EPO-3

Extreme Winterslaap Interrupter
Midterm report

Projectgroep A1

Technische Universiteit Delft

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

TU Delft Delft University of Technology

Challenge the future

INHOUDSOPGAVE

1

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 opstakels die overwonnen moesten worden zijn het ontvangen en verwerken van het DCF-77 signaal, het aansturen van het licht, het aansturen van het geluid en het aansturen van een LCD schermpje. In het verslag is te vinden hoe al deze subsystemen zijn ontworpen en uitgewerkt.

2

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 gesyntetiseerd worden. Als aan het eind van het project van het hele systeem een lay-out gemaakt is, kan het systeem op een chip gezet worden.

3

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 scheidingsteken 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
- DCF_debug, bit om aan te geven of er een DCF-signaal ontvangen wordt.

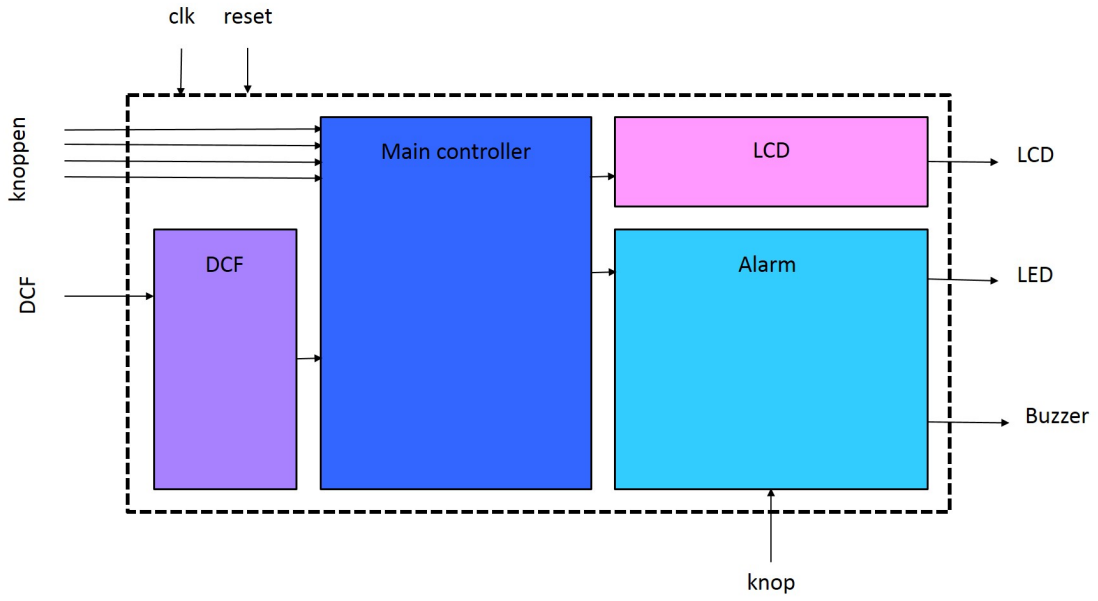
4

SYSTEEM OVERZICHT EN ONTWERP

Het systeem is opgedeeld in vier blokken:

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

In figuur 4.1 is te zien welke ingangs- en uitgangssignalen het systeem in en uit gaan en hoe de blokken elkaar aansturen.



Figuur 4.1: 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-signaal gegenereerd wordt wat naar een LED gaat.
De LCD controller zorgt dat de datum, tijd, ingestelde alarmtijd en de veranderingen in het menu op de LCD zichtbaar zijn. Er wordt een LCD scherm gebruikt waar de pixels afzonderlijk van elkaar aangestuurd worden. Tussen de chip en het scherm zit nog een microcontroller, waarin de karakters zijn opgeslagen, dit zou namelijk te groot zijn om op de chip te regelen.

5

DCF CONTROLLER

5.1. INLEIDING

De basis van onze wekker wordt gelegd door een klok. Uit het onderdeel, genaamd DCF-controller, komt verschillende data, als de tijd, de datum en het weeknummer. Van de tijd uitgang wordt verwacht dat deze gesynchroniseerd met het DCF-signaal is, maar mocht het signaal uitvallen moet de tijd door blijven tellen.

5.2. SPECIFICATIES

5.2.1. INGANGEN

Dit onderdeel maakt gebruik van de volgende ingangen:

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

5.2.2. UITGANGEN

Dit onderdeel heeft de volgende uitgangen:

- Uren, een binaire vector van 6 bits.
- minuten, een binaire 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.

5.3. GEDRAG

Een van de eigenschappen van deze klok zal zijn dat hij gesynchroniseerd wordt met een zogenaamd DCF-signaal. Dit is een signaal dat in duitsland verzonden wordt en allerlei informatie bevat, als de actuele tijd en de datum. Wij zullen meerdere van deze elementen gebruiken in onze wekker. Al deze data wordt verzonden door middel van een pulssignaal. Vanuit Duitland 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, gevolgt door een seconde 'rust', dat elke minuut opnieuw verzonden wordt. De uren en minuten van dit signaal zullen gebruikt worden om de tijd van een interne klok bij te werken. Daarnaast zal de dag van de week, de dag van de maand, de maand en het jaar doorgegeven naar de andere onderdelen van de wekker doorgegeven.

6

MAIN CONTROLLER

6.1. 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.

6.2. SPECIFICATIES

6.2.1. 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

6.2.2. 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 uberhaupt 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 6.1 wat voor informatie te vinden is in de uitgangen van de controller.

Uitgang	Informatie over wat in de uitgang te vinden is
wekdata	De huidige info over de wekker instellingen uit geheugen wekdata[5 down to 0] daarin staan de minuten wekdata[10 down to 6] daarin staan de uren wekdata[11] geluid bit wekdata[12] led bit wekdata[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

Tabel 6.1: Uitgangen van de controller

6.2.3. 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 geld voor het geluid.
Dit alles moet zo gebruiksvriendelijk mogelijk gebeuren.

6.3. FUNCTIONALITEIT

6.3.1. FSM

In fig. 6.1 staat de gemaakte fsm en in tabel 6.2 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 = menu
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 = menu
Geluid	enable =’0’ wekker=wekdata of een mooi getal code menu = 100
Geluid toggle	enable =’1’ wekker[10 down to 0]=wekdata[10 down to 0] wekker[11] = niet wekdata[11] wekker[13 downto 12] = wekdata[13 downto 12] menu = menu

Uren set	enable =’0’ wekker=wekdata menu = 001
Uren plus	enable =’1’ wekker=wekker+1 menu =menu
Uren min	enable =’1’ wekker=wekdata-1 menu = menu
Minuten set	enable =’0’ wekker=wekdata menu = 010
Minuten plus	enable =’1’ Wektijd=wekdata+1 menu = menu
Minuten min	enable =’1’ wekker=wekdata-1 menu = menu

Tabel 6.2: Uitgangen binnen de state van de controller

6.3.2. VHDL CODE

De code voor de controller van de wekker is te vinden in appendix A. 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.1 en A.2.
- Het menu, hierin zit de echte logica verwerkt. Deze is te vinden in appendices A.3 en A.4.
- Het gebruikte geheugen element voor de opslag van 14 bits, te vinden in appendices A.5 en A.6.
- De gebruikte buffer is te vinden in appendices A.7 en A.8. De buffer regelt het ingangssignaal, en zorgt ervoor dat er maar 1 klokperiode lang een hoog signaal gelezen word.

Voor het testen van de code zijn er testbenches gemaakt welke te vind zijn in appendices B.1 tot B.4.

6.4. 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*.

6.5. SIMULATIE

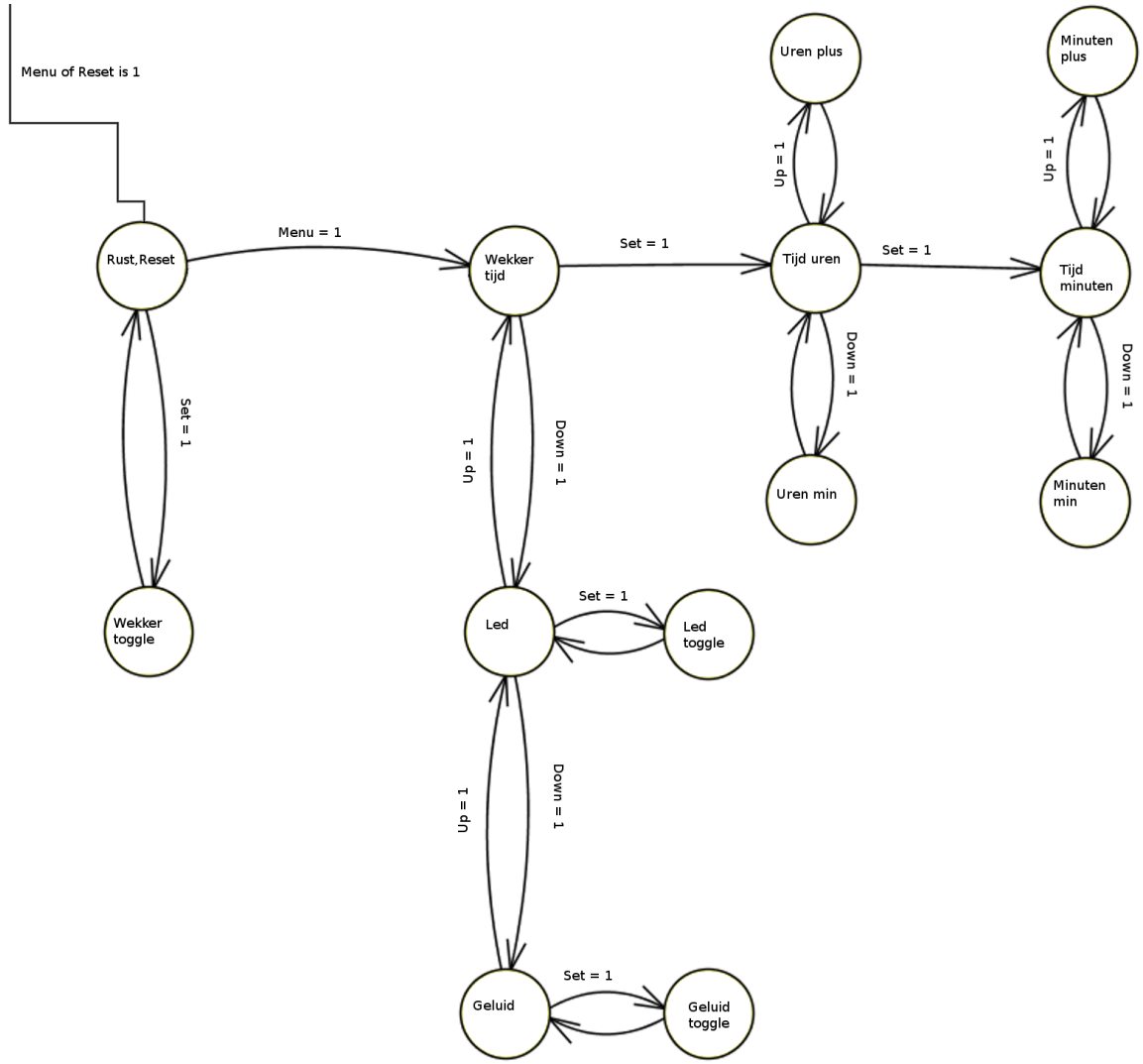
De resultaten van de simulatie staan in appendix C. De testbench is te lang om in een keer weer te geven daarom is deze op geknipt in vier stukken. De testbench die gemaakt is voor de simulatie staat in appendix B.1

6.6. RESULTATEN

Van fig. C.1 tot en met fig. C.12 is te zien dat iedere simulatie tot hetzelde resultaat leid en daarmee succesvol is. De minimale klokperiode kan afgelezen worden aan de hand van fig. C.13. Hieruit is op te makken dat deze 60ns is.

6.6.1. 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 transitoren 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 geimplementeerd. Vlak



Figuur 6.1: 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.

7

ALARM

7.1. 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.

7.2. SPECIFICATIES

7.2.1. 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 ingestels in de main controller.
- Sec, seconde signaal gegenereerd in de DCF controller.
- Knop, alarm uitschakelen.

7.2.2. UITGANGEN

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

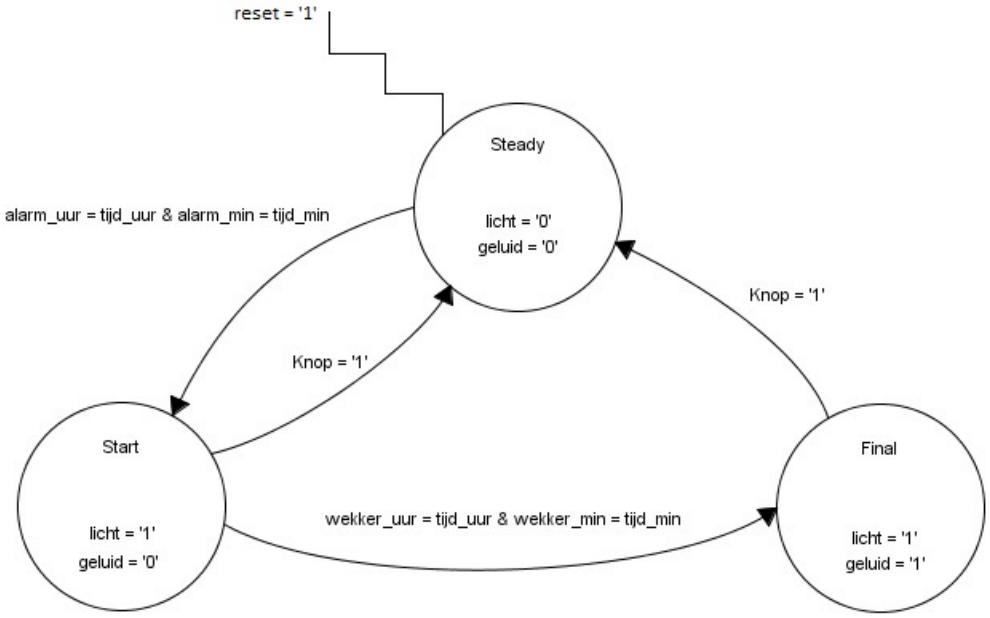
7.2.3. GEDRAG

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 "lenght"kleiner. Deze begon op 64 en wordt vergeleken met een andere counter die elke klokflank telt, tot 64. Als de counter groter of gelijk is aan "length"dan is het pwm-signaal hoog. Als 15 minuten zijn verstreken na het aangaan van de led, dus de ingestelde tijd is gelijk aan de huidige tijd, brandt de led op z'n felst. Ook zal dan een "geluid"signaal naar '1' gaan. Dit blijft zo totdat de knop wordt ingedrukt of alles wordt gereset.

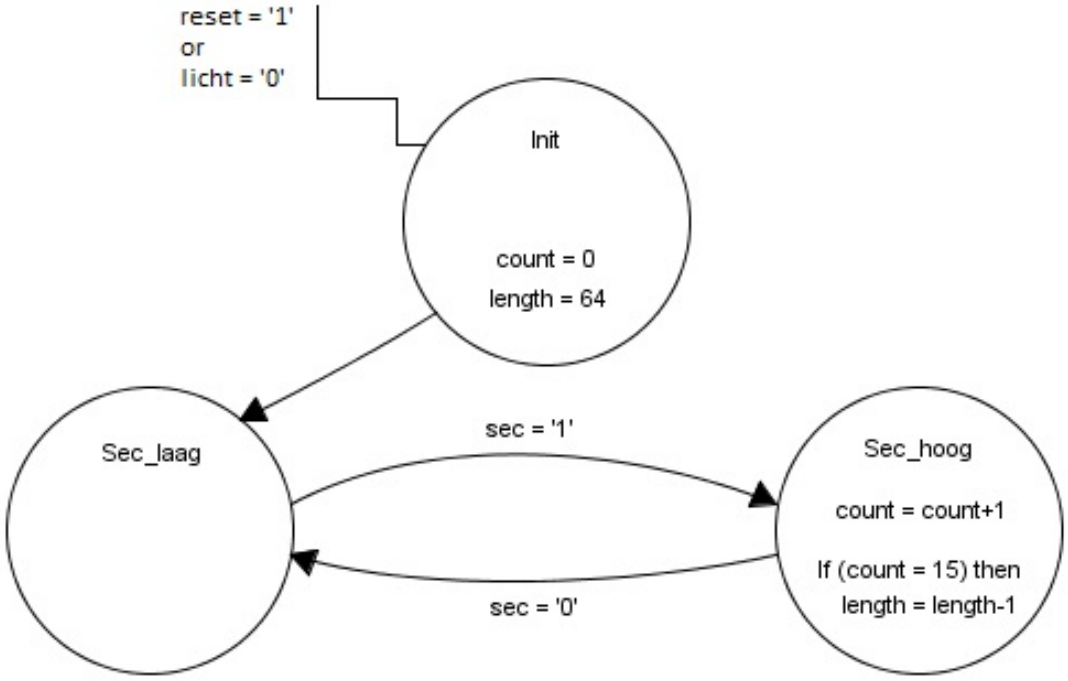
7.3. FUNCTIONALITEIT

7.3.1. FSM

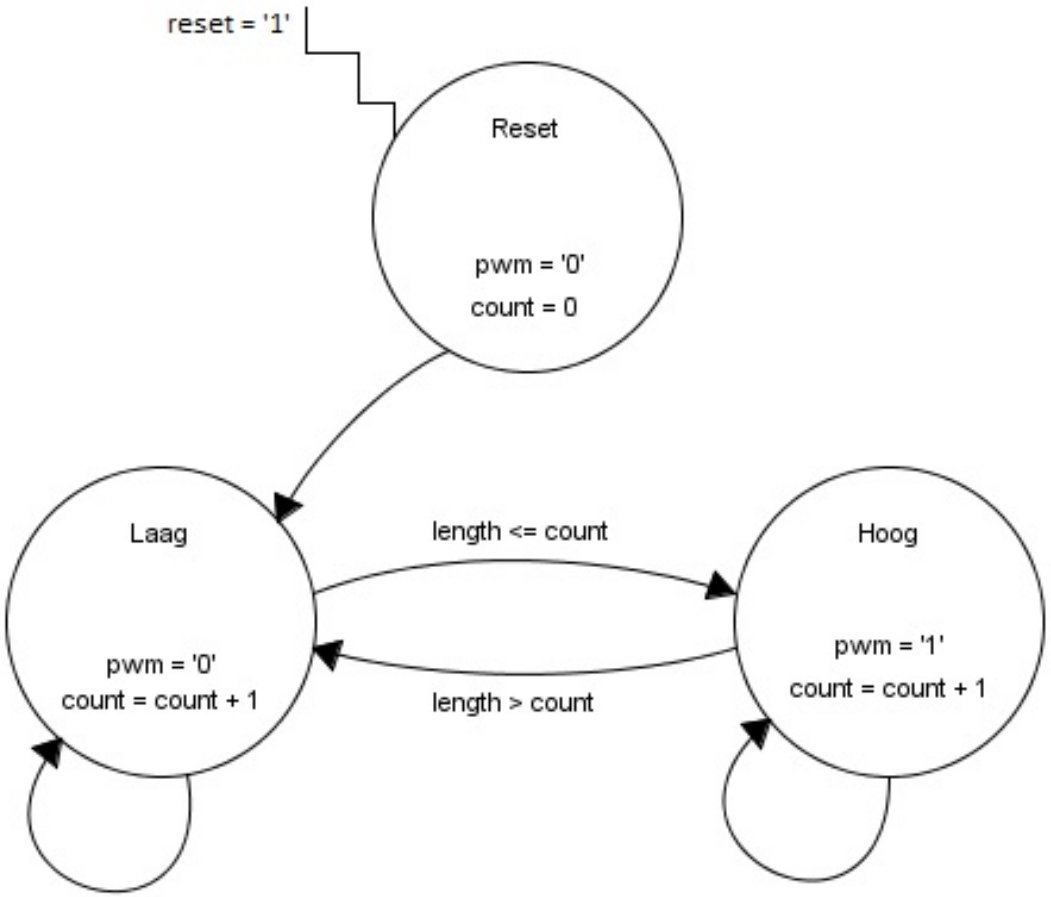
FSM van “alarm-compare", het gedeelte waar de tijd vergeleken wordt met de ingestelde wekker tijd.



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



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



7.3.2. CODE

De code voor het alarm is opgedeeld in 3 stukken:

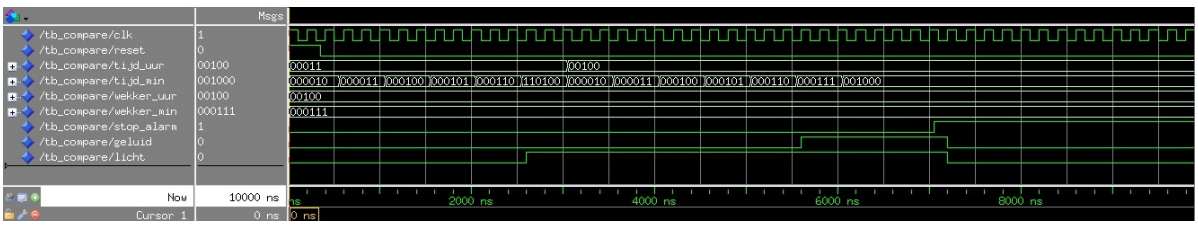
- alarm-compare
- alarm-counter
- alarm-pwm

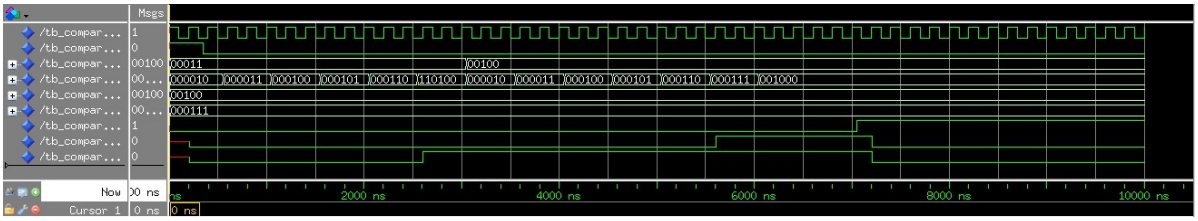
Alarm-counter en alarm-pwm zijn onderdeel van een top entity, alarm. Omdat het nog niet zeker is waar alarm-compare geplaatst gaat worden op de chip is die daar niet bij inbegrepen. De code voor het alarm is te vinden in appendix D.

- De entity en behavioural van alarm compare is te vinden in appendices D.1 en D.2.
- De top entity en port map van alarm is te vinden in appendices D.3 en D.4.
- De entity en behavioural van alarm-counter is te vinden in appendices D.5 en D.6.
- De entity en behavioural van alarm-pwm is te vinden in appendices D.7 en D.8.

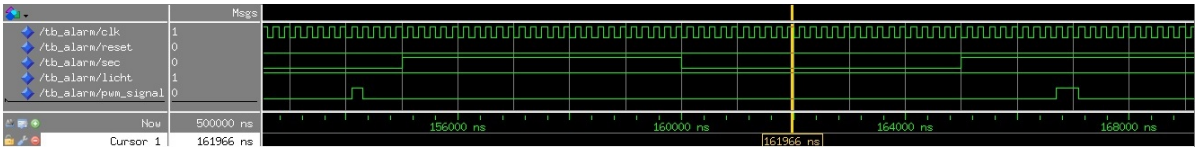
7.4. RESULTATEN

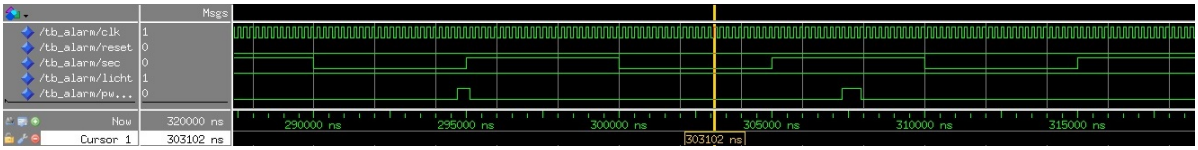
Alle onderdelen werken in de simulaties. Zowel de simulatie van de behaviour als van de extracted vhdl. Onderstaande afbeeldingen zijn de resultaten van de simulaties, eerst die van de behaviour daarna van de extracted. De eerste 2 afbeedingen zijn van “alarm-compare", de andere 2 van “alarm-pwm".





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.

8

LCD CONTROLLER

8.1. INLEIDING
8.2. SPECIFICATIES
8.2.1. 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;

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

8.2.3. GEDRAG
8.3. FUNCTIONALITEIT
8.3.1. FSM
8.3.2. VHDL CODE
8.4. TESTEN
8.5. SIMULATIE
8.6. RESULTATEN
8.6.1. CONCLUSIE EN DISCUSSIE

9

RESULTS FOR TOTAL DESIGN

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

10

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.

10.1. 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.

10.2. 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.

11

VOORTGANG VAN HET PROJECT

11.1. 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 geimplementeerd 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.

11.2. WERKVERDELING

11.2.1. 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.

11.2.2. 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.

11.3. 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.

11.4. 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.

12

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.

A

VHDL CODE VAN DE CONTROLLER

A.1. TOP LEVEL ENTITY

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  entity controller is
5     port(clk    :in    std_logic;
6          reset  :in    std_logic;
7          knoppen:in    std_logic_vector(3 downto 0);
8          wekker :out   std_logic_vector(13 downto 0);
9          menu_state   :out   std_logic_vector(2 downto 0));
10 end controller;
```

A.2. BEHAVIOURAL VHDL CODE CONTROLLER

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  architecture behaviour of controller is
5  component menu is                                    -- Het blok waar het mooie en slimmen
       onderdelen van de schakeling gedaan worden
6      port(clk                :in    std_logic;
7          reset        :in    std_logic;
8          knoppen      :in    std_logic_vector(3 downto 0);
9          wekdata      :in    std_logic_vector(13 downto 0);
10         enable           :out   std_logic;
11         wekker           :out         std_logic_vector(13 downto 0);
12         menu_signal          :out         std_logic_vector(2 downto 0));
13 end component menu;
14
15 component geheugen is                              -- 14 bit opslag
16     port(clk    :in    std_logic;
17         reset  :in    std_logic;
18         enable :in    std_logic;
19         wek_in :in    std_logic_vector(13 downto 0);
20         wek_out:out   std_logic_vector(13 downto 0));
21 end component geheugen;
22
23
24 component buff is                            --De buffer die speciaal gemaakt is voor de menu
       met extra eigenschappen
25     port(clk        :in     std_logic;
26         reset        :in     std_logic;
27         knoppen      :in     std_logic_vector(3 downto 0);
28         knopjes      :out    std_logic_vector(3 downto 0));
29 end component buff;
30
31 signal knoppen_buff : std_logic_vector(3 downto 0);
32 --signal menu_state         : std_logic_vector(2 downto 0);
```

```
33 signal wekdata_men,wekker_men : std_logic_vector(13 downto 0);
34 signal write_enable : std_logic;
35
36 begin
37 buffer_portmap : buff port map (clk,reset,knoppen,knoppen_buff);
38 menu_portmap : menu port map (clk,reset,knoppen_buff,wekdata_men,write_enable,wekker_men,
       menu_state);
39 menory_portmap: geheugen port map (clk,reset,write_enable,wekker_men,wekdata_men);
40 wekker <= wekdata_men;
41
42 end behaviour;
```

A.3. MENU ENTITY

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use IEEE.Numeric_Std.all;
4
5  entity menu is
6     port(clk              :in    std_logic;
7          reset       :in    std_logic;
8          knoppen     :in    std_logic_vector(3 downto 0);
9          wekdata     :in    std_logic_vector(13 downto 0);
10         enable           :out   std_logic;
11         wekker           :out         std_logic_vector(13 downto 0);
12         menu_signal          :out         std_logic_vector(2 downto 0));
13 end menu;
```

A.4. BEHAVIOURAL VHDL CODE MENU

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use IEEE.numeric_Std.all;
4
5  architecture behaviour of menu is
6  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);
7  signal state, new_state : fsm_states;
8  signal uren_tmp : std_logic_vector (4 downto 0);
9  signal minuten_tmp : std_logic_vector (5 downto 0);
10 begin
11     assign : process(clk, reset) --Daadwerkelijk alles toekennen
12     begin
13         if rising_edge(clk) then
14             if reset = '0' then
15                 state <= new_state;
16             else
17                 state <= rust;
18             end if;
19         end if;
20     end process assign;
21
22     actie_uitvoeren : process(knoppen, wekdata, clk, reset, state) --Voer acties uit
23     begin
24         case state is
25             when rust =>
26                 enable <= '0';
27                 wekker <= wekdata;
28                 menu_signal <= "000";
29
30             when wekker_toggle =>
31                 enable <= '1';
32                 wekker(12 downto 0) <= wekdata(12 downto 0);
33                 wekker(13) <= not wekdata(13);
34                 menu_signal <= "000";
35
36             when wekkertijd =>
37                 enable <= '0';
38                 wekker <= wekdata;
39                 menu_signal <= "000";
```

```
40
41              when led =>
42                  enable <= '0';
43                  wekker <= wekdata;
44                  menu_signal <= "011";
45
46              when led_toggle =>
47                  enable <= '1';
48                  wekker(11 downto 0) <= wekdata(11 downto 0);
49                  wekker(12) <= not wekdata(12);
50                  wekker(13) <= wekdata(13);
51                  menu_signal <= "011";
52
53              when geluid =>
54                  enable <= '0';
55                  wekker <= wekdata;
56                  menu_signal <= "100";
57
58              when geluid_toggle =>
59                  enable <= '1';
60                  wekker(10 downto 0) <= wekdata(10 downto 0);
61                  wekker(11) <= not wekdata(11);
62                  wekker(13 downto 12) <= wekdata(13 downto 12);
63                  menu_signal <= "100";
64
65              when uren_set =>
66                  enable <= '0';
67                  wekker <= wekdata;
68                  menu_signal <= "001";
69
70              when uren_plus =>
71                  enable <= '1';
72                  menu_signal <= "101"; --let op dit is alleen gedaan voor testen
73                  if (to_integer(unsigned(wekdata(10 downto 6)))) < 23 then
74                      wekker(10 downto 6) <= std_logic_vector(to_unsigned(to_integer(unsigned(
                            wekdata(10 downto 6))) + 1, 5));
75                  else
76                      wekker(10 downto 6) <= "00000";
77                  end if;
78                  wekker(13 downto 11) <= wekdata(13 downto 11);
79                  wekker(5 downto 0) <= wekdata(5 downto 0);
80
81              when uren_min =>
82                  enable <= '1';
83                  menu_signal <= "110"; --let op dit is alleen gedaan voor testen
84                  if (to_integer(unsigned(wekdata(10 downto 6)))) > 0 then
85                      wekker(10 downto 6) <= std_logic_vector(to_unsigned(to_integer(unsigned(
                            wekdata(10 downto 6))) - 1, 5));
86                  else
87                      wekker(10 downto 6) <= "10111";
88                  end if;
89                  wekker(13 downto 11) <= wekdata(13 downto 11);
90                  wekker(5 downto 0) <= wekdata(5 downto 0);
91
92              when minuten_set =>
93                  enable <= '0';
94                  wekker <= wekdata;
95                  menu_signal <= "010";
96
97              when minuten_plus =>
98                  enable <= '1';
99                  if (to_integer(unsigned(wekdata(5 downto 0)))) < 59 then
100                     wekker(5 downto 0) <= std_logic_vector(to_unsigned(to_integer(unsigned(
                            wekdata(5 downto 0))) + 1, 6));
101                 else
102                     wekker(5 downto 0) <= "000000";
103                 end if;
104                 menu_signal <= "111"; --let op dit is alleen gedaan voor testen
105                 wekker(13 downto 6) <= wekdata(13 downto 6);
106
107             when minuten_min =>
```

```
108                 enable <= '1';
109                 if (to_integer(unsigned(wekdata(5 downto 0)))) > 0 then
110                     wekker(5 downto 0) <= std_logic_vector(to_unsigned(to_integer(unsigned(
                            wekdata(5 downto 0))) - 1, 6));
111                 else
112                     wekker(5 downto 0) <= "111011";
113                 end if;
114                 menu_signal <= "111"; --let op dit is alleen gedaan voor testen
115                 wekker(13 downto 6) <= wekdata(13 downto 6);
116         end case;
117     end process actie_uitvoeren;
118 
119     next_state : process (knoppen, wekdata, clk, reset, state) -- Bepaal nieuwe state
120     begin
121         case state is
122             when rust =>
123                 if knoppen(0) = '1' then
124                     new_state <= wekkertijd;
125                 elsif knoppen(1) = '1' then
126                     new_state <= wekker_toggle;
127                 else
128                     new_state <= rust;
129                 end if;
130 
131             when wekker_toggle =>
132                 new_state <= rust;
133 
134             when wekkertijd =>
135                 if knoppen(0) = '1' then
136                     new_state <= rust;
137                 elsif knoppen(2) = '1' then
138                     new_state <= geluid;
139                 elsif knoppen(3) = '1' then
140                     new_state <= led;
141                 elsif knoppen(1) = '1' then
142                     new_state <= uren_set;
143                 else
144                     new_state <= wekkertijd;
145                 end if;
146 
147             when led =>
148                 if knoppen(0) = '1' then
149                     new_state <= rust;
150                 elsif knoppen(2) = '1' then
151                     new_state <= wekkertijd;
152                 elsif knoppen(3) = '1' then
153                     new_state <= geluid;
154                 elsif knoppen(1) = '1' then
155                     new_state <= led_toggle;
156                 else
157                     new_state <= led;
158                 end if;
159 
160             when led_toggle =>
161                 new_state <= led;
162 
163             when geluid =>
164                 if knoppen(0) = '1' then
165                     new_state <= rust;
166                 elsif knoppen(2) = '1' then
167                     new_state <= led;
168                 elsif knoppen(3) = '1' then
169                     new_state <= wekkertijd;
170                 elsif knoppen(1) = '1' then
171                     new_state <= geluid_toggle;
172                 else
173                     new_state <= geluid;
174                 end if;
175 
176             when geluid_toggle =>
177                 new_state <= geluid;
```

```
178
179             when uren_set =>
180                 if knoppen(0) = '1' then
181                     new_state <= rust;
182                 elsif knoppen(2) = '1' then
183                     new_state <= uren_plus;
184                 elsif knoppen(3) = '1' then
185                     new_state <= uren_min;
186                 elsif knoppen(1) = '1' then
187                     new_state <= minuten_set;
188                 else
189                     new_state <= uren_set;
190                 end if;
191
192             when uren_plus =>
193                 new_state <= uren_set;
194
195             when uren_min =>
196                 new_state <= uren_set;
197
198             when minuten_set =>
199                 if knoppen(0) = '1' then
200                     new_state <= rust;
201                 elsif knoppen(2) = '1' then
202                     new_state <= minuten_plus;
203                 elsif knoppen(3) = '1' then
204                     new_state <= minuten_min;
205                 elsif knoppen(1) = '1' then
206                     new_state <= rust;
207                 else
208                     new_state <= minuten_set;
209                 end if;
210
211             when minuten_plus =>
212                 new_state <= minuten_set;
213
214             when minuten_min =>
215                 new_state <= minuten_set;
216         end case;
217     end process next_state;
218 end behaviour;
```

A.5. MEMORY

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  entity geheugen is
5     port(clk    :in    std_logic;
6          reset  :in    std_logic;
7          enable :in    std_logic;
8          wek_in :in    std_logic_vector(13 downto 0);
9          wek_out:out   std_logic_vector(13 downto 0));
10 end geheugen;
```

A.6. BEHAVIOURAL VHDL MEMORY

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  architecture behaviour of geheugen is
5  signal wek_opslag,wek_temp : std_logic_vector(13 downto 0 );
6  begin
7  assign : process(clk,reset,wek_temp,wek_in)
8  begin
9      if rising_edge(clk) then
10         if reset = '1' then
11             wek_temp<= (others => '0');
12         else
13             if enable = '1' then
```

```
14                  wek_temp <= wek_in;
15              else
16                  wek_temp <= wek_temp;
17              end if;
18          end if;
19      end if;
20      wek_out<=wek_temp;
21  end process assign;
22  end behaviour;
```

A.7. ENTITY BUFFER

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

A.8. BEHAVIOURAL VHDL BUFFER

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use IEEE.numeric_Std.all;
4
5  architecture behaviour of buff is
6  type fsm_states is (rust, one, zero);
7  signal state, new_state : fsm_states;
8  signal knopjes_temp : std_logic_vector(3 downto 0);
9  begin
10     assign : process(clk, reset) --Daadwerkelijk alles toekennen
11     begin
12         if rising_edge(clk) then
13             if reset = '0' then
14                 state <= new_state;
15             else
16                 state <= rust;
17             end if;
18             knopjes <= knopjes_temp;
19         end if;
20     end process assign;
21     actie_uitvoeren : process(knoppen,clk, reset, state) --Voer acties uit
22     begin
23         case state is
24             when rust =>
25                 --knopjes <= "0000";
26                 if ((knoppen(0) = '1' xor knoppen(1) = '1') xor (knoppen(2) = '1' xor knoppen
                       (3) = '1')) then
27                     new_state <= one;
28                     knopjes_temp <= knoppen;
29                 else
30                     new_state <= state;
31                     knopjes_temp <= "0000";
32                 end if;
33             when zero =>
34                 --knopjes <= "0000";
35                 knopjes_temp <= "0000";
36                 if ((knoppen(0) = '0' and knoppen(1) = '0') and (knoppen(2) = '0' and knoppen
                       (3) = '0')) then
37                     new_state <= rust;
38                 else
39                     new_state <= state;
40                 end if;
41             when one =>
42                 --knopjes <= knopjes_temp;
43                 --knopjes_temp <= knopjes_temp;
44                 new_state <= zero;
```

```
45                  knopjes_temp<="0000";
46          end case;
47
48      end process actie_uitvoeren;
49  end behaviour;
```

B

TESTBENCHS VOOR DE CONTROLLER

B.1. TESTBENCH VHDL CONTROLLER

```
1  --In case of doubt, blame Kevin
2
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(13 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 (13 downto 0);
20
21 begin
22     clk     <=  '1' after 0 ns,
23             '0' after 40 ns when clk /= '0' else '1' after 40 ns;
24
25     reset   <=  '1' after 0 ns,     --knoppen(0) = menu
26             '0' after 124 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             "0010" after 368 ns,    --wekker_toggle -> rust
32             "0000" after 448 ns,    --knoppen(3) = down
33             "0001" after 608 ns,    --rust -> wekkertijd
34             "0000" after 688 ns,    --knoppen(3) = down
35             "0001" after 848 ns,    --wekkertijd -> rust
36             "0000" after 928 ns,    --knoppen(3) = down
37             "0001" after 1088 ns,   --rust -> wekkertijd
38             "0000" after 1168 ns,   --knoppen(3) = down
39             "0010" after 1328 ns,   --wekkertijd -> uren_set
40             "0000" after 1408 ns,   --knoppen(3) = down
41             "0100" after 1568 ns,   --uren_set -> uren_plus
42             "0000" after 1648 ns,   --knoppen(3) = down
43             "1000" after 2008 ns,   --uren_set -> uren_min
44             "0000" after 2088 ns,   --uren_min -> uren_set
45             "0001" after 2248 ns,   --uren_set -> rust
46             "0000" after 2328 ns,   --knoppen(3) = down
47             "0001" after 2488 ns,   --rust -> wekkertijd
48             "0000" after 2568 ns,   --knoppen(3) = down
```

```
            "0010" after 2768 ns,   --wekkertijd -> uren_set
            "0000" after 2808 ns,   --knoppen(3) = down
            "0010" after 2968 ns,   --uren_set -> minuten_set
            "0000" after 3048 ns,   --knoppen(3) = down
            "0100" after 3208 ns,   --minuten_set -> minuten_plus
            "0000" after 3288 ns,   --minuten_plus -> minuten_set
            "1000" after 3448 ns,   --minuten_set -> minuten_min
            "0000" after 3528 ns,   --minuten_min -> minuten_set
            "0001" after 3688 ns,   --minuten_set -> rust
            "0000" after 3768 ns,   --knoppen(3) = down
            "0001" after 3928 ns,   --rust -> wekkertijd
            "0000" after 4008 ns,   --knoppen(3) = down
            "0010" after 4168 ns,   --wekkertijd -> uren_set
            "0000" after 4248 ns,   --knoppen(3) = down
            "0010" after 4408 ns,   --uren_set -> minuten_set
            "0000" after 4488 ns,   --knoppen(3) = down
            "0010" after 4648 ns,   --minuten_set -> rust
            "0000" after 4728 ns,   --knoppen(3) = down
            "0001" after 4888 ns,   --rust -> wekkertijd
            "0000" after 4968 ns,   --knoppen(3) = down
            "1000" after 5128 ns,   --wekkertijd -> led
            "0000" after 5208 ns,   --knoppen(3) = down
            "0001" after 5368 ns,   --led -> rust
            "0000" after 5448 ns,   --knoppen(3) = down
            "0001" after 5608 ns,   --rust -> wekkertijd
            "0000" after 5688 ns,   --knoppen(3) = down
            "0100" after 5848 ns,   --wekkertijd -> geluid
            "0000" after 6128 ns,   --knoppen(3) = down
            "0100" after 6288 ns,   --geluid -> led
            "0000" after 6368 ns,   --knoppen(3) = down
            "0100" after 6528 ns,   --led -> wekkertijd
            "0000" after 6608 ns,   --knoppen(3) = down
            "1000" after 6768 ns,   --wekkertijd -> led
            "0000" after 6848 ns,   --knoppen(3) = down
            "1000" after 7008 ns,   --led -> geluid
            "0000" after 7088 ns,   --knoppen(3) = down
            "1000" after 7248 ns,   --geluid -> wekkertijd
            "0000" after 7328 ns,   --knoppen(3) = down
            "1000" after 7488 ns,   --wekkertijd -> led
            "0000" after 7568 ns,   --knoppen(3) = down
            "0010" after 7728 ns,   --led -> led_toggle
            "0000" after 7808 ns,   --led_toggle -> led
            "1000" after 7968 ns,   --led -> geluid
            "0000" after 8048 ns,   --knoppen(3) = down
            "0010" after 8208 ns,   --geluid -> geluid_toggle
            "0000" after 8288 ns,   --geluid_toggle -> geluid
            "0001" after 8448 ns,   --geluid -> rust
            "0000" after 8528 ns;   --done, done, done;

    controller_pm: controller port map(clk, reset, knoppen, wekker,menu_signal);
end architecture;
```

B.2. TESTBENCH VHDL MENU

```
--In case of doubt, blame Kevin

library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.Numeric_Std.all;

architecture behaviour of menu_tb is
component menu is
    port(clk        :in     std_logic;
        reset       :in     std_logic;
        knoppen     :in     std_logic_vector(3 downto 0);
        wekdata     :in     std_logic_vector(13 downto 0);
        enable      :out    std_logic;
        wekker      :out    std_logic_vector(13 downto 0);
        menu_signal     :out    std_logic_vector(2 downto 0));
end component menu;
```

```
17
18 signal clk, reset, enable                          :   std_logic;
19 signal menu_signal                                  : std_logic_vector (2 downto 0);
20 signal knoppen                                   : std_logic_vector (3 downto 0);
21 signal wekdata, wekker      :   std_logic_vector (13 downto 0);
22
23 begin
24     clk      <=  '1' after 0 ns,
25             '0' after 20 ns when clk /= '0' else '1' after 20 ns;
26
27     reset    <=  '1' after 0 ns,      --knoppen(0) = menu
28             '0' after 62 ns;         --knoppen(1) = set
29                                      --knoppen(2) = up
30     knoppen <=  "0000" after 0 ns,   --knoppen(3) = down
31             "0010" after 68 ns,      --rust -> wekker_toggle
32             "0010" after 108 ns,     --wekker_toggle -> rust
33             "0001" after 148 ns,     --rust -> wekkertijd
34             "0001" after 188 ns,     --wekkertijd -> rust
35             "0001" after 228 ns,     --rust -> wekkertijd
36             "0010" after 268 ns,     --wekkertijd -> uren_set
37             "0100" after 308 ns,     --uren_set -> uren_plus
38             "0000" after 348 ns,     --uren_plus -> uren_set
39             "1000" after 388 ns,     --uren_set -> uren_min
40             "0000" after 428 ns,     --uren_min -> uren_set
41             "0001" after 468 ns,     --uren_set -> rust
42             "0001" after 508 ns,     --rust -> wekkertijd
43             "0010" after 548 ns,     --wekkertijd -> uren_set
44             "0010" after 588 ns,     --uren_set -> minuten_set
45             "0100" after 628 ns,     --minuten_set -> minuten_plus
46             "0000" after 668 ns,     --minuten_plus -> minuten_set
47             "1000" after 708 ns,     --minuten_set -> minuten_min
48             "0000" after 748 ns,     --minuten_min -> minuten_set
49             "0001" after 788 ns,     --minuten_set -> rust
50             "0001" after 828 ns,     --rust -> wekkertijd
51             "0010" after 868 ns,     --wekkertijd -> uren_set
52             "0010" after 908 ns,     --uren_set -> minuten_set
53             "0010" after 948 ns,     --minuten_set -> wekkertijd     EIGENLIJK GAAT DIT NAAR
                   RUST TOE
54             "0001" after 988 ns,     --rust -> wekkertijd
55             "1000" after 1028 ns,    --wekkertijd -> led
56             "0001" after 1068 ns,    --led -> rust
57             "0001" after 1108 ns,    --rust -> wekkertijd
58             "0100" after 1148 ns,    --wekkertijd -> geluid
59             "0100" after 1188 ns,    --geluid -> led
60             "0100" after 1228 ns,    --led -> wekkertijd
61             "1000" after 1268 ns,    --wekkertijd -> led
62             "1000" after 1308 ns,    --led -> geluid
63             "1000" after 1348 ns,    --geluid -> wekkertijd
64             "1000" after 1388 ns,    --wekkertijd -> led
65             "0010" after 1428 ns,    --led -> led_toggle
66             "0000" after 1468 ns,    --led_toggle -> led
67             "1000" after 1508 ns,    --led -> geluid
68             "0010" after 1548 ns,    --geluid -> geluid_toggle
69             "0000" after 1588 ns,    --geluid_toggle -> geluid
70             "0001" after 1628 ns,    --geluid -> rust
71             "0000" after 1668 ns;    --done, done, done;
72
73     wekdata <= "00000001000001" after 20 ns;
74
75     menu_pm: menu port map(clk, reset, knoppen, wekdata, enable, wekker, menu_signal);
76 end architecture;
```

B.3. TESTBENCH VHDL GEHEUGEN

```
1 library IEEE;
2 use IEEE.std_logic_1164.ALL;
3
4 architecture behaviour of geheugen_tb is
5 component geheugen is
6     port(clk    :in    std_logic;
```

```
        reset  :in    std_logic;
        enable :in    std_logic;
        wek_in :in    std_logic_vector(13 downto 0);
        wek_out:out   std_logic_vector(13 downto 0));
end component geheugen;

signal  clk,enable,reset    :   std_logic;
signal  wek_in,wek_out      :       std_logic_vector(13 downto 0);


begin
    clk     <=  '0' after 0 ns,
            '1' after 20 ns  when clk /= '1' else '0' after 20 ns;

    reset   <=  '1' after 0 ns,
            '0' after 85 ns;


    enable <=   '0' after 0 ns,
            '1' after 150 ns,
            '0' after 290 ns,
            '1' after 590 ns;

    wek_in <=   "00000000000001" after 0 ns,
            "00000000000010" after 70 ns,
            "00000000000011" after 110 ns,
            "00000000000100" after 150 ns,
            "00000000000101" after 190 ns,
            "00000000000110" after 230 ns,
            "00000000000111" after 270 ns,
            "00000000001000" after 310 ns,
            "00000000001001" after 350 ns,
            "00000000001010" after 390 ns,
            "00000000001011" after 430 ns,
            "00000000001100" after 470 ns,
            "00000000001101" after 510 ns,
            "00000000001110" after 550 ns,
            "00000000001111" after 590 ns,
            "00000000010000" after 630 ns,
            "00000000010001" after 680 ns,
            "00000000010010" after 735 ns,
            "00000000010111" after 779 ns;

    geheugen_pm: geheugen port map(clk,reset,enable,wek_in,wek_out);
end behaviour;
```

B.4. TESTBENCH VHDL BUFFER

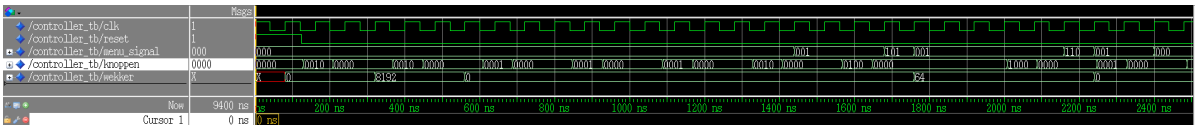
```
library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of buff_tb is
component buff is
    port(clk            :in     std_logic;
        reset       :in     std_logic;
        knoppen     :in     std_logic_vector(3 downto 0);
        knopjes     :out    std_logic_vector(3 downto 0));
end component buff;

signal  clk,enable,reset        :   std_logic;
signal  knoppen,knoppjes        :       std_logic_vector(3 downto 0);


begin
    clk     <=  '0' after 0 ns,
            '1' after 20 ns  when clk /= '1' else '0' after 20 ns;

    reset   <=  '1' after 0 ns,
            '0' after 85 ns;

```

```
23      knoppen <=  "0000" after 0 ns,
24                  "1111" after 100 ns,
25                  "0000" after 150 ns,
26                  "1000" after 190 ns,
27                  "0000" after 240 ns,
28                  "0001" after 290 ns;
29
30      buff_pm: buff port map(clk,reset,knoppen,knoppjes);
31  end behaviour;
```

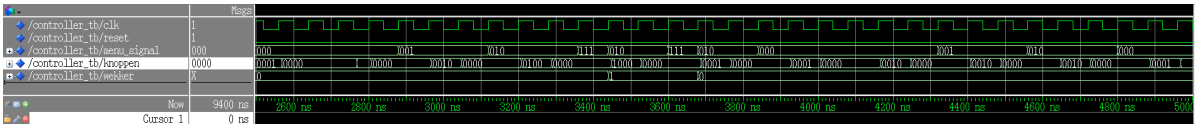
C

SIMULATIES RESULTATEN VAN DE CONTROLLER

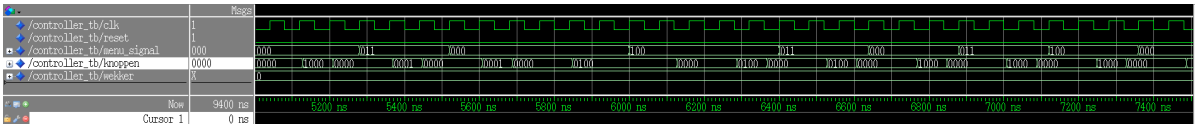
C.1. BEHAVORAL SIMULATIE



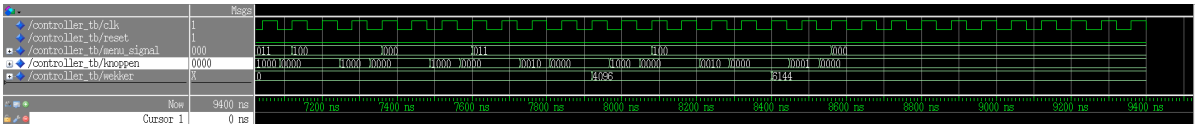
Figuur C.1: Simulatie van 0 tot 2500ns



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

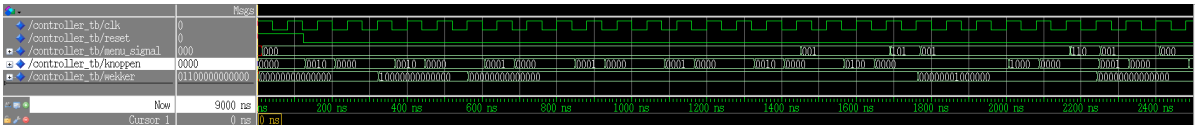


Figuur C.3: Simulatie van 5000ns tot 7500ns

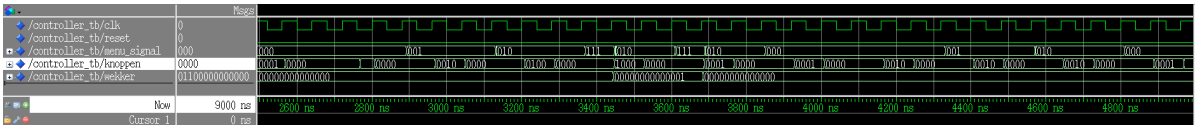


Figuur C.4: Simulatie van 7500ns tot het einde

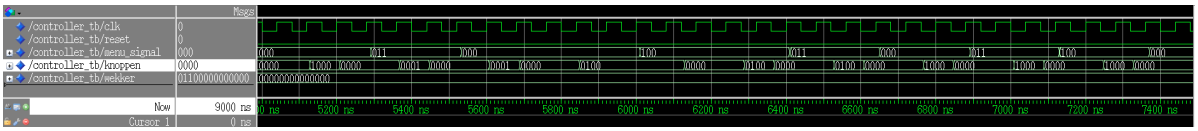
C.2. SYNTHESIZE SIMULATIE



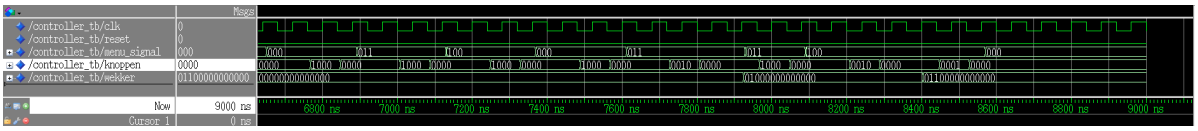
Figuur C.5: Simulatie van 0 tot 2500ns



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

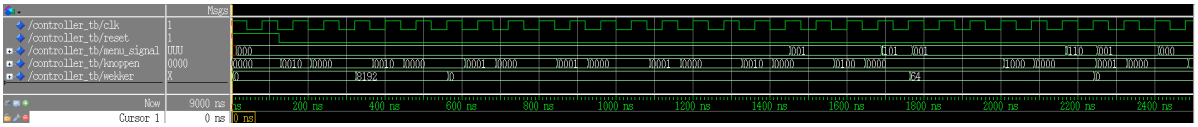


Figuur C.7: Simulatie van 5000ns tot 7500ns

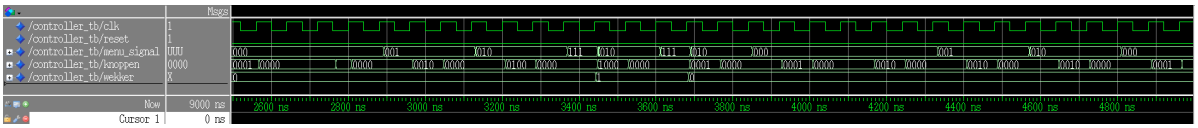


Figuur C.8: Simulatie van 7500ns tot het einde

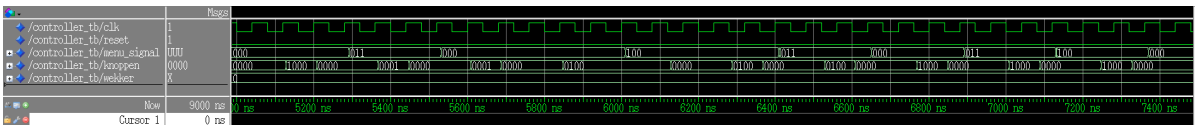
C.3. EXTRACTED SIMULATIE



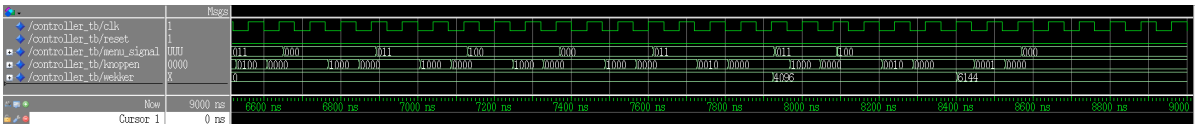
Figuur C.9: Simulatie van 0 tot 2500ns



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

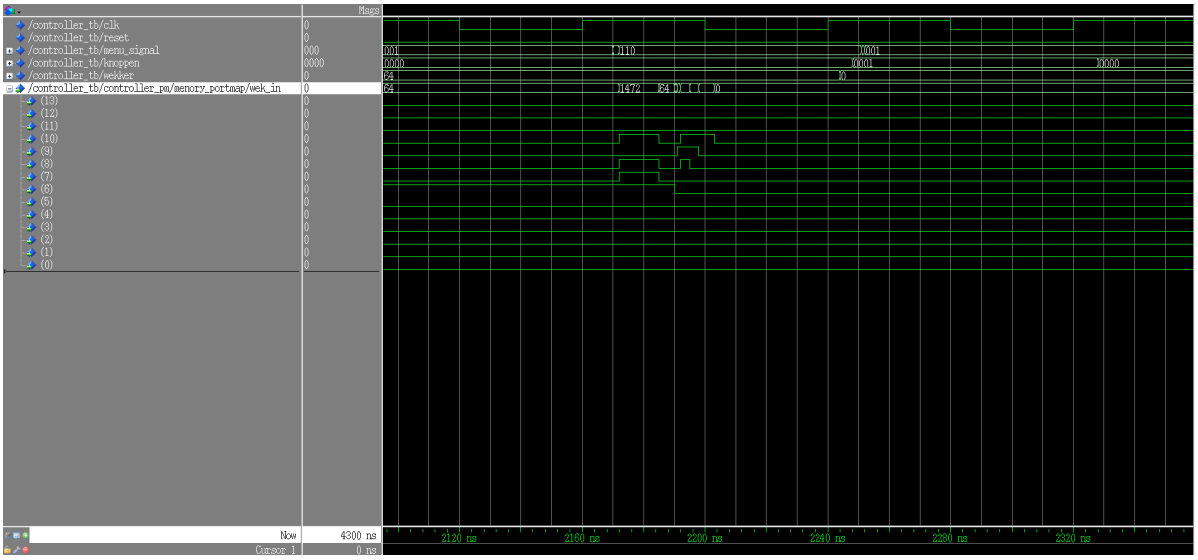


Figuur C.11: Simulatie van 5000ns tot 7500ns



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

C.4. TIMING



Figuur C.13: Timing problemen

D

VHDL CODE VAN HET ALARM

D.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;
```

D.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)
                           ) - 15, 6));
21                     alarm_uur <= wekker_uur;
22                 else
23                     alarm_min <= std_logic_vector(to_unsigned(60 - (15-to_integer(unsigned(
                           wekker_min))),6));
24                     if (to_integer(unsigned(wekker_uur)) = 0) then
25                         alarm_uur <= std_logic_vector(to_unsigned(23, 5));
26                     else
27                         alarm_uur <= std_logic_vector(to_unsigned(to_integer(unsigned(
                               wekker_uur)) - 1, 5));
```

```
                    end if;
                end if;
                state <= new_state;
            end if;
        end if;
    end process;
    lbl2: process (state, alarm_min, alarm_uur, wekker_uur, wekker_min, tijd_min, tijd_uur)
    begin
        case state is
            when steady =>
                geluid <= '0';
                licht <= '0';
                if (alarm_min = tijd_min) and (alarm_uur = tijd_uur) then
                    new_state <= start;
                else
                    new_state <= steady;
                end if;
            when start =>
                geluid <= '0';
                licht <= '1';
                if (wekker_uur = tijd_uur) and (wekker_min = tijd_min) then
                    new_state <= final;
                else
                    new_state <= start;
                end if;
            when final =>
                geluid <= '1';
                licht <= '1';
                new_state <= final;
            when others =>
                geluid <= '0';
                licht <= '1';
                new_state <= state;
        end case;
    end process;
end behaviour;
```

D.3. TOP ENTITY ALARM

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity alarm is
   port(clk       :in    std_logic;
        reset     :in    std_logic;
        sec       :in    std_logic;
        licht     :in    std_logic;
        pwm_signal:out   std_logic);
end alarm;
```

D.4. BEHAVIOURAL ALARM

```
library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of alarm is
component counter
     port(  clk   :in    std_logic;
                reset :in    std_logic;
                sec   :in    std_logic;
        licht :in    std_logic;
                length:out   std_logic_vector(5 downto 0));
end component;
component pwm
    port(   clk   :in    std_logic;
                reset :in    std_logic;
                length:in    std_logic_vector(5 downto 0);
                pwm_signal    :out   std_logic);
end component;
signal length : std_logic_vector (5 downto 0);
```

```
19 begin
20     counter_1 : counter port map (clk => clk, reset => reset, sec => sec, licht => licht,
           length => length);
21     pwm_1 : pwm port map (clk => clk, reset => reset, length => length, pwm_signal =>
           pwm_signal);
22 end behaviour;
```

D.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;
```

D.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;
```

```
                    new_state <= laag;
                end if;
            when hoog =>
                if (sec = '0') then
                    new_state <= laag;
                else
                    new_state <= hoog;
                end if;
                new_count <= count;
                new_length2 <= length2;
            when others =>
                new_count <= count;
                new_length2 <= length2;
                new_state <= hoog;
        end case;
    end process;
end behaviour;
```

D.7. ENTITY ALARM-PWM

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity pwm is
   port(clk    :in    std_logic;
        reset  :in    std_logic;
        length:in     std_logic_vector(5 downto 0);
        pwm_signal    :out    std_logic);
end pwm;
```

D.8. BEHAVIOURAL ALARM-PWM

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.numeric_std.ALL;

architecture behaviour of pwm is
    type pwm_state is (hoog, laag, res_state);
    signal counter, new_counter: unsigned(5 downto 0);
    signal state, new_state: pwm_state;
begin
    lbl1: process(clk)
    begin
        if(clk'event and clk = '1') then
            if (reset = '1') then
                state <= res_state;
                counter <= (others => '0');
            else
                state <= new_state;
                counter <= new_counter;
            end if;
        end if;
    end process;
    lbl2: process(counter, length, state)
    begin
        case state is
            when res_state =>
                pwm_signal <= '0';
                new_counter <= (others => '0');
                new_state <= laag;
            when laag =>
                pwm_signal <= '0';
                new_counter <= counter + 1;
                if (unsigned(length) <= counter) then
                    new_state <= hoog;
                else
                    new_state <= laag;
                end if;
            when hoog =>
                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;
```

BIBLIOGRAFIE