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

TU Delft Delft University of Technology

Challenge the future

SAMENVATTING

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

INHOUDSOPGAVE

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

2

ONTWERP SPECIFICATIE

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.

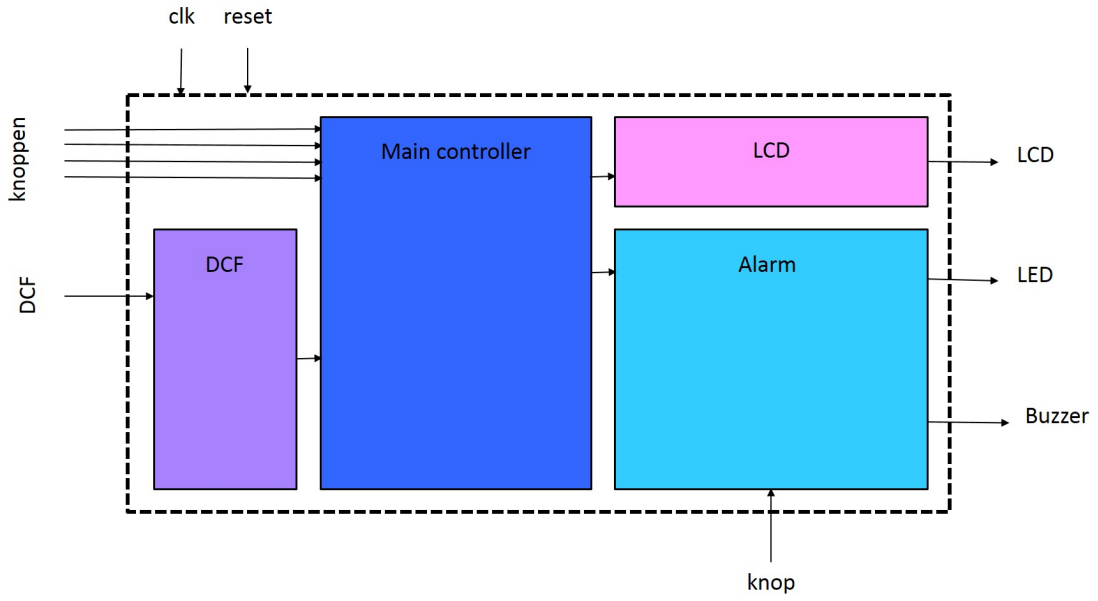
3

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

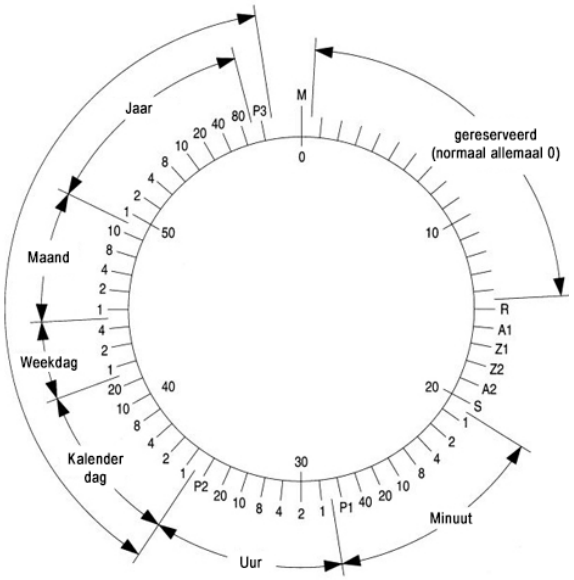
bestuurd worden.
In het alarmblok wordt eerst de vijftien minuten van de wekkertijd afgetrokken. De ingestelde tijd is namelijk de tijd waarop het geluid aan moet gaan, de lamp moet al een kwartier eerder beginnen met branden. Daarnaast zorgt het alarm ervoor dat een PWM-signaal gegenereerd wordt wat naar een LED gaat.
De LCD controller zorgt dat de datum, tijd, ingestelde alarmtijd en de veranderingen in het menu op de LCD zichtbaar zijn. Er wordt een LCD scherm gebruikt waar de pixels afzonderlijk van elkaar aangestuurd worden. Tussen de chip en het scherm zit nog een microcontroller, waarin de karakters zijn opgeslagen, dit zou namelijk te groot zijn om op de chip te regelen.

4

DCF77

4.1. INLEIDING

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



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

4.2. SPECIFICATIES

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

4.2.1. INGANGEN

Dit onderdeel maakt gebruik van de volgende ingangen:

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

4.2.2. UITGANGEN

Dit onderdeel genereert de volgende uitgangen:

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

4.2.3. GEDRAG

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

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

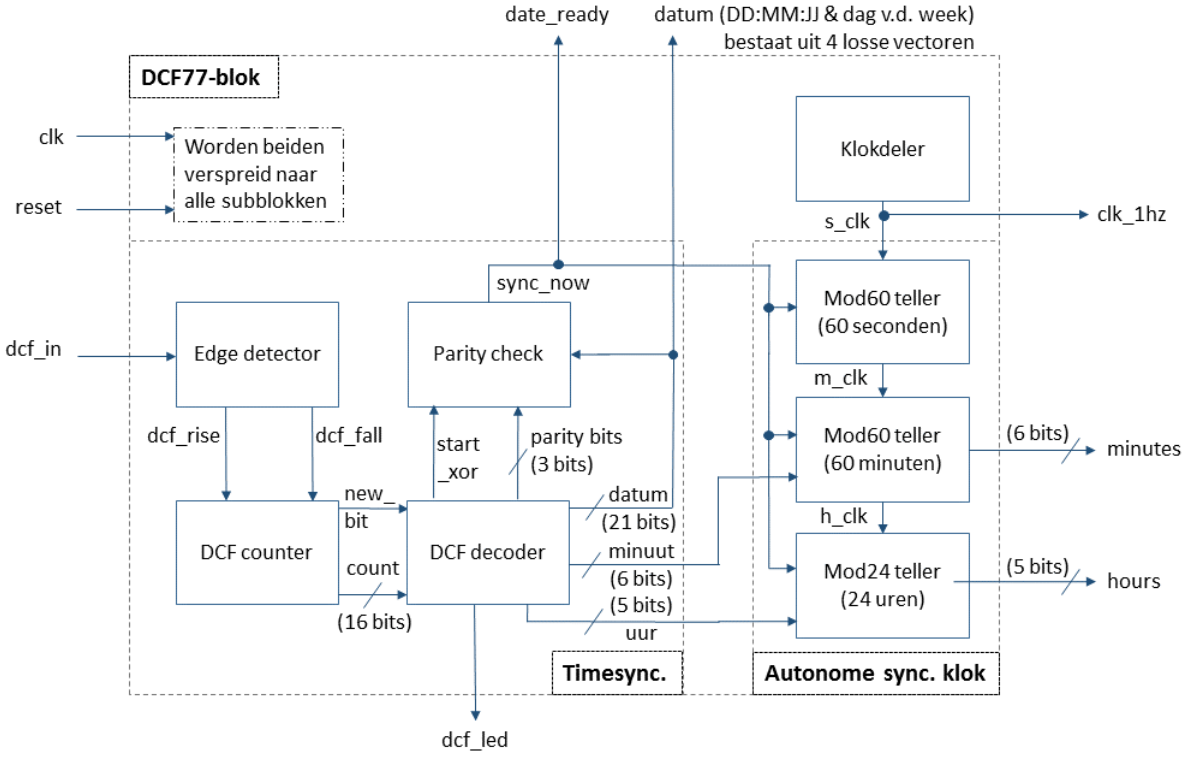
4.3. FUNCTIONALITEIT

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

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

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

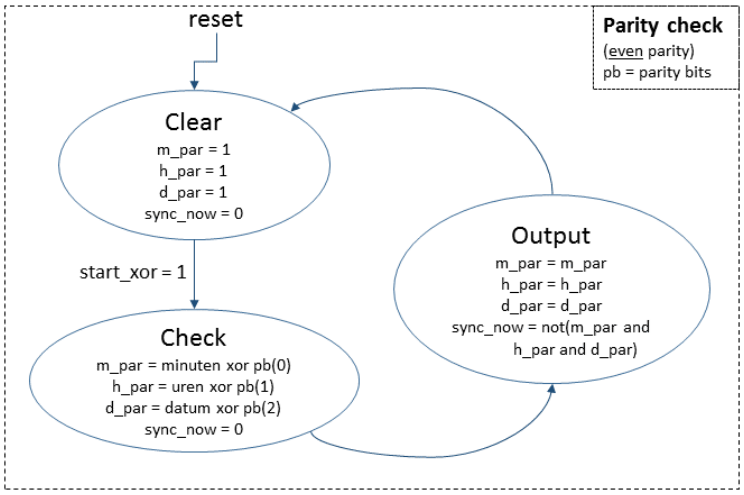
Het sync_now signaal komt vervolgens aan bij de autonome synchroniseerbare klok. Dit subblok besstaat vervolgens zelf weer uit twee mod60 tellers en een mod24 teller, waarmee autonoom de tijd kan worden bijgehouden, mocht het DCF77 signaal onverhoopt wegvallen. Wanneer sync_now hoog wordt, worden deze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



