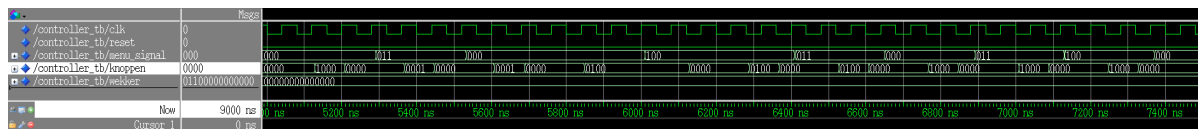
## B.2. SYNTHESIZE SIMULATIE



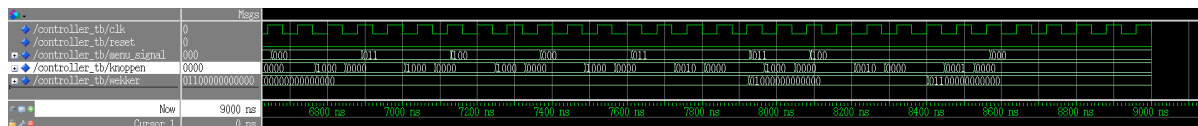
Figuur B.5: Simulatie van 0 tot 2500ns



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

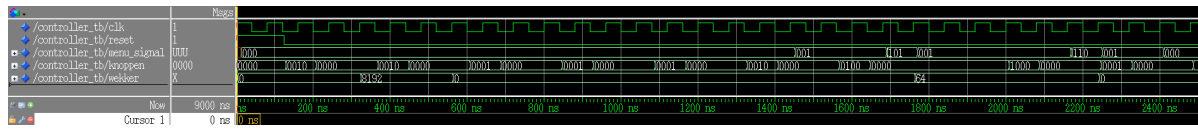


Figuur B.7: Simulatie van 5000ns tot 7500ns

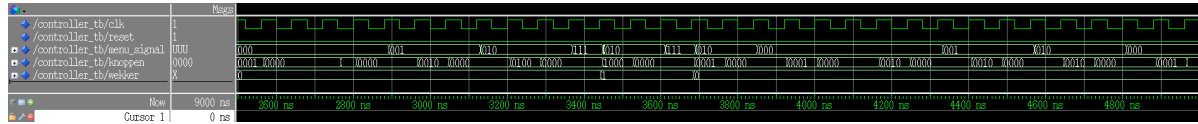


Figuur B.8: Simulatie van 7500ns tot het einde

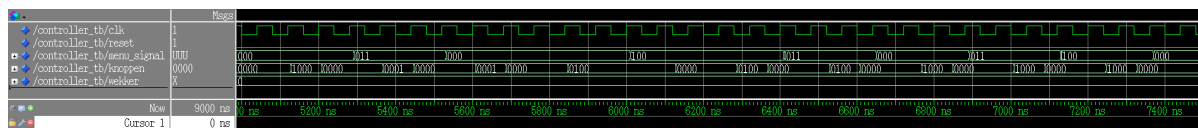
## B.3. EXTRACTED SIMULATIE



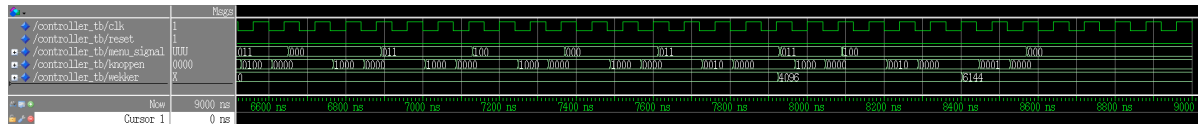
Figuur B.9: Simulatie van 0 tot 2500ns



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

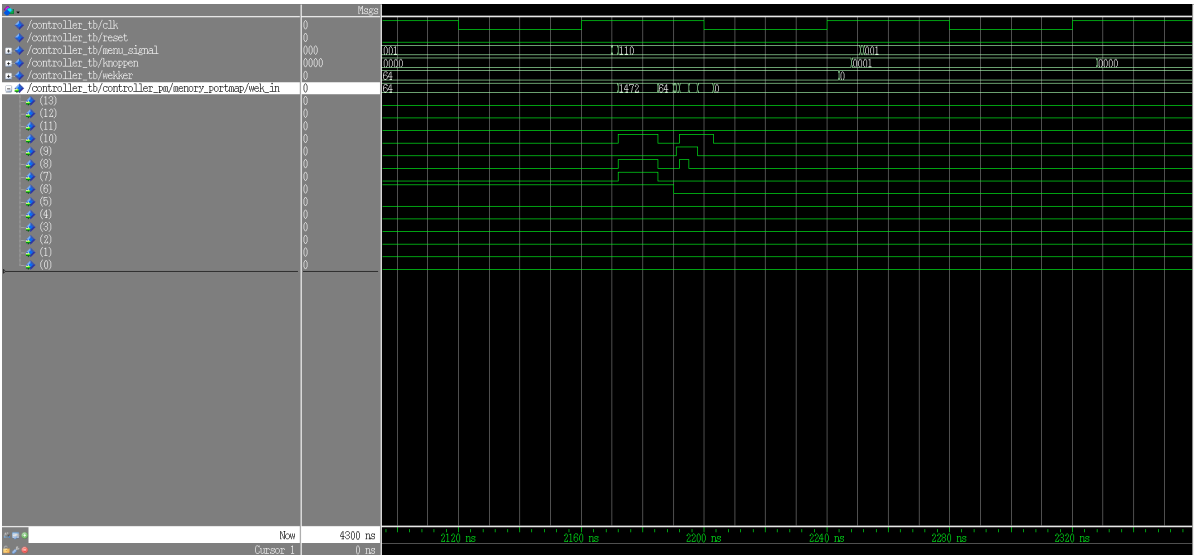


Figuur B.11: Simulatie van 5000ns tot 7500ns



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

B.4. TIMING



Figuur B.13: Timing problemen

# BIBLIOGRAFIE

- [1] Physikalisch-Technische Bundesanstalt (PTB), [Dcf77 zeitkode](#), (08-12-2014).