

## EE2821 EPO-3: Project “maak een chip”

Een implementatie van  
Conway's Game of Life  
op een  
Sea of Gates chip

21 december 2012

## Groep A2

Lisa Audenaert	4152654
Patrick Fuchs	1527436
Bas Generowicz	4029542
Ahmet Gerçekcioglu	4042700
Arjan van der Kruijt	4165705
Leon Noordam	4139526
Gert-Jan van Raamsdonk	4143264
Pim Veldhuisen	4153448
Niels van Wijngaarden	4063899

## Samenvatting

Dit verslag beschrijft een ontwerp van een Sea-Of-Gates chip die een levensproces simuleert volgens de regels van Conway's Game of Life [1]. Voor deze toepassing is een systeem ontworpen bestaande uit verschillende delen. Het systeem is beschreven in VHDL, vervolgens gesynthetiseerd en er is een layout gemaakt voor plaatsing op een Sea-Of-Gates chip. De uiteindelijke chip wordt bediend aan de hand van drukknoppen en het resultaat, een 'levend'  $8 \times 8$  grid, wordt weergegeven op een monitor via een VGA aansluiting. In dit verslag staat het ontwerpen, implementeren en testen van de schakeling beschreven.

# Inhoudsopgave

<b>1</b>	<b>Inleiding</b>	<b>4</b>
<b>2</b>	<b>Systeemspecificaties</b>	<b>5</b>
2.1	Functie van de schakeling . . . . .	5
2.2	Functionele eisen . . . . .	6
2.3	Gebruikersinterface . . . . .	6
2.4	Externe randvoorwaarden . . . . .	7
<b>3</b>	<b>De Deelsystemen</b>	<b>10</b>
3.1	Inleiding . . . . .	10
3.2	Opdeling . . . . .	10
3.3	Interface tussen de deelsystemen . . . . .	11
<b>4</b>	<b>Systeemcontroller</b>	<b>12</b>
4.1	Specificaties . . . . .	12
4.2	Implementatie . . . . .	13
4.2.1	Uitlezen van de inputsignalen . . . . .	13
4.2.2	State bepalen . . . . .	14
4.3	VHDL simulatie . . . . .	15
4.4	Synthese . . . . .	15
4.5	Switch-level simulatie . . . . .	16
4.6	Conclusie . . . . .	16
<b>5</b>	<b>Evaluatie &amp; Geheugen</b>	<b>18</b>
5.1	Specificaties . . . . .	18
5.2	Implementatie . . . . .	18
5.3	VHDL simulatie . . . . .	21
5.4	Synthese & Layout . . . . .	23
5.5	Switch-level simulatie . . . . .	23
5.6	Conclusie . . . . .	25
<b>6</b>	<b>RGB-generator</b>	<b>26</b>
6.1	Specificaties . . . . .	26
6.2	Implementatie . . . . .	26
6.3	VHDL simulatie . . . . .	27
6.4	Synthese & Layout . . . . .	28
6.5	Switch-level simulatie . . . . .	28
6.6	Conclusie . . . . .	28

<b>7</b>	<b>VGA-timer</b>	<b>30</b>
7.1	Specificaties	30
7.2	Implementatie	31
7.3	VHDL simulatie	33
7.4	Synthese & Layout	34
7.5	Switch-level simulatie	34
7.6	Conclusie	34
<b>8</b>	<b>Klok</b>	<b>36</b>
<b>9</b>	<b>Resultaat totaalontwerp</b>	<b>38</b>
9.1	Inleiding	38
9.2	Simulatie	38
9.3	FPGA Prototyping	38
9.4	Synthese en layout	38
9.5	Switch-level-simulatie	39
<b>10</b>	<b>Testplan voor de chip</b>	<b>40</b>
10.1	Testen met de logic analyser	40
10.1.1	Losse cel	40
10.1.2	Deelsystemen direct aansturen	40
10.1.3	Tussensignalen uitlezen	41
10.1.4	Conclusie	42
<b>11</b>	<b>Verloop van het project</b>	<b>44</b>
11.1	FPGA Prototyping	44
11.2	Synthetiseren	45
<b>12</b>	<b>Conclusie</b>	<b>46</b>
	<b>Bibliografie</b>	<b>47</b>
<b>A</b>	<b>Simulatie van het complete systeem</b>	<b>48</b>
<b>B</b>	<b>VHDL-code</b>	<b>49</b>
B.1	Top-level	49
B.2	Systeem-controller	53
B.3	Evaluatie & Geheugen	57
B.3.1	Evaluatie & Geheugen Toplevel	57
B.3.2	Clock-divider	58
B.3.3	Pixels	59
B.3.4	Pixel	61
B.4	RGB-generator	62
B.4.1	VGA_top	62
B.4.2	VGA_control	63
B.4.3	VGA_game	64
B.4.4	VGA_text	67
B.5	VGA-timer	70

# Hoofdstuk 1

## Inleiding

Het project EPO-3 in het tweede jaar in de opleiding EWI-BSc Elektrotechniek vraagt de studenten om een ontwerp te maken, uit te werken en te beschrijven in VHDL zodat deze uiteindelijk op een SOG-chip gezet kan worden. De projecthandleiding geeft enkele ideeën en bijbehorende specificaties [2]. Toch hebben we voor een origineel ontwerp gekozen waarbij zelf de complexiteit bepaald kon worden. Gezamenlijk is besloten om de uitdagende simulatie van “Conway’s Game of Life” te realiseren.

Dit rapport specificeert een systeem dat Conway’s Game of Life simuleert en beschrijft een implementatie hiervan. Deze implementatie is gericht op de EPO-3 Sea-Of-Gates chip, voldoet aan de ontwerperegels en is haalbaar binnen de beschikbare tijd.

Het eerste onderdeel van dit rapport is de specificatie van het systeem. Daarna wordt de implementatie besproken; hierbij is het gehele systeem onderverdeeld in deelsystemen met elk een eigen, goed gedefiniëerde functie. Na het implementeren van ieder deelsysteem zal deze gesimuleerd worden. Dit zal gebeuren middels een VHDL-simulatie. Na de VHDL-simulatie zal het deelsysteem gesynthetiseerd worden. Vervolgens zal er een lay-out worden gemaakt. Bij dit laatste proces is het streven om zo min mogelijk chipoppervlak te gebruiken. Als laatste wordt er een switch-level simulatie uitgevoerd. De switch-level en de VHDL simulaties zullen bij ieder deelsysteem beschreven worden.

Nadat alle deelsystemen correct werken, zullen deze samengevoegd en gekoppeld worden. Ook wordt het totaalontwerp gesimuleerd. Aan de hand van een testbench zal er nagegaan worden of de combinatie van alle deelsystemen correct werkt. Hierna zal het totaalontwerp op een FPGA-chip geplaatst worden. De FPGA-chip wordt op een monitor aangesloten en zal dan Conway’s Game of Life moeten simuleren. Wanneer dit naar tevredenheid werkt, zal het totaalontwerp gesynthetiseerd worden. Bij de synthese is te zien hoeveel transistoren het totaalontwerp nodig heeft.

Tenslotte wordt er een testplan besproken om de deelsystemen op de chip in het geheel en individueel te kunnen testen.

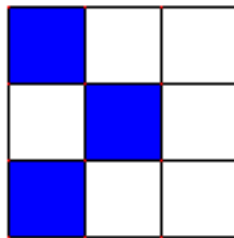
## Hoofdstuk 2

# Systemspecificaties

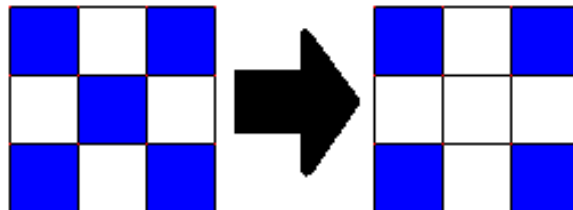
### 2.1 Functie van de schakeling

De schakeling die wij gaan ontwerpen moet Conway's Game of Life[1] simuleren. Dit is een simulatie die bestaat uit een rechthoekig raster van cellen. Deze cellen kunnen 'levend' of 'dood' zijn. De simulatie verloopt in stappen en bij elke volgende stap wordt bepaald wat de status van elke cel is aan de hand van de vorige staat van zichzelf en zijn burenen. Een cel heeft in totaal 8 burenen. De volgende regels bepalen de status van een cel:

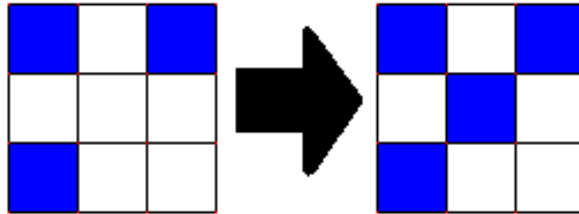
- Wanneer een cel leeft en er minder dan 2 burenen leven, sterft de cel aan eenzaamheid.
- Wanneer een cel leeft en 2 of 3 burenen leven, overleeft de cel, zie figuur 2.1.
- Wanneer een cel leeft en 4 of meer burenen leven, sterft de cel aan overbevolking, zie figuur 2.2.
- Wanneer een cel dood is en 3 burenen heeft, komt de cel tot leven, zie figuur 2.3.



**Figuur 2.1:** In de buurt zijn 2 cellen, de middelste cel blijft leven



**Figuur 2.2:** Overbevolking; in de buurt zijn 4 cellen, de middelste cel zal sterven



**Figuur 2.3:** Er zijn 3 burens, de middelste cel gaat leven

De simulatie start met een beginsituatie die de gebruiker zelf in kan stellen. Aan de hand van de beginsituatie telt het systeem per cel hoeveel levende burens deze heeft. Aan de hand van het aantal levende burens in de huidige situatie moet de schakeling berekenen of de cel in de volgende situatie leeft of dood is. Vervolgens moet de schakeling dit opslaan. Hierna moet het systeem de berekende situatie op een VGA monitor weergeven. De gebruiker moet op ieder gewenst moment de simulatie kunnen pauzeren of resetten.

## 2.2 Functionele eisen

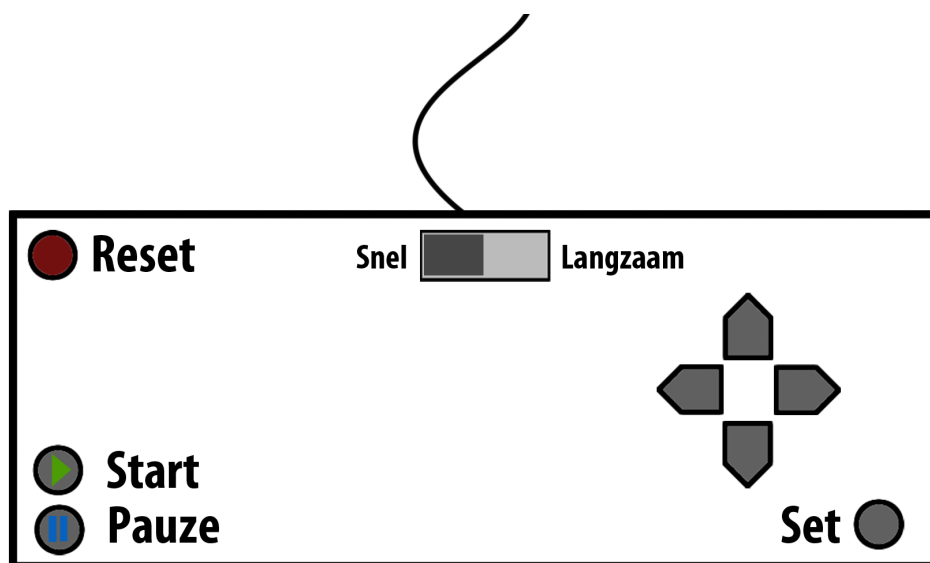
Om bovenstaande functie goed uit te kunnen voeren, hebben we de volgende functionele eisen samengesteld:

- De schakeling moet in een redelijk tijdsbestek (ca. 1 seconde) het gehele speelveld evalueren en een nieuwe iteratie tonen, anders wordt de simulatie te langzaam.
- De schakeling moet minimaal een  $8 \times 8$  grid kunnen sturen en uitrekenen, anders wordt het speelveld te klein.
- De speler moet elke cel van dit raster kunnen aan- of uitzetten voordat hij het spel start.
- Er moeten 9 knoppen beschikbaar zijn voor de bediening van de simulatie. Vier om de positie te bepalen, één om de cellen te zetten, één om het spel te beginnen, één om het spel op pauze te zetten, één om het spel te resetten en één om de simulatiesnelheid in te stellen, zie figuur 2.4.
- Bij het doorlopen van het veld moet de cursor, die de positie aanwijst, door de rand heen kunnen lopen en op dezelfde rij/kolom aan de andere kant uitkomen.
- Bij het evalueren van het veld, moet er ook door de rand heen geëvalueerd kunnen worden, zodat oneindig doorgaande patronen aan het begin van het veld weer binnen komen en zo werkelijk oneindig blijven leven.
- Er moet een bitsignaal worden gegenereerd om het speelveld van  $8 \times 8$  samen te kunnen vatten, zodat dit signaal vertaalt kan worden om de cellen weer te geven op een monitor.

## 2.3 Gebruikersinterface

De gebruiker kan het systeem aansturen met 9 knoppen en een schakelaar. Een overzicht hiervan is te zien in figuur 2.4.

De reset-knop wordt gebruikt om de simulatie terug te brengen in zijn begintoestand. Na het resetten kan de gebruiker een nieuwe beginsituatie instellen. Bij het invoeren van de begintoestand wordt de locatie van een geselecteerde cel weergegeven op het scherm. Welke cel is geselecteerd kan worden veranderd door middel van de vier richtingsknoppen. Een cel kan vervolgens ‘levend’



**Figuur 2.4:** Een overzicht van de knoppen en de schakelaar die beschikbaar zijn voor de gebruiker

worden gemaakt door de set-knop in te drukken. Hierna kunnen naar behoeven meer cellen levend worden gemaakt op dezelfde wijze. Als de gewenste toestand is bereikt kan de simulatie worden gestart met de start-knop. Het spel kan vervolgens gepauzeerd worden met de pauze-knop, om te kunnen zien wat voor wonderbaarlijk leven je hebt gemaakt. Na de pauze-knop kan je ook weer de start-knop indrukken waardoor het spel verder gaat leven.

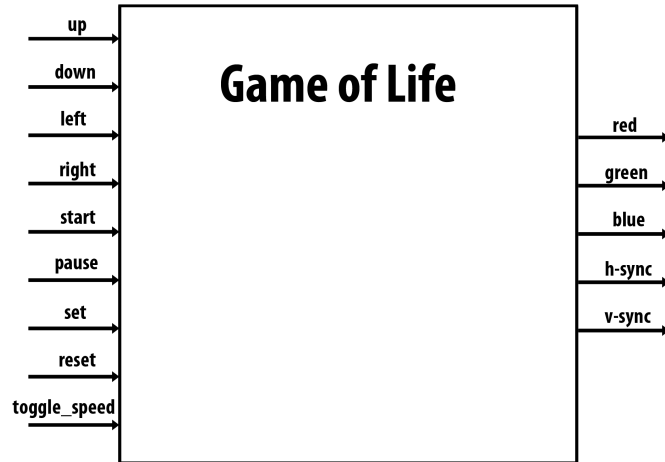
Tijdens de simulatie kan de evaluatie-snelheid worden ingesteld op twee standen met een schakelaar. De simulatie geeft zijn output door middel van een signaal volgens de VGA-standaard. Het VGA-signaal bestaat uit de bits Red, Green, Blue, h-sync en v-sync. Dit signaal wordt in meer detail besproken in de sectie VGA-timer. De in- en uitgangen van het systeem zijn te zien in figuur 2.5.

Het VGA signaal is bedoeld om aangesloten te worden op een monitor met een beeld van  $640 \times 480$ . Op de monitor wordt dan een raster getoond met de cellen. Dode cellen zijn hier zwart, levende cellen zijn rood. De cursor wordt op een levende cel weergegeven met geel, op een dode cel met groen. Boven het raster wordt de tekst “Game of life” in het blauw weergegeven. De achtergrond van het scherm is zwart. In figuur 2.6 zijn de beschreven beelden te zien in een aantal voorbeeld-situaties.

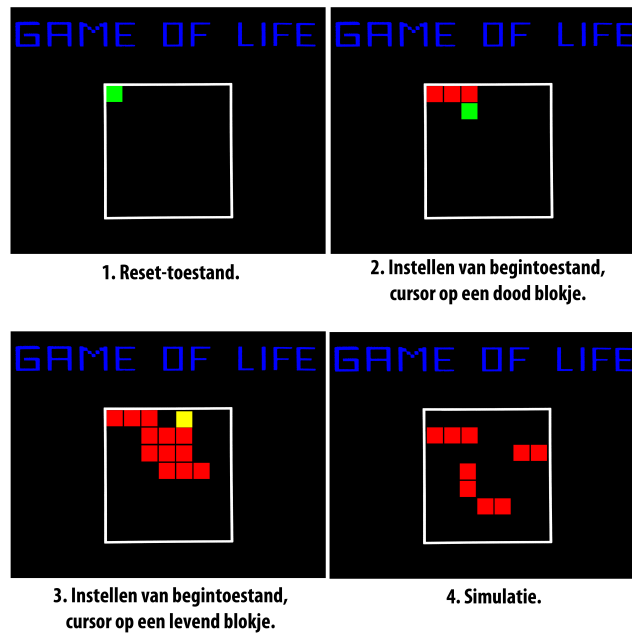
## 2.4 Externe randvoorwaarden

De volgende randvoorwaarden worden gegeven in de projecthandleiding [2]. Met deze randvoorwaarden dient rekening gehouden te worden in verband met het vergroten van de slaagkans van het IC ontwerp.

- Voor de FSM's (Finite State Machine) mogen alleen deze van het Moore-type gebruikt worden.
- Als de schakeling geactiveerd wordt, moeten alle FSM's in hun resettoestand komen door middel van een resetsignaal.



**Figuur 2.5:** De in- en uitgangen van het systeem.



**Figuur 2.6:** Een aantal voorbeeld-weergaven van het systeem.

- Er moet bij het ontwerpen rekening gehouden worden met het feit dat de schakeling later getest zal worden en dat van de verschillende besturingen ook de huidige toestand gelezen moet kunnen worden.
- Voor het genereren van het kloksignaal kan gebruik gemaakt worden van een kristal van  $6.144\text{ MHz}$  of  $32\text{ kHz}$ .
- Het streven is om zo weinig mogelijk componenten extern te gebruiken. De dissipatie van de chip dient echter ook beperkt te zijn. Dit geeft een compromis voor de maximale stroom die de elektronica mag dissiperen voor de aansturing van de LEDs, etc.



- De voedingsspanning van het IC bedraagt 5 Volt.
- Het IC wordt gemaakt in een CMOS-proces.
- Het beschikbare chip-oppervlak is circa  $0.4 \text{ cm}^2$ , hetgeen overeenkomt met ongeveer 40.000 transistorparen, die gelijk zijn verdeeld over 2 bond\_bars.
- Er zijn op het IC, naast de voedingspinnen, per bond\_bar 32 aansluitingen beschikbaar. Hoewel een totale schakeling meer dan één bond\_bar aan oppervlakte kan hebben, kunnen bij de montage slechts de aansluitingen van één bond\_bar worden aangesloten.

**Technologieparameters Sea-of-Gates**

$$V_{DD} = 5V$$

$$V_{T0} = 0.7V$$

$$t_{ox} = 25 * 10^{-9}m$$

$$N_{mos_{L \times W}} = 1.6 \times 23.2\mu m$$

$$P_{mos_{L \times W}} = 1.6 \times 29.6\mu m$$

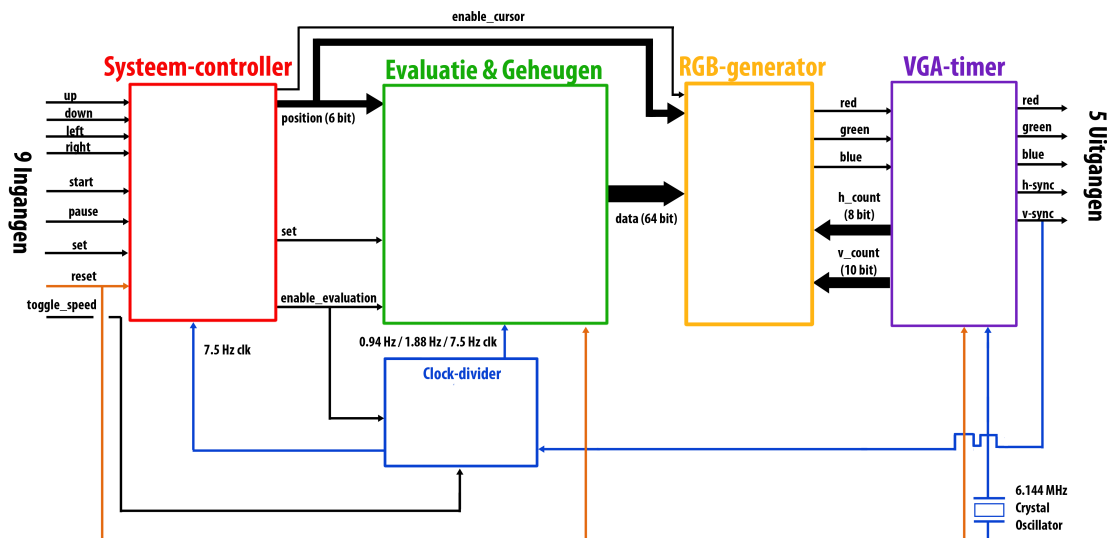
## Hoofdstuk 3

# De Deelsystemen

### 3.1 Inleiding

Het systeem is opgesplitst in kleinere deelsystemen, met elk een aparte, goed gedefiniëerde functie. Ook de interactie tussen deze deelsystemen is nauwkeurig afgesproken, zodat de deelsystemen gecombineerd kunnen worden tot één functionerend systeem. Het opdelen van het systeem in deelsystemen heeft als voordeel dat een groter probleem wordt opgedeeld in kleinere, makkelijker op te lossen deelproblemen. Daarnaast kunnen de teamleden op deze manier efficiënt werken aan hun eigen deel van het systeem. Tevens zorgt het er voor dat het makkelijker is om een eventuele fout in het systeem op te sporen.

### 3.2 Opdeling



**Figuur 3.1:** Overzicht van het systeem met deelsystemen en signalen

De centrale functie van het systeem is het uitvoeren van de regels van Conway's Game of Life. Hiervoor is het noodzakelijk om de staat van de cellen op te slaan. Deze twee functies, die nauw aan elkaar verwant zijn, worden uitgevoerd door het deelsysteem Evaluatie & Geheugen

Conway's Game of Life moet niet continu gesimuleerd worden. De simulatie zal soms worden gepauzeerd en er moet een beginsituatie kunnen worden ingevoerd. Deze functies worden geïmplementeerd door het deelsysteem Systeem-controller.

Het is belangrijk dat de simulatieresultaten worden getoond aan de gebruiker. Voor dit doel zal het ontwerp een VGA-sigitaal genereren dat kan worden getoond op een monitor.

Deze functie wordt geïmplementeerd door twee deelsystemen. Het eerste deelsysteem is de VGA-timer, die de positie op de monitor bepaalt en het uiteindelijke signaal doorstuurt. De VGA-timer doet dit echter niet alleen; voor het bepalen van de kleurwaarde van een bepaalde pixel geeft de VGA-timer zijn positie door aan het deelsysteem RGB-generator, dat vervolgens de RGB-waarde van de huidige pixel teruggeeft.

### 3.3 Interface tussen de deelsystemen

We gaan nu dieper op elk deelonderwerp in, hierbij is figuur 3.1 het totaalplaatje waar naar gekeken moet worden tijdens het lezen van dit onderdeel. De systeem-controller stuurt allereerst het signaal 'enable\_evaluation' naar het deelsysteem Evaluatie & Geheugen. Dit signaal bepaalt of de regels van Game of Life worden gesimuleerd. Daarnaast stuurt de controller het 6-bit signaal 'position' uit naar zowel de Evaluatie & Geheugen als de RGB-generator. Dit signaal houdt bij op welke cel de cursor zich bevindt. Tijdens het invoeren van de beginsituatie wordt aan de hand van dit signaal de geselecteerde cel weergegeven. In de Evaluatie & Geheugen kan deze cel dan levend worden gemaakt met het ingangssignaal 'set'. Ook wordt er een signaal 'enable\_cursor' naar de RGB-generator gestuurd die ervoor zorgt dat de cursor weergegeven wordt.

Het deelsysteem Evaluatie & Geheugen stuurt de waarde van iedere cel door naar de RGB-generator. Dit signaal, dat uit 64 bits bestaat, wordt niet gemultiplexed. Hier is voor gekozen omdat de RGB-generator deze signalen ook allemaal los nodig heeft en de RGB-generator dicht bij de Evaluatie & Geheugen geplaatst kan worden. De verbindingen van de Evaluatie & Geheugen naar de RGB-generator zullen dus naar verwachting ongeveer evenveel ruimte in beslag nemen als de verbindingen van de Evaluatie & Geheugen naar een eventuele multiplexer zou zijn. Een schakeling met multiplexers en demultiplexers zou daarom extra ruimte innemen.

De RGB-generator krijgt van de VGA-timer de signalen 'h\_count' en 'v\_count', waarin de positie van de pixels die de VGA-timer op dat moment wil weergeven, gecodeerd zijn. Aan de hand hiervan, de signalen van de systeem-controller en Evaluatie & Geheugen, genereert de RGB-generator een RGB waarde[3]. Deze waarde wordt niet direct aan de monitor doorgegeven, maar gaat eerst naar de VGA-timer. De VGA-timer zorgt er dan voor dat tijdens de periode dat de elektronen-straal teruggaat, de RGB waarden op nul staan. De waarden die in deze periode door de RGB-generator worden doorgegeven, zijn dus niet relevant, en hierdoor kan de RGB-generator efficiënter werken door bepaalde controles weg te laten.

De VGA-timer stuurt aan de hand van de binnen gekregen RGB waarde een output RGB waarde naar de pixel welke op dat moment bepaald is door de coördinerende signalen 'h\_count' en 'v\_count'. 'H\_count' en 'v\_count' bepalen ook de waarden voor de signalen hsync en vsync. Wanneer hsync op 0 staat stuurt de VGA-timer geen RGB-waarden door, maar is de monitor bezig met het synchroniseren van de lijn. De werking van vsync is analoog met hsync, alleen wordt bij vsync het gehele scherm gesynchroniseerd.

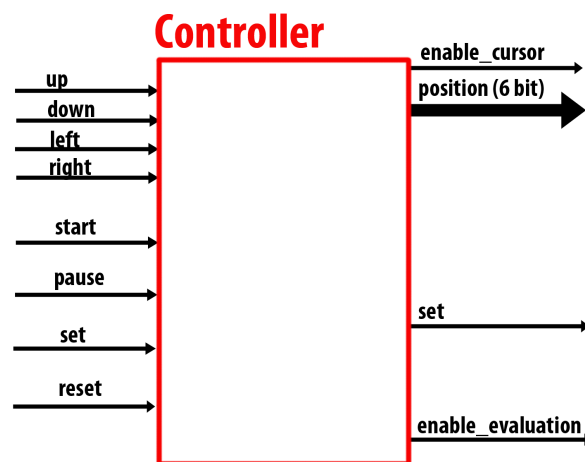
## Hoofdstuk 4

# Systemcontroller

### 4.1 Specificaties

Het deelblok systeemcontroller moet de volgende functies uitvoeren:

- Uitlezen van de inputsignalen
- State bepalen
- Signalen doorsturen die andere blokken nodig hebben



**Figuur 4.1:** Overzicht van ingangs- en uitgangssignalen

De systeem-controller heeft acht ingangssignalen (zie figuur: 4.1):

- |         |         |
|---------|---------|
| • Up    | • Start |
| • Down  | • Pause |
| • Left  | • Set   |
| • Right | • Reset |

De ingangssignalen zullen geïmplementeerd worden door middel van drukknoppen. De signalen up, down, left en right zijn voor het verplaatsen van de cursor. Wanneer start ingedrukt wordt, zal de simulatie van Game of Life beginnen. Nadat de simulatie begonnen is zorgt de pauze knop

ervoor dat de simulatie gepauzeerd wordt en de huidige situatie zichtbaar blijft. Set zorgt voor het levend maken van een cel, de cel wordt rood gekleurd. De resetknop zorgt ervoor dat de simulatie gereset wordt en dat de gebruiker opnieuw een beginsituatie in kan stellen.

De systeem-controller heeft vier uitgangssignalen (zie figuur: 4.1):

- Enable\_cursor
- Position
- Set
- Enable\_evaluation

De eerste twee signalen zijn om te zorgen dat tijdens het instellen van een beginsituatie de cursor op de monitor weergegeven wordt. Enable\_cursor wordt enkel naar de RGB-generator gestuurd. Als Enable\_cursor waarde '1' heeft, moet de cursor zichtbaar zijn en wanneer Enable\_cursor waarde '0' heeft, moet de cursor niet zichtbaar zijn. Het signaal Position houdt de cel bij waar de cursor zich op dat moment bevindt. Er is gekozen voor een raster van  $8 \times 8$  wat inhoudt dat er 64 cellen zijn. Aangezien het 64 cellen zijn is gekozen voor een 6-bitssignaal. Dit 6-bitssignaal houdt de waarde van de cel bij waar de cursor zich op dat moment bevindt (zie figuur 4.2). Dit wil zeggen dat bij de cel linksboven de waarde 0 heeft, wat in een 6-bitssignaal "000000" is. De cel rechtsonder heeft de waarde 63, wat in een 6-bitssignaal "111111" is. Dit signaal wordt zowel naar de RGB-generator als naar de Evaluatie & Geheugen gestuurd. De RGB-generator heeft deze waarde nodig zodat deze de cursor op de juiste positie weer kan geven. Op het moment dat een cel levend gemaakt wordt zal dit in het geheugen geplaatst moeten worden. Daarom wordt Position ook naar de Evaluatie & geheugen gestuurd. Het set signaal geeft aan als er een cel levend gemaakt moet worden. Is dit het geval dan zal er het set signaal '1' worden. Enable\_evaluation geeft aan dat de simulatie gestart of vervolgd moet worden.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Figuur 4.2: De nummering van het raster

## 4.2 Implementatie

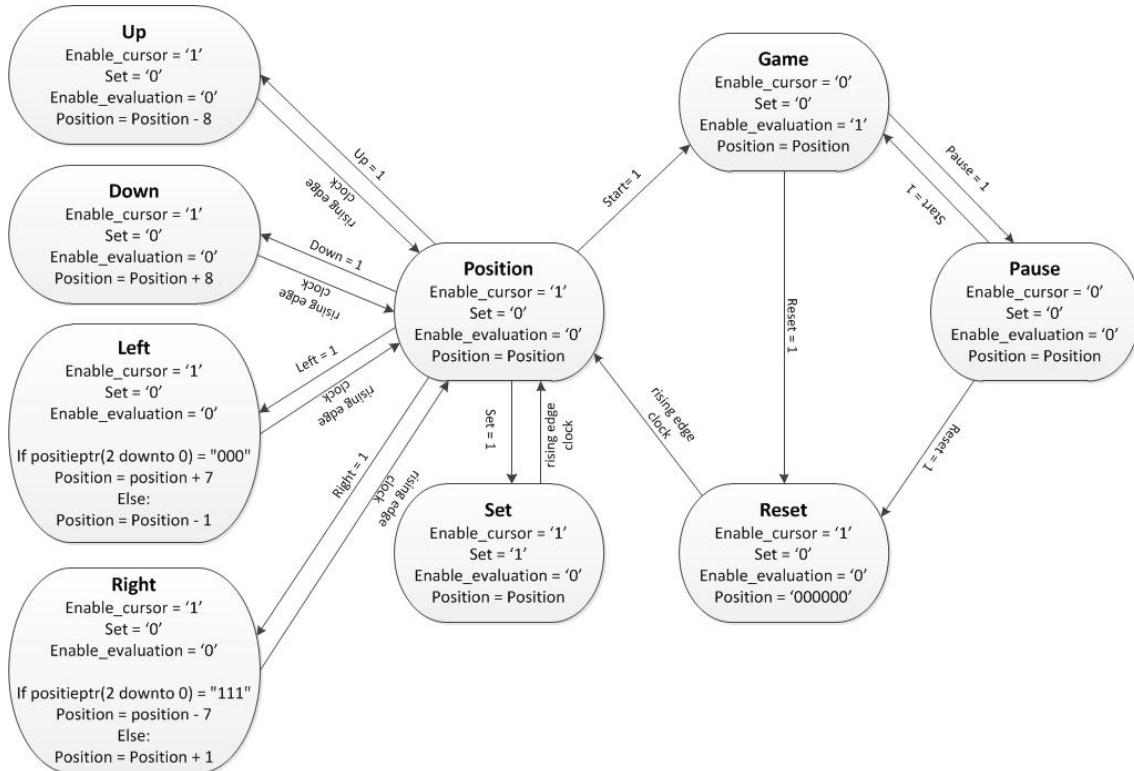
### 4.2.1 Uitlezen van de inputsignalen

De inputsignalen worden uitgelezen door middel van een inputbuffer. Deze wordt geïmplementeerd met één flipflop per ingangssignaal die het variabele signaal van de knop opslaan. Dit zorgt voor

een stabilisering van het signaal. Het uitlezen van dit signaal gebeurt met een klokfrequentie van  $7.5Hz$ . Alle knoppen zijn drukknoppen. Een knop moet dus minstens 0.13 seconden in worden gedrukt om er zeker van te zijn dat deze juist wordt uitgelezen. Bij het ingedrukt houden van de richtingsknoppen of de start/pauze knop (langer dan 0.26 seconden) zal er een signaal worden geproduceerd die een frequentie van  $3.75Hz$  heeft. Je kunt niet sneller gaan door de drukknop vaker of sneller in te drukken.

#### 4.2.2 State bepalen

De controller bepaalt aan de hand van de ingangssignalen wat het gedrag van de chip is. De states zijn: het instellen van een beginsituatie, de simulatie uitvoeren, de simulatie pauzeren en het systeem resetten waardoor er weer opnieuw begonnen kan worden. Voor een correcte implementatie hiervan is een state diagram gemaakt, zie figuur 4.3).



**Figuur 4.3:** Het state diagram van de systeem-controller

In het state diagram zijn de volgende states te herkennen:

- |           |         |         |
|-----------|---------|---------|
| • Positie | • Left  | • Spel  |
| • Up      | • Right | • Pauze |
| • Down    | • Set   | • Reset |

Positie bepaalt tijdens het instellen van een beginsituatie waar de cursor zich bevindt. De states: up, down, left en right zorgen ervoor dat de cursor zich in de richting van die state beweegt. Bij left en right is binnen de state een if statement geplaatst. Bij right zorgt dit statement ervoor dat de cursor zich naar rechts blijft bewegen wanneer de cursor zich aan de rechterrاند van het raster bevindt. Zonder het if-statement zou de cursor aan het eind van de regel een regel omlaag springen. Bij left is de werking van het if-statement gelijkwaardig, behalve dat deze controleert of de cursor zich aan de linkerrand van het raster bevindt. De state set zorgt ervoor dat de cel, waarop de cursor zich bevindt, levend gemaakt wordt. Wanneer er twee knoppen tegelijkertijd ingedrukt worden, dan zullen beide knoppen genegeerd worden.

In de state spel zal de ingestelde beginsituatie gesimuleerd worden. Tijdens deze simulatie kan de gebruiker op de pauzeknop drukken. Wanneer dit gebeurt zal naar de pauzestate toegegaan worden. Hierin zal het spel gepauzeerd worden en de huidige state op de monitor afgebeeld blijven staan. Als nu op de start knop gedrukt wordt zal de simulatie vervolgen. Te allen tijde kan het spel gereset worden door op de reset knop te drukken. Als deze ingedrukt wordt zal naar de reset state toegegaan worden. Hierna kan de gebruiker een nieuwe beginsituatie instellen.

## 4.3 VHDL simulatie

Om de werking van de systeem-controller te testen is er een VHDL simulatie gemaakt. Voor de implementatie van de systeem-controller is er gebruik gemaakt van het state-diagram uit figuur 4.3. Daarom is het bij de simulatie belangrijk om te testen of alle states op de juiste manier geïmplementeerd zijn. Dit betekent dat iedere state op het juiste moment gebruikt wordt en dat de uitgangssignalen in die state correct zijn. Aan de hand van een pixelpointer van zes bits wordt bijgehouden op welke cel de cursor zich bevindt. Het is belangrijk om te testen of het gedrag van deze pointer voldoet. Ten slotte is het belangrijk om te controleren of de bepaalde prioriteit van de knoppen klopt. Bij de simulatie is er een 4 bitssignaal richting. Ieder bit van dit signaal geeft een bepaalde richting aan. De volgende waarden en richtingen horen bij elkaar:

- 0001 : up
- 0010 : right
- 0100 : down
- 1000 : left

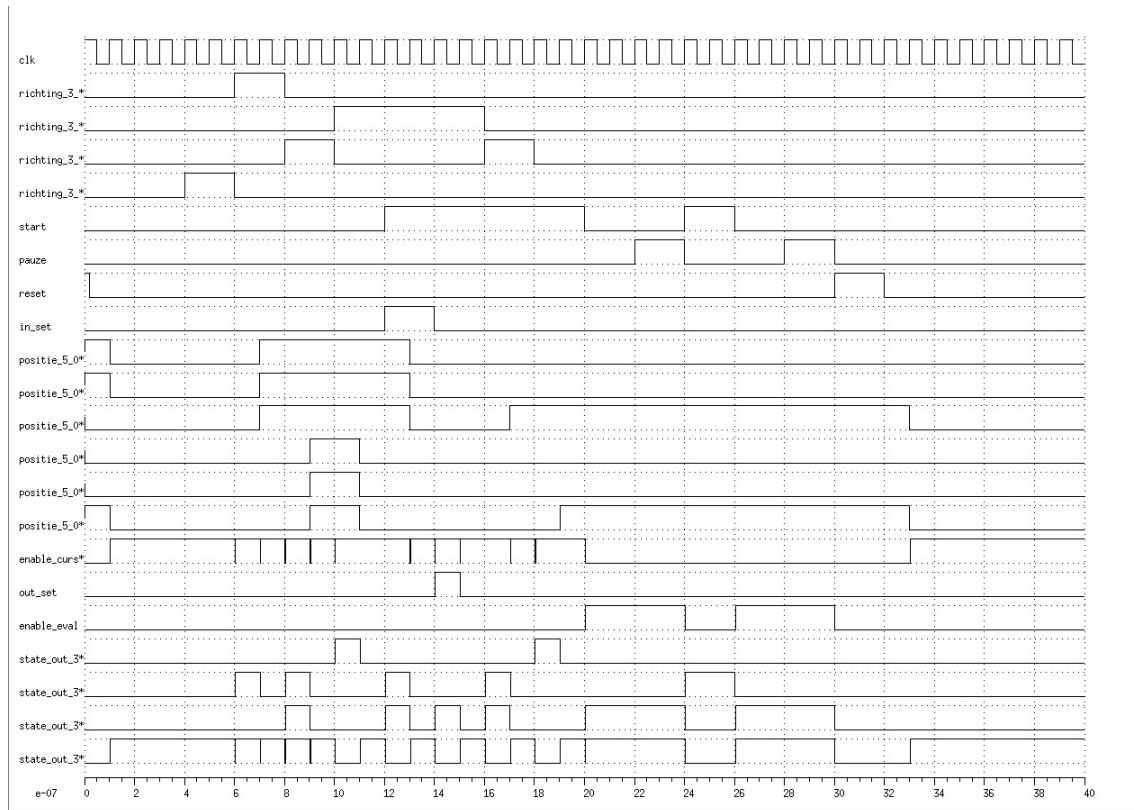
De simulatie is opgedeeld in twee delen. In het eerste deel (zie figuur: 4.4) wordt getest of de richtingsknoppen correct verwerkt worden. Tevens wordt gesimuleerd of de prioriteiten juist zijn. Het tweede deel van de simulatie (zie figuur: 4.5) wordt getest of het simuleren en pauzeren van Game Of Life werkt zoals ontworpen.

## 4.4 Synthese

Nadat de simulaties een juiste werking toonden is er een layout gemaakt. De uiteindelijke layout heeft de volgende eigenschappen:

[illegible][illegible][illegible]





**Figuur 4.6:** Switch level simulatie van systeem-controller

daarop aangesloten worden, anders zal het testen van het systeem alleen maar onnodig moeilijker worden.

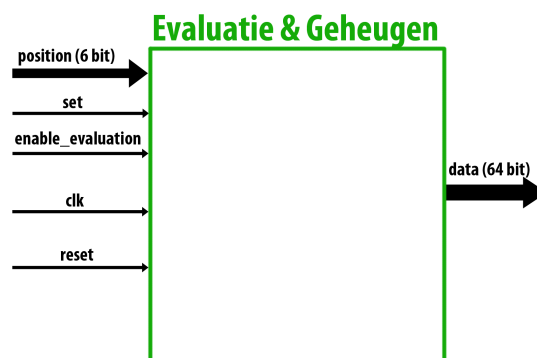
## Hoofdstuk 5

# Evaluatie & Geheugen

### 5.1 Specificaties

Het subonderdeel Evaluatie & Geheugen heeft vijf verschillende ingangssignalen: *position*, *set*, *enable\_evaluation*, *clk* en *reset*. Het *reset* signaal zal alle geheugenelementen in de schakeling resetten. *clk* is het kloksignaal waarop de geheugenelementen schakelen. De oplettende lezer zal hebben opgemerkt dat in het systeemoverzicht de uitgang van de klokdeler naar dit deelsysteem verschillende waarden aan kan nemen. De reden hiervoor zal in het volgende paragrafen verder onderbouwd worden.

Het signaal *enable\_evaluation* kan de schakeling in de staat zetten waarin bij elke klokslag het veld doorgerekend wordt. Wanneer dit signaal echter laag is, zullen de flip-flops na elke klokpuls hun waarde behouden. De zes-bits vector *position* is een verwijzing naar een enkele pixel in het geheugenblok. Door met *position* een pixel aan te wijzen kan vervolgens met een hoog signaal op *set* de pixel levend gemaakt worden. De uitgang van het systeem is de 64-bits vector *data*. Elke bit in deze vector geeft de staat van een individuele cel aan: levend of dood. Zie figuur 5.1 voor het overzicht van de hierboven beschreven signalen.



Figuur 5.1: Overzicht van ingangs- en uitgangssignalen

### 5.2 Implementatie

Het deelsysteem Evaluatie & Geheugen heeft twee functies: het bijhouden van de huidige waarde van de cellen en het berekenen van de volgende staat van het spel. Er zijn twee manieren bedacht om dit aan te pakken. De eerste manier is het scheiden van de logica en het geheugenblok. Dit

houdt in dat er op de chip één combinatorisch circuit bevindt, dat door alle cellen gebruikt wordt om de volgende staat te berekenen. Er zal dan een controller ontworpen moeten worden die alle cellen sequentieel afgaat en hun volgende staat bepaalt. Het probleem van deze aanpak is dat voor het berekenen van de volgende staat voor een cel, de vorige staat van de acht burens van deze cel nodig is. En wanneer de staat van de cellen sequentieel wordt aangepast, is de vorige staat niet van alle cellen niet beschikbaar. Om dit op te lossen, is het mogelijk gebruik te maken van twee geheugenblokken, één voor het opslaan van de volgende staat voor een cel en één voor het opslaan van de vorige staat van een cel. Het nadeel is dat hiervoor twee flip-flops per cel nodig zijn en dat neemt in verhouding tot de andere modules erg veel ruimte in op de chip.

Een tweede manier is een systeem waarin elke cel zijn eigen stuk combinatorische logica heeft om de volgende staat te berekenen. Op deze manier is er geen controller nodig die sequentieel alle cellen af gaat, de berekeningen vinden immers op hetzelfde moment plaats. Nog een voordeel