Figuur 4.2: Verdeling van het DCF blok in subblokken

4.3.1. FSM DIAGRAMMEN

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



Figuur 4.3: FSM diagram van het subblok parity check

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

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

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

4.3.2. VHDL CODE

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

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 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 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 geld 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 downto 12] = wekdata[13 downto 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 B.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 B.2.1 en B.2.2.
- Het menu, hierin zit de echte logica verwerkt. Deze is te vinden in appendices B.2.3 en B.2.4.
- Het gebruikte geheugen element voor de opslag van 14 bits, te vinden in appendices B.2.5 en B.2.6.
- De gebruikte buffer is te vinden in appendices B.2.7 en B.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 B.3.1 tot B.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

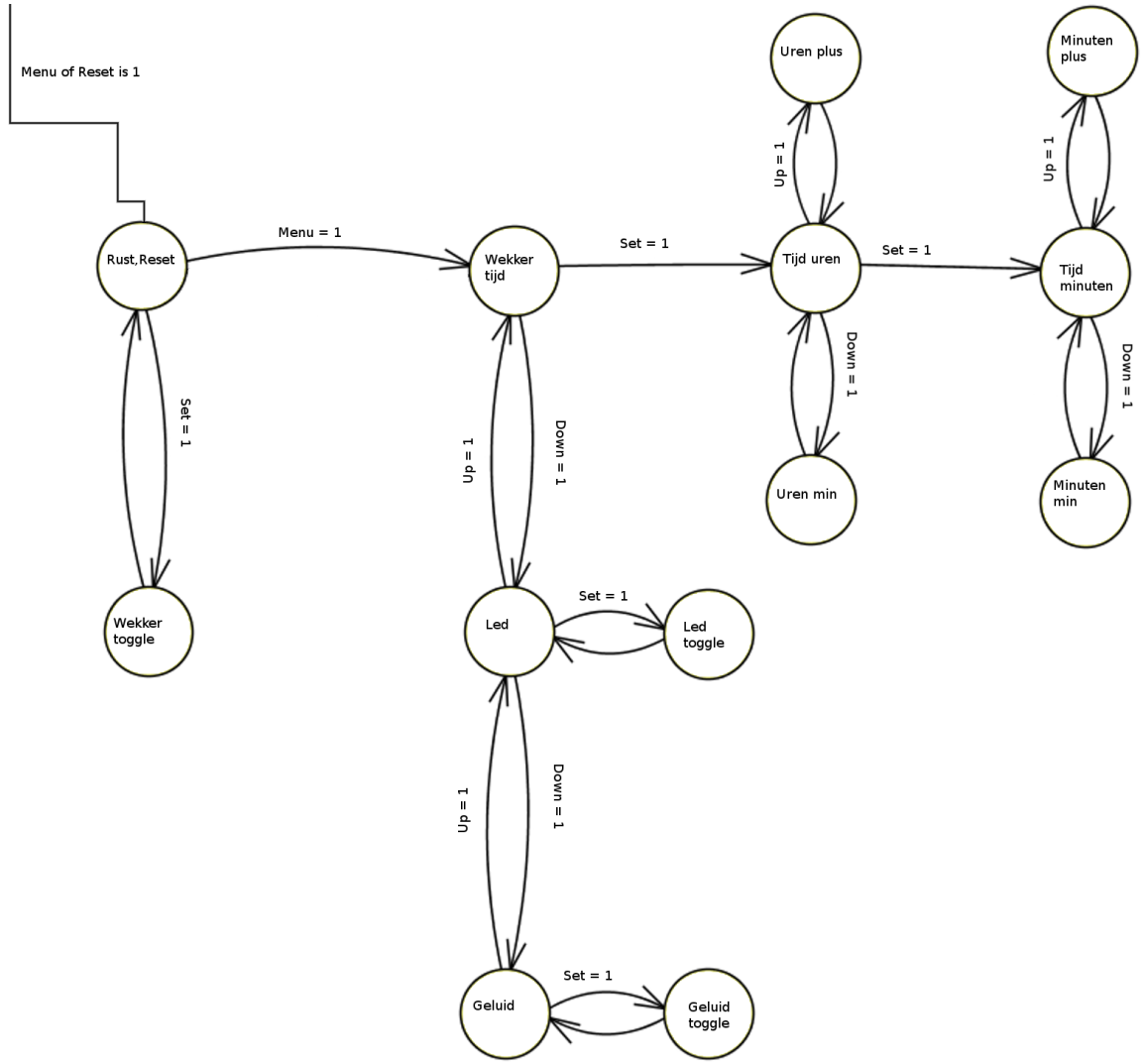
De resultaten van de simulatie staan in appendix C. De testbench is te lang om in een keer weer te geven daarom is deze op geknipt in vier stukken. De testbench die gemaakt is voor de simulatie staat in appendix B.3.1

5.6. 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.

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

6.2.2. UITGANGEN

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

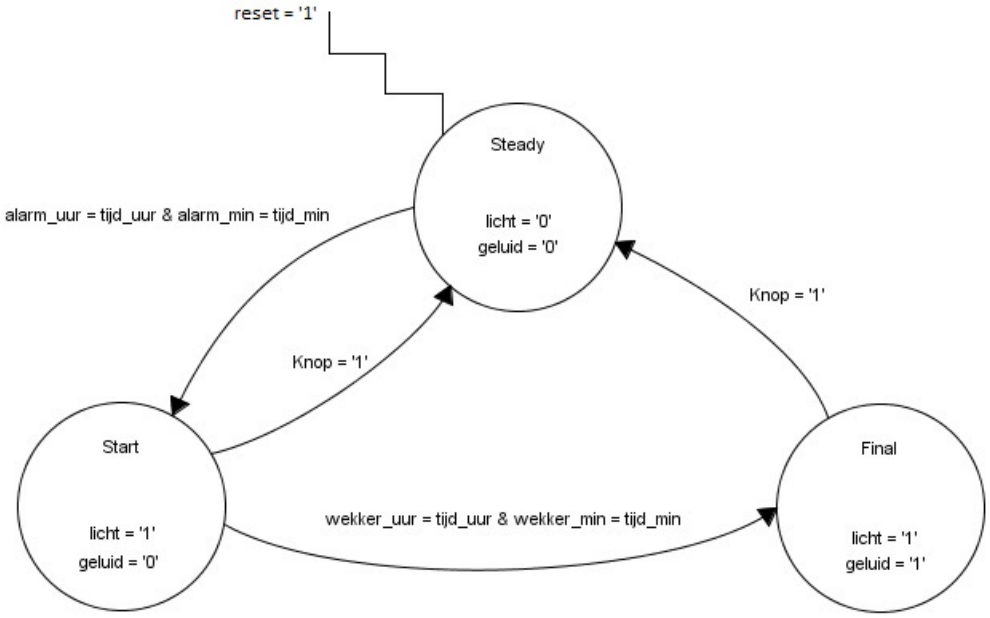
6.2.3. GEDRAG

Het alarm moet een bepaalde tijd voordat de wekker is ingesteld aangaan, nu gekozen voor 15 minuten. Er wordt 15 minuten van de ingestelde tijd afgetrokken. Zodra die tijd gelijk is aan de huidige tijd komt er een signaal (licht) aan bij het gedeelte wat voor een pwm signaal zorgt. In dat gedeelte wordt een pwm signaal gegenereerd dat elke 15 seconde breder wordt. Dit wordt gedaan door in een counter 15 seconde te tellen. Elke 15 seconde wordt de variable "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.

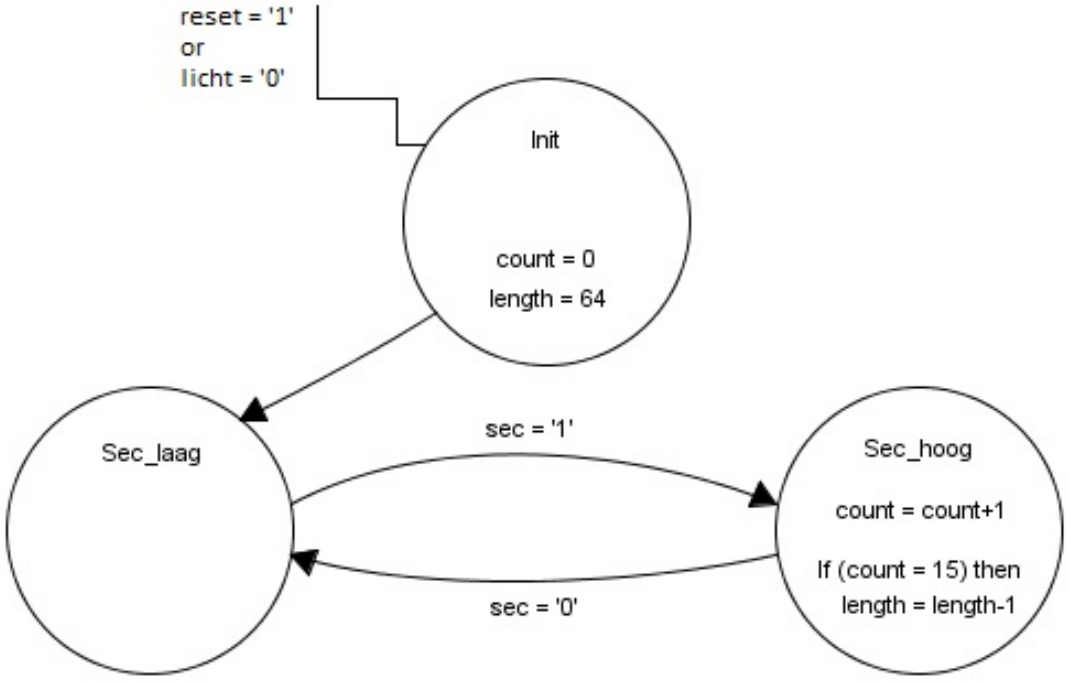
6.3. FUNCTIONALITEIT

6.3.1. FSM

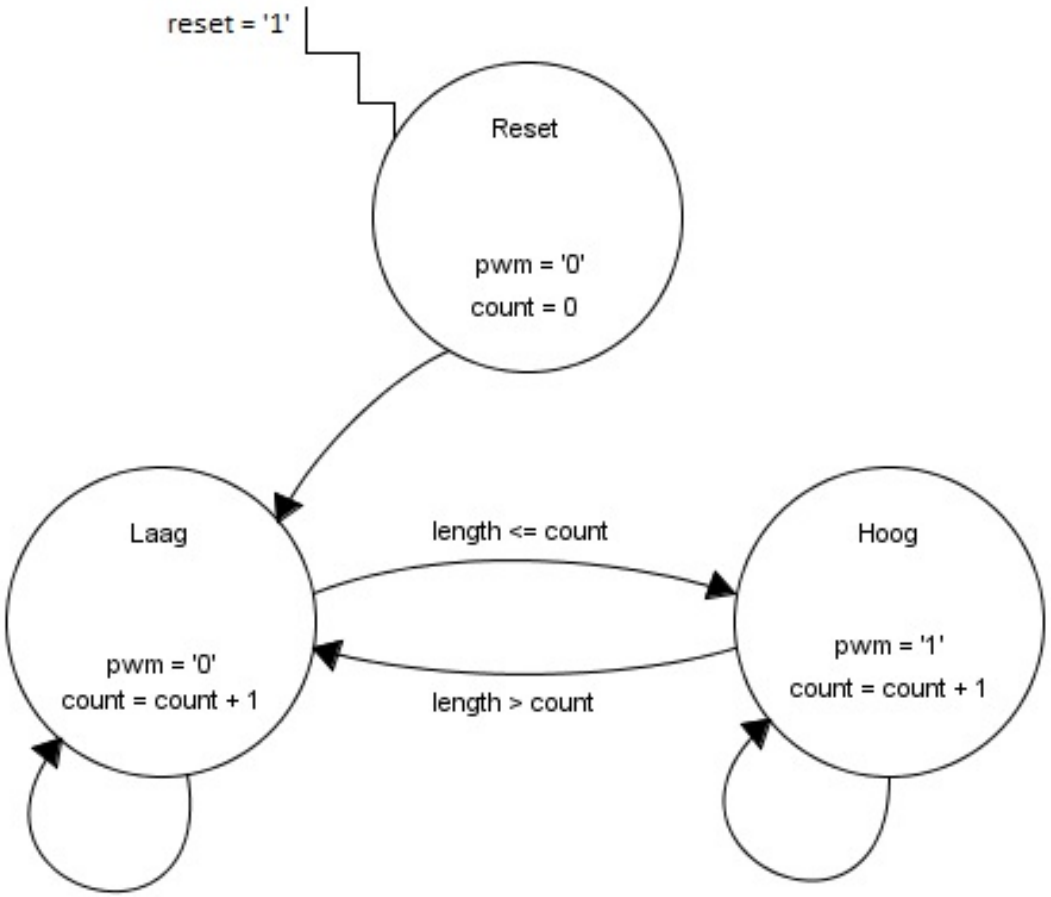
FSM van “alarm-compare", het gedeelte waar de tijd vergeleken wordt met de ingestelde wekker tijd.



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



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



6.3.2. CODE

De code voor het alarm is opgedeeld in 3 stukken:

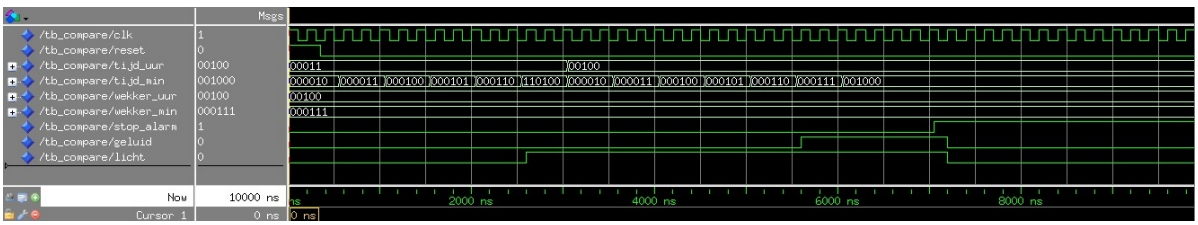
- alarm-compare
- alarm-counter
- alarm-pwm

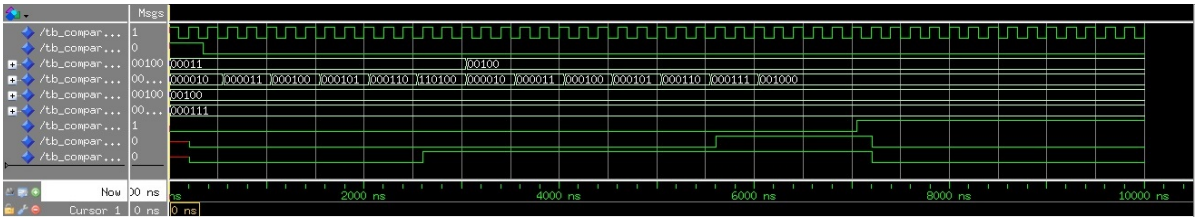
Alarm-counter en alarm-pwm zijn onderdeel van een top entity, alarm. Omdat het nog niet zeker is waar alarm-compare geplaatst gaat worden op de chip is die daar niet bij inbegrepen. De code voor het alarm is te vinden in appendix B.4.

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

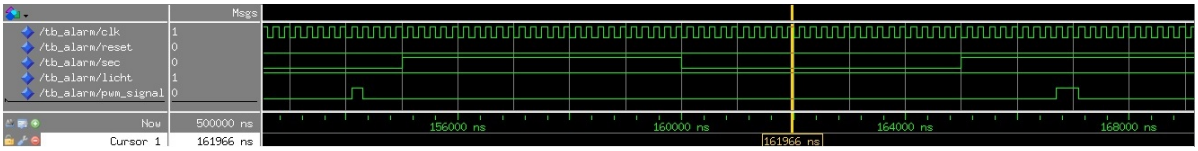
6.4. RESULTATEN

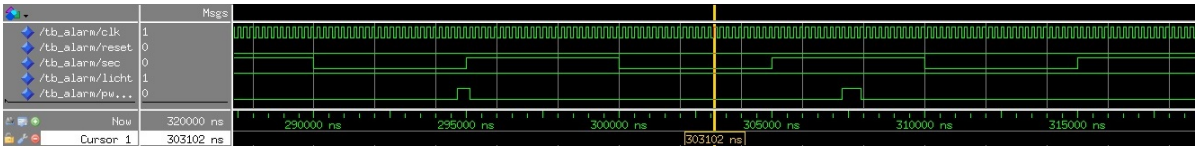
Alle onderdelen werken in de simulaties. Zowel de simulatie van de behaviour als van de extracted vhdl. Onderstaande afbeeldingen zijn de resultaten van de simulaties, eerst die van de behaviour daarna van de extracted. De eerste 2 afbeedingen zijn van “alarm-compare", de andere 2 van “alarm-pwm".





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.

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 coordinaten 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 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.

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

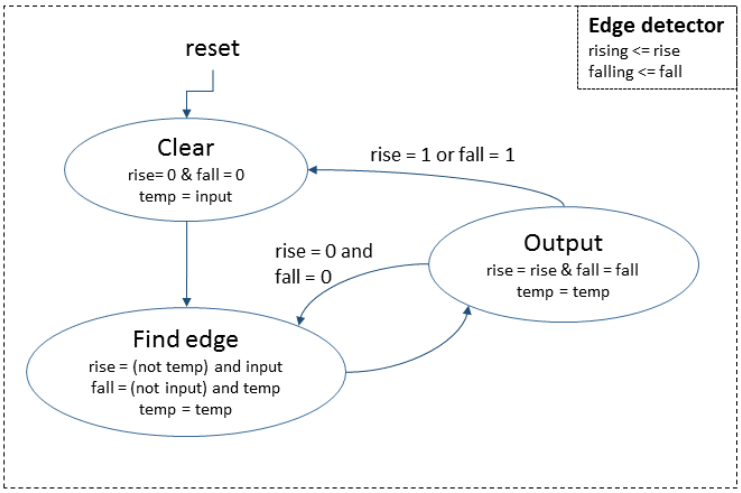
CONCLUSIE

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.

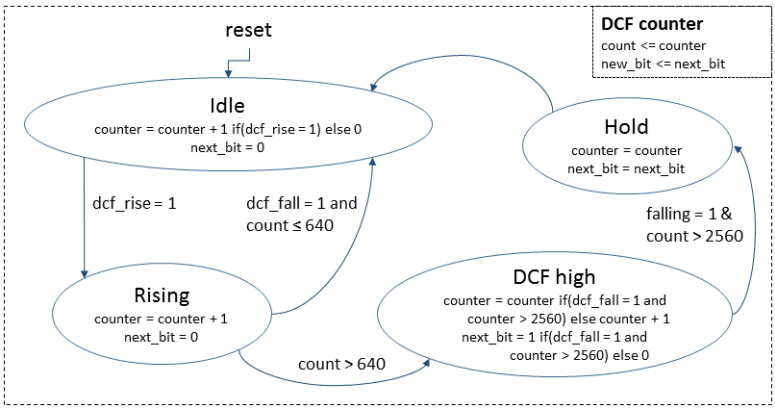
A

FSM DIAGRAMMEN (DCF77 BLOK)

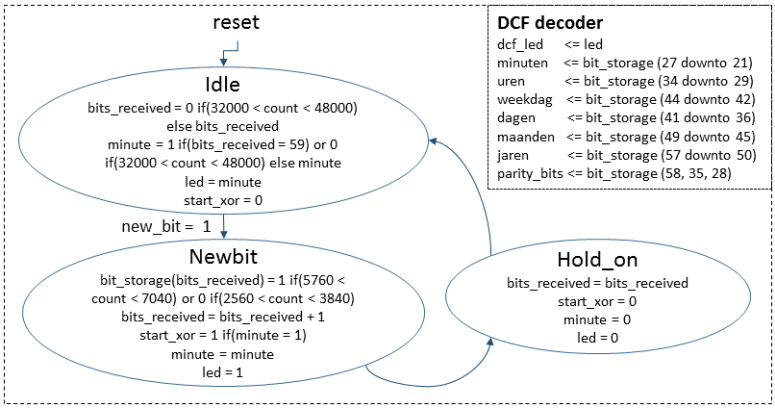
SUBBLOKKEN VAN SYNCTIME



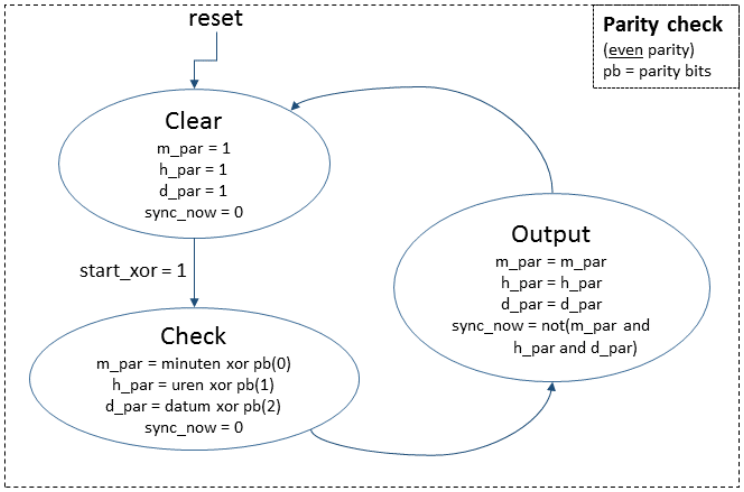
Figuur A.1: FSM diagram van het subblok edge detector



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

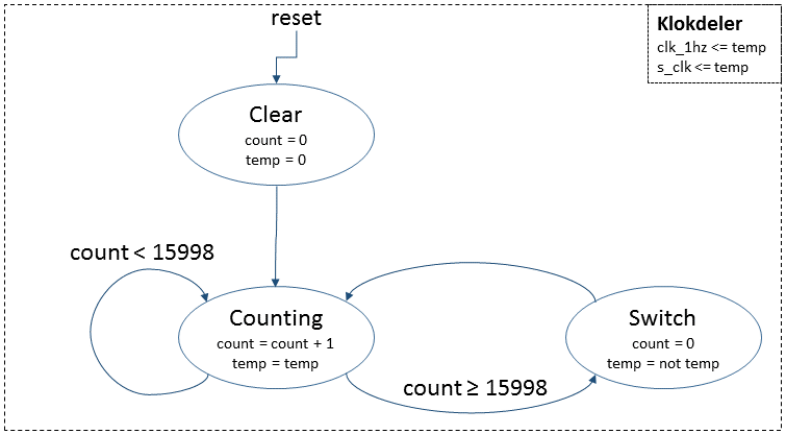


Figuur A.3: FSM diagram van het subblok DCF decoder



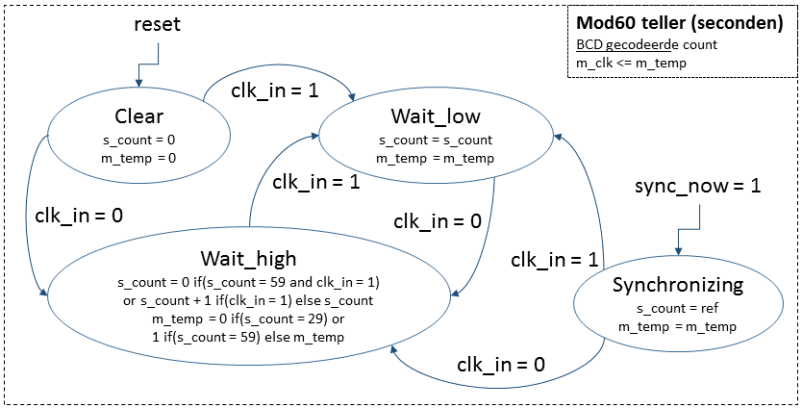
Figuur A.4: FSM diagram van het subblok parity check

KLOKDELER

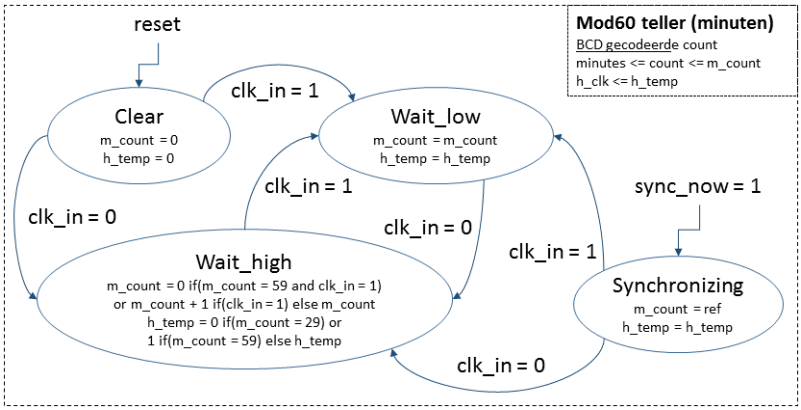


Figuur A.5: FSM diagram van het subblok klokdeler

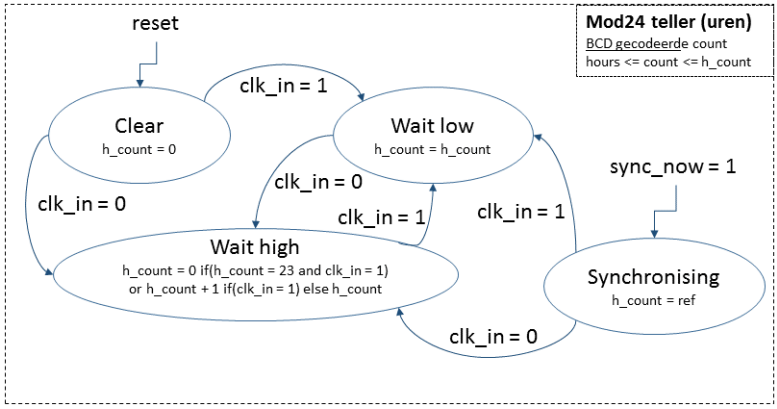
SUBBLOKKEN VAN DE KLOK



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



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



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

B

VHDL CODE

B.1. VHDL CODE VAN HET DCF77 BLOK

B.1.1. EDGE DETECTOR

```
1  -- Alex Oudsen, 4325494
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  entity edge_detector is
7     port(clk    :in    std_logic;    -- 32 kHz systeemklok
8          reset  :in    std_logic;    -- Systeemreset
9          input  :in    std_logic;    -- digitaal DCF77 signaal
10         rising :out   std_logic;    -- DCF77 rising edge
11         falling:out   std_logic);   -- DCF77 falling edge
12 end edge_detector;
```

```
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                 state <= clear;
```

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

B.1.2. DCF COUNTER

```vhdl
-- Alex Oudsen, 4325494

library ieee;
use ieee.std_logic_1164.all;

entity dcf_counter is
   port(clk      :in     std_logic;    -- 32 kHz systeemklok
        reset    :in     std_logic;    -- Systeemreset
        dcf_rise:in      std_logic;    -- DCF77 signaal - rising edge
        dcf_fall:in      std_logic;    -- DCF77 signaal - falling edge
        count    :out    std_logic_vector(15 downto 0);  -- Tellerwaarde
    new_bit :out    std_logic); -- Een nieuwe bit is geteld
end dcf_counter;
```

```vhdl
-- Alex Oudsen, 4325494
-- De dcf_counter is verantwoordelijk voor het detecteren van de pulsen
-- van het dcf77 signaal waarmee de informatiebits zijn gecodeerd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

architecture behaviour of dcf_counter is

    type count_state is (idle, rising, dcf_high, hold);
    signal state, new_state: count_state;

    signal next_bit, new_next_bit: std_logic;
    signal counter, new_counter: std_logic_vector(15 downto 0);

begin
    count <= counter;
    new_bit <= next_bit;

    process(clk) is
    begin
        if(clk'event and clk = '1') then
            if(reset = '1') then        -- Systeemreset
                counter <= (others => '0');
                next_bit <= '0';
                state <= idle;
            else
                counter <= new_counter;
                next_bit <= new_next_bit;
                state <= new_state;
            end if;
        end if;
    end process;

    process(state, dcf_rise, dcf_fall, counter, next_bit) is
    begin
        case state is
            when idle =>
                -- Controleer op een rising edge van het dcf77 signaal
                if(dcf_rise = '1') then
                    new_counter <= (others => '0');
                    new_next_bit <= '0';
                    new_state <= rising;
                else
                    new_counter <= counter + 1;
                    new_next_bit <= '0';
                    new_state <= idle;
                end if;
            when rising =>
                -- Hier wordt bepaald of een rising edge hoort bij
                -- een daadwerkelijke puls of een spike/glitch
                new_counter <= counter + 1;
                new_next_bit <= '0';
                if(counter > 640) then
```

```
                    new_state <= dcf_high;
                elsif(dcf_fall = '1') then
                    new_state <= idle;
                else
                    new_state <= rising;
                end if;
            when dcf_high =>
                -- Wacht op een falling edge van het dcf77 signaal
                -- Negeer bovendien falling edges t.g.v. spikes/glitches
                if(dcf_fall = '1' and counter > 2560) then
                    new_counter <= counter;
                    new_next_bit <= '1';
                    new_state <= hold;
                else
                    new_counter <= counter + 1;
                    new_next_bit <= '0';
                    new_state <= dcf_high;
                end if;
            when hold =>
                new_counter <= counter;
                new_next_bit <= next_bit;
                new_state <= idle;
            when others =>          -- Zou nooit mogen voorkomen
                new_counter <= (others => '0');
                new_next_bit <= '0';
                new_state <= idle;
        end case;
    end process;
end behaviour;
```

B.1.3. DCF DECODER

```
-- Alex Oudsen, 4325494

library ieee;
use ieee.std_logic_1164.all;

entity dcf_decoder is
   port(clk          :in  std_logic; -- 32 kHz systeemklok
        reset            :in  std_logic; -- Systeemreset
        count            :in  std_logic_vector(15 downto 0); -- Tellerwaarde
    new_bit      :in  std_logic; -- Een nieuwe bit is geteld
    dcf_led      :out std_logic; -- Debug signaal voor signaalontvangst
    start_xor        :out std_logic; -- Enable signaal voor parity_check
    minuten      :out std_logic_vector(6 downto 0);  -- Minuten in BCD
    uren         :out std_logic_vector(5 downto 0);  -- Uren in BCD
    weekdag      :out std_logic_vector(2 downto 0);  -- Dagen (001 is maandag, enz.)
    dagen        :out std_logic_vector(5 downto 0);  -- Dagen (de cijfers) in BCD
    maanden      :out std_logic_vector(4 downto 0);  -- Nummer van de maand in BCD
    jaren        :out std_logic_vector(7 downto 0);  -- Laatste 2 cijfers van het jaartal; in
        BCD
    parity_bits      :out std_logic_vector(2 downto 0)); -- De 3 parity bits uit het DCF77
        signaal
end dcf_decoder;
```

```
-- Joran Out, 4331958 & Alex Oudsen, 4325494
-- De dcf_decoder filtert met behulp van de informatie die
-- de dcf_counter over het digitale dcf ingangssignaal levert
-- de gewenste bits waarmee datum en tijd zijn gecodeerd
-- Bovendien worden de drie bijbehorende parity_bits
-- ook meegegeven aan het volgende onderdeel (parity_check)
-- Het uitgangssignaal dcf_led geeft aan of het dcf signaal
-- goed wordt ontvangen door uit te gaan op het moment dat
-- er bits uit de reeks van 59 die het dcf77 signaal bevat, ontbreken

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

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.
                systeemklok
47              bits_received <= new_bits_received;
48              bit_storage <= new_bit_storage;
49              minute <= new_minute;
50              led <= new_led;
51              state <= new_state;
52          end if;
53      end process;
54
55      process(state, count, new_bit, minute, bits_received, bit_storage, led) is
56
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;
```

B.1.4. PARITY CHECK

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

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

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

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

B.1.5. SYNCTIME

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

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

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

B.1.6. KLOKDELER

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

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

```
6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_unsigned.all;
8
9  architecture behaviour of klokdeler is
10
11     type kd_state is (clear, counting, switch);                  -- Declaratie van de
           gebruikte states
12     signal state, new_state: kd_state;
13
14     signal count, new_count: std_logic_vector(13 downto 0);      -- Voor het tellen tot 32000
15     signal temp, new_temp: std_logic;                    -- Interne versie van het 1 Hz
           signaal
16
17 begin
18     clk_1hz <= temp;                     -- Uitvoeren van het interne 1 Hz signaal
19
20     process(clk) is
21     begin
22         if(clk'event and clk = '1') then
23             if(reset = '1') then             -- Systeemreset
24                 temp <= '0';
25                 count <= (others => '0');
26                 state <= clear;
27             else
28                 temp <= new_temp;
29                 count <= new_count;
30                 state <= new_state;
31             end if;
32         end if;
33     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;
```

B.1.7. MOD60 (SECONDEN) TELLER

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

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

```
1   -- Joran Out, 4331958 & Alex Oudsen, 4325494
2   -- Deze mod60 teller telt elke rising edge van clk_in 1
3   -- bij de huidige waarde van count op in bcd codering
4   -- De waarde van count wordt echter gelijk gemaakt aan ref,
5   -- wanneer het signaal sync_now hoog wordt
6
7   library ieee;
8   use ieee.std_logic_1164.all;
9   use ieee.std_logic_unsigned.all;
10
11  architecture behaviour of mod60_clk_bcd is
12
13      type m60_state is (clear, wait_high, wait_low, synchronising);
14      signal state, new_state: m60_state;
15
16      signal s_count, new_s_count: std_logic_vector(6 downto 0);
17      signal m_temp, new_m_temp: std_logic;
18
19  begin
20      m_clk <= m_temp;
21
22      process(clk) is
23      begin
24          if(clk'event and clk = '1') then
25              if(reset = '1') then                -- 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;
```

```
                else
                    new_s_count <= s_count;
                    new_m_temp <= m_temp;
                    new_state <= wait_high;
                end if;
            when wait_low =>
                if(clk_in = '0') then
                    new_s_count <= s_count;
                    new_m_temp <= m_temp;
                    new_state <= wait_high;
                else
                    new_s_count <= s_count;
                    new_m_temp <= m_temp;
                    new_state <= wait_low;
                end if;
            when synchronising =>
                new_s_count <= ref;
                new_m_temp <= m_temp;
                if(clk_in = '1') then
                    new_state <= wait_low;
                else
                    new_state <= wait_high;
                end if;
            when others =>              -- Zou nooit mogen voorkomen
                new_s_count <= (others => '0');
                new_m_temp <= '0';
                new_state <= clear;
        end case;
    end process;
end behaviour;
```

B.1.8. MOD60 (MINUTEN) TELLER

```
-- Joran Out, 4331958 & Alex Oudsen, 4325494

library ieee;
use ieee.std_logic_1164.all;

entity mod60_tel_bcd is
   port(clk     :in    std_logic;  -- 32 kHz systeemklok
        clk_in  :in    std_logic;  -- Sturende klok(lokaal gegenereerd)
        reset   :in    std_logic;  -- Systeemreset
        sync_now:in    std_logic;  -- Enable signaal voor synchronisatie
        ref     :in    std_logic_vector(6 downto 0);    -- Tijdsreferentie
        count   :out   std_logic_vector(6 downto 0);    -- Huidige tellerstand
    h_clk   :out   std_logic); -- Sturende klok voor volgende teller
end mod60_tel_bcd;
```

```
-- Joran Out, 4331958 & Alex Oudsen, 4325494
-- Deze mod60 teller telt elke rising edge van clk_in 1
-- bij de huidige waarde van count op in bcd codering
-- De waarde van count wordt echter gelijk gemaakt aan ref,
-- wanneer het signaal sync_now hoog wordt

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

architecture behaviour of mod60_tel_bcd is

    type m60_state is (clear, wait_high, wait_low, synchronising);
    signal state, new_state: m60_state;

    signal m_count, new_m_count: std_logic_vector(6 downto 0);
    signal h_temp, new_h_temp: std_logic;

begin
    count <= m_count;
    h_clk <= h_temp;

    process(clk) is
```

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

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

B.1.9. MOD24 (UREN) TELLER

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

```
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
```

```
                        new_h_count <= (others => '0');
                    elsif(h_count(3 downto 0) = "1001") then
                        new_h_count <= h_count + 7;
                    else
                        new_h_count <= h_count + 1;
                    end if;
                    new_state <= wait_low;
                else
                    new_h_count <= h_count;
                    new_state <= wait_high;
                end if;
            when wait_low =>
                if(clk_in = '0') then
                    new_h_count <= h_count;
                    new_state <= wait_high;
                else
                    new_h_count <= h_count;
                    new_state <= wait_low;
                end if;
            when synchronising =>
                new_h_count <= ref;
                if(clk_in = '1') then
                    new_state <= wait_low;
                else
                    new_state <= wait_high;
                end if;
            when others =>          -- Zou nooit mogen voorkomen
                new_h_count <= (others => '0');
                new_state <= clear;
        end case;
    end process;
end behaviour;
```

B.1.10. AUTONOME SYNCHRONISEERBARE KLOK

```
-- Alex Oudsen, 4325494

library ieee;
use ieee.std_logic_1164.all;

entity ausy_klok_bcd is
    port (clk:          in  std_logic;  -- 32 kHz systeemklok
      s_clk:        in  std_logic;  -- 1 Hz klok uit klokdeler
      reset:        in  std_logic;  -- Systeemreset
      sync_now:     in  std_logic;  -- Enable signaal voor synchronisatie
      min_ref:      in  std_logic_vector(6 downto 0);  -- Referentietijd (minuten bcd)
      hr_ref:       in  std_logic_vector(5 downto 0);  -- Referentietijd (uren bcd)
      minutes:      out std_logic_vector(6 downto 0);  -- Huidige tijd (minuten bcd)
      hours:        out std_logic_vector(5 downto 0)); -- Huidige tijd (uren bcd)
end ausy_klok_bcd;
```

```
-- Alex Oudsen, 4325494
-- Deze autonome, synchroniseerbare, bcd gecodeerde klok wordt gebruikt om
-- de huidige tijd bij te houden, ook als het dcf signaal niet beschikbaar is
-- Wanneer het dcf signaal wel beschikbaar is,
-- wordt de klok gesynchroniseerd met dit signaal

-- Voor de bcd gecodeerde klok wordt gebruik gemaakt van de volgende subblokken:
-- mod24_tel_bcd, mod60_tel_bcd & mod60_clk_bcd

library ieee;
use ieee.std_logic_1164.all;

architecture structure of ausy_klok_bcd is
    component mod60_clk_bcd is
        port(clk:       in    std_logic;
                    clk_in:         in    std_logic;
                    reset:      in    std_logic;
             sync_now:      in    std_logic;
                    ref:        in    std_logic_vector(6 downto 0);
             m_clk:     out   std_logic);
```

```
    end component mod60_clk_bcd;

    component mod60_tel_bcd is
        port (clk:      in  std_logic;
              clk_in:       in  std_logic;
              reset:        in  std_logic;
              sync_now: in  std_logic;
              ref:      in  std_logic_vector(6 downto 0);
              count:    out std_logic_vector(6 downto 0);
              h_clk:    out std_logic);
    end component mod60_tel_bcd;

    component mod24_tel_bcd is
            port (clk:      in  std_logic;
              clk_in:       in  std_logic;
              reset:        in  std_logic;
              sync_now: in  std_logic;
              ref:      in  std_logic_vector(5 downto 0);
              count:    out std_logic_vector(5 downto 0));
    end component mod24_tel_bcd;

    signal m_clk: std_logic;
    signal h_clk: std_logic;
    signal sec_ref: std_logic_vector(6 downto 0);

begin

    sec_ref <= "0000000";

    SEC: mod60_clk_bcd port map(clk, s_clk, reset, sync_now, sec_ref, m_clk);
    MIN: mod60_tel_bcd port map(clk, m_clk, reset, sync_now, min_ref, minutes, h_clk);
    HRS: mod24_tel_bcd port map(clk, h_clk, reset, sync_now, hr_ref, hours);

end structure;
```

B.1.11. DCF77 (TOP-LEVEL)

```
-- Alex Oudsen, 4325494

library ieee;
use ieee.std_logic_1164.all;

entity dcf77_bcd is
    port (clk:            in  std_logic;
      reset:        in  std_logic;
      dcf_in:        in  std_logic;
      dcf_led:      out std_logic;
      clk_1hz:       out std_logic;
      minutes:       out std_logic_vector(6 downto 0);
      hours:        out std_logic_vector(5 downto 0);
      weekday:      out std_logic_vector(2 downto 0);
      day:        out std_logic_vector(5 downto 0);
      month:         out std_logic_vector(4 downto 0);
      year:       out std_logic_vector(7 downto 0);
      date_ready:   out std_logic);
end dcf77_bcd;
```

```
-- Alex Oudsen, 4325494
-- Dit is de top-level beschrijving van
-- de bcd versie van het dcf77 blok

-- Er wordt gebruik gebmaakt van de volgende subblokken:
-- synctime, klokdeler en ausy_klok_bcd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

architecture structure of dcf77_bcd is
    component synctime is
            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 klokdeler is
28              port (clk:      in  std_logic;
29                reset:        in  std_logic;
30                clk_1hz:  out std_logic);
31      end component klokdeler;
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_1hz <= s_clk;
51
52      sytime: synctime            port map(clk, reset, dcf_in, dcf_led, sync, minuut, uur,
            weekday, day, month, year);
53      divide: klokdeler               port map(clk, reset, s_clk);
54      r_time: ausy_klok_bcd           port map(clk, s_clk, reset, sync, minuut, uur, minutes,
            hours);
55
56  end structure;
```

B.2. VHDL CODE CONTROLLER

B.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;
```

B.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 menory_portmap: geheugen port map (clk,reset,write_enable,wekker_men,wekdata_men);
41 wekker <= wekdata_men;
42
43 end behaviour;
```

B.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;
```

B.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 opgande klokflank gebeurt, en gaat naar de
       nieuwe state toe (of de reset-state)
4  -- 2de process voert de state uit
5  -- 3de process bepalen van de nieuwe state
6  library IEEE;
7  use IEEE.std_logic_1164.ALL;
8  use IEEE.numeric_Std.all;
9
10 architecture behaviour of menu is
11 type fsm_states is (rust, wekkertijd, led, led_toggle, geluid, geluid_toggle, wekker_toggle,
       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
                             en enkele weer terug naar 0
80                          wekker(10 downto 7) <= "0000";
81                          wekker(12 downto 11) <= std_logic_vector(to_unsigned(to_integer(
                                unsigned(wekdata(12 downto 11))) + 1 , 2));
82                      else
83                          wekker(10 downto 7) <= std_logic_vector(to_unsigned(to_integer(
                                unsigned(wekdata(10 downto 7))) + 1 , 4)); -- 1 minuut erbij
                                optellen
84                          wekker(12 downto 11) <= wekdata(12 downto 11); -- Tientallen blijven
                                constant
85                      end if;
86                  end if;
87                  wekker(15 downto 13) <= wekdata(15 downto 13); -- Af
88                  wekker(6 downto 0) <= wekdata(6 downto 0); --Af
89
90              when uren_min =>
91                  if wekdata(12 downto 7) = "000000" then
92                      wekker(12 downto 7) <= "100011"; --23
93                  else
94                      if wekdata(10 downto 7) = "0000" then
95                          wekker(10 downto 7) <= "1001";
96                          wekker(12 downto 11) <= std_logic_vector(to_unsigned(to_integer(
                                unsigned(wekdata(12 downto 11))) - 1 , 2));
97                      else
98                          wekker(10 downto 7) <= std_logic_vector(to_unsigned(to_integer(
                                unsigned(wekdata(10 downto 7))) - 1 , 4));
99                          wekker(12 downto 11) <= wekdata(12 downto 11);
100                     end if;
101                 end if;
102                 wekker(15 downto 13) <= wekdata(15 downto 13);
103                 wekker(6 downto 0) <= wekdata(6 downto 0);
104                 enable <= '1';
105                 menu_signal <= "101";
106
107             when minuten_set =>
108                 enable <= '0';
109                 wekker <= wekdata;
110                 menu_signal <= "010";
111
112             when minuten_plus =>
113                 enable <= '1';
114                 if wekdata(6 downto 0) = "1011001" then --59
115                     wekker(6 downto 0) <= "0000000"; --Bij de 59 minuten gaan weer op nieuw
                            beginnen
116                 else
117                     if wekdata(3 downto 0) = "1001" then --Bij x9 minuten 1 op tellen bij de
                            x en enkele weer terug naar 0
118                         wekker(3 downto 0) <= "0000";
```

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

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

B.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 geheugen;
```

B.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;
```

B.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;
```

B.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;
```

```
            end if;
            knoppen_out <= knoppen_temp;
        end if;
    end process assign;
    actie_uitvoeren : process(knoppen_in,clk, reset, state) --Voer acties uit
    begin
        case state is
            when rust =>
                if ((knoppen_in(0) = '1' xor knoppen_in(1) = '1') xor (knoppen_in(2) = '1'
                    xor knoppen_in(3) = '1')) then
                    new_state <= one;
                    knoppen_temp <= knoppen_in;
                else
                    new_state <= state;
                    knoppen_temp <= "0000";
                end if;
            when zero =>
                knoppen_temp <= "0000";
                if ((knoppen_in(0) = '0' and knoppen_in(1) = '0') and (knoppen_in(2) = '0'
                    and knoppen_in(3) = '0')) then
                    new_state <= rust;
                else
                    new_state <= state;
                end if;
            when one =>
                new_state <= zero;
                knoppen_temp<="0000";
            when others =>
                new_state <= rust;
                knoppen_temp <= "0000";
        end case;
    end process actie_uitvoeren;
end behaviour;
```

B.3. TESTBENCHS VOOR DE CONTROLLER

B.3.1. VHDL CONTROLLER

```
-- Kevin Hill 4287592  & Rens Hamburger 4292936
-- Testbench voor de controller met gebruik van de buffer en geheugen element
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.Numeric_Std.all;

architecture behaviour of controller_tb is
component controller is
    port(clk    :in    std_logic;
        reset  :in    std_logic;
        knoppen:in    std_logic_vector(3 downto 0);
        wekker :out   std_logic_vector(15 downto 0);
        menu_state   :out   std_logic_vector(2 downto 0));
end component controller;

signal clk, reset                           :   std_logic;
signal menu_signal                          :   std_logic_vector(2 downto 0);
signal knoppen                              : std_logic_vector (3 downto 0);
signal wekker       :   std_logic_vector (15 downto 0);

begin
    clk     <=  '1' after 0 ns,
            '0' after 40 ns when clk /= '0' else '1' after 40 ns;       --31250

    reset   <=  '1' after 0 ns,     --knoppen(0) = menu
            '0' after 128 ns;       --knoppen(1) = set
                                    --knoppen(2) = up
    knoppen <=  "0000" after 0 ns,  --knoppen(3) = down
            "0010" after 128 ns,    --rust -> wekker_toggle
            "0000" after 208 ns,    --knoppen(3) = down
            "0001" after 608 ns,    --rust -> wekkertijd
            "0000" after 688 ns,    --knoppen(3) = down
            "0001" after 848 ns,    --wekkertijd -> rust
```

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

B.3.2. TESTBENCH VHDL MENU

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

```
5  use IEEE.std_logic_1164.ALL;
6  use IEEE.Numeric_Std.all;
7
8  architecture behaviour of menu_test is
9  component menu is        --component initialiseren, met de volgende in/uitgangen:
10     port(clk        :in        std_logic;
11         reset        :in        std_logic;
12         knoppen        :in        std_logic_vector    (3 downto 0);    --dit zijn de fysieke
               knoppen
13         wekdata        :in        std_logic_vector    (15 downto 0);    --komt bij het
               register vandaan
14         enable        :out    std_logic;
15         wekker        :out    std_logic_vector    (15 downto 0);
16         menu_signal    :out    std_logic_vector    (2 downto 0)); --voor de LCD'
17 end component menu;
18
19 signal clk, reset, enable    :    std_logic;
20 signal menu_signal            :    std_logic_vector (2 downto 0);
21 signal knoppen,minuten_enkel,uren_enkel            :    std_logic_vector (3 downto 0);
            --signalen voor de port map
22 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
            daadwerkelijke port map
86  end architecture;
```

B.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;
```

B.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,knoppjes        :       std_logic_vector(3 downto 0);
16
17
18 begin
19     clk     <=  '0' after 0 ns,
20             '1' after 20 ns  when clk /= '1' else '0' after 20 ns;
21
22     reset   <=  '1' after 0 ns,
23             '0' after 85 ns;
24
25     knoppen <=  "0000" after 0 ns,
26                 "1111" after 100 ns,
27                 "0000" after 150 ns,
28                 "1000" after 190 ns,
29                 "0000" after 240 ns,
30                 "0001" after 290 ns;
31
32     buff_pm: buff port map(clk,reset,knoppen,knoppjes);
33 end behaviour;
```

B.4. VHDL CODE VAN HET ALARM

B.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;
```

B.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)
                            ) - 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));
28                      end if;
29                  end if;
30                  state <= new_state;
31              end if;
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;
```

B.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;
```

B.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,
            length => length);
21      pwm_1 : pwm port map (clk => clk, reset => reset, length => length, pwm_signal =>
            pwm_signal);
22 end behaviour;
```

B.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;
```

B.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;
```

B.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;
```

B.4.8. BEHAVIOURAL ALARM-PWM

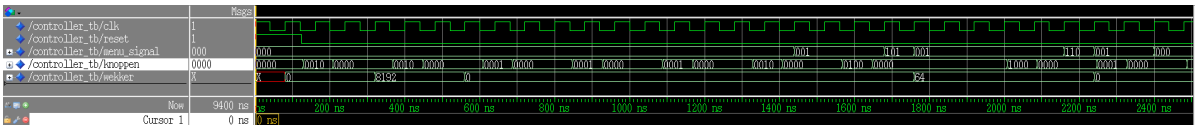
```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  architecture behaviour of pwm is
6      type pwm_state is (hoog, laag, res_state);
7      signal counter, new_counter: unsigned(5 downto 0);
8      signal state, new_state: pwm_state;
9  begin
10     lbl1: process(clk)
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;
```

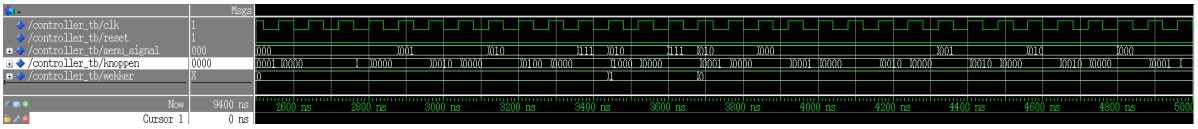
C

SIMULATIES RESULTATEN VAN DE CONTROLLER

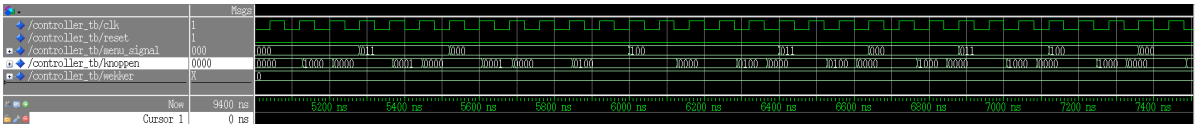
C.1. BEHAVIORAL SIMULATIE



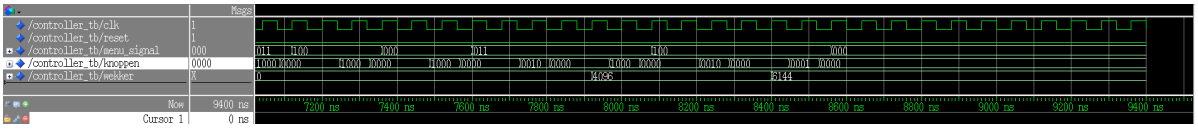
Figuur C.1: Simulatie van 0 tot 2500ns



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

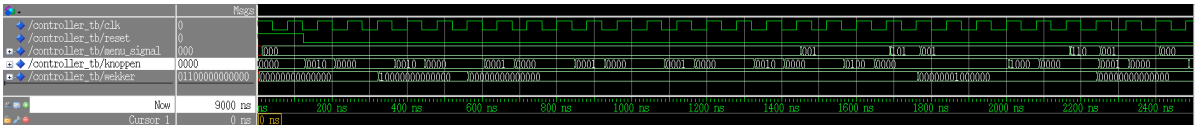


Figuur C.3: Simulatie van 5000ns tot 7500ns

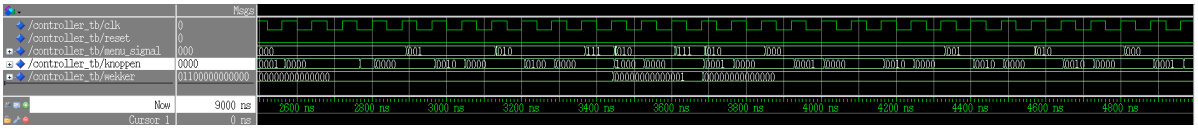


Figuur C.4: Simulatie van 7500ns tot het einde

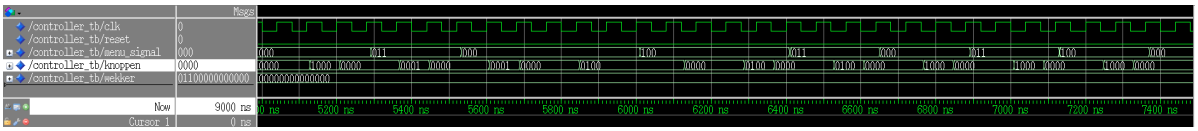
C.2. SYNTHESIZE SIMULATIE



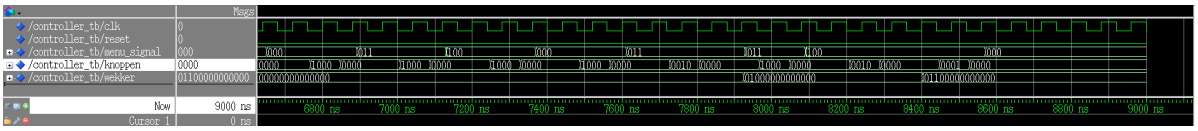
Figuur C.5: Simulatie van 0 tot 2500ns



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

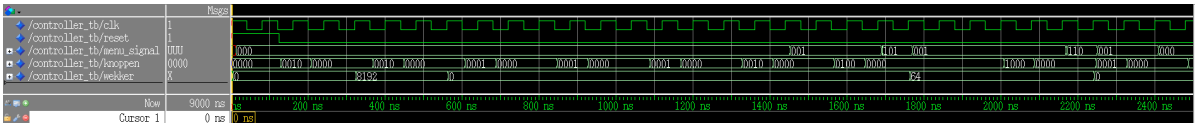


Figuur C.7: Simulatie van 5000ns tot 7500ns

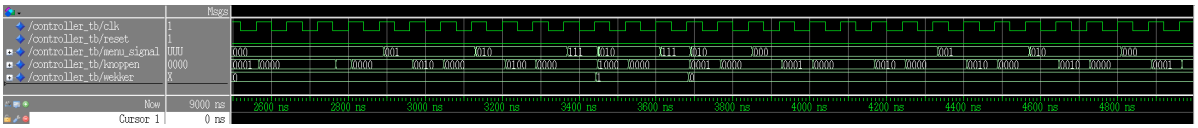


Figuur C.8: Simulatie van 7500ns tot het einde

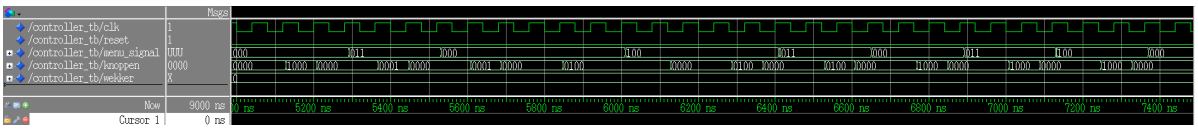
C.3. EXTRACTED SIMULATIE



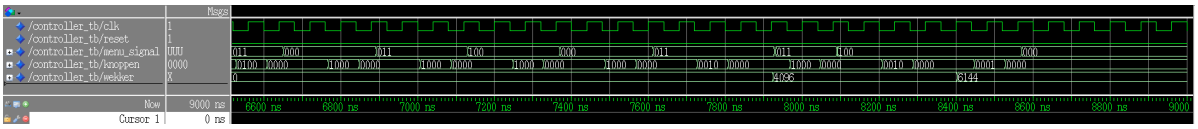
Figuur C.9: Simulatie van 0 tot 2500ns



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

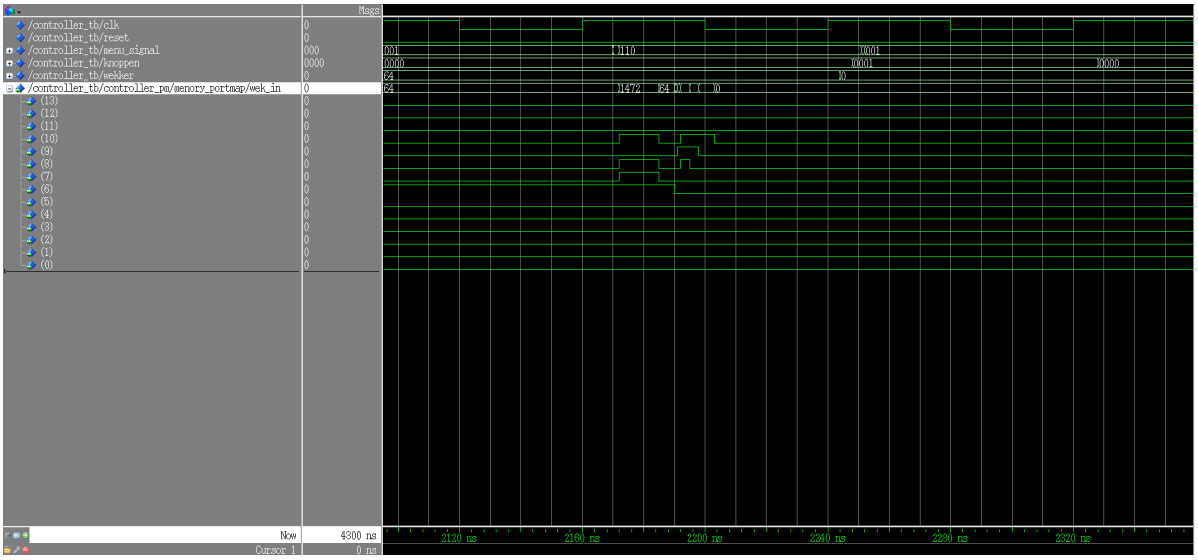


Figuur C.11: Simulatie van 5000ns tot 7500ns



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

C.4. TIMING



Figuur C.13: Timing problemen

BIBLIOGRAFIE

[1] Wikipedia, *Dcf77,* Geraadpleegd op 08-12-2014, url: en.wikipedia.org/wiki/DCF77.

[2] Physikalisch-Technische Bundesanstalt (PTB), *Dcf77 zeitkode,* Geraadpleegd op 08-12-2014, url: www.ptb.de/cms/fachabteilungen/abt4/fb-44/ag-442/verbreitung-der-gesetzlichen-zeit/dcf77/zeitcode.html.

[3] C. J. Wells, *Mathematics - number systems - binary-coded-decimal,* Geraadpleegd op 12-12-2014, url: www.technologyuk.net/mathematics/number_systems/binary_coded_decimal.shtml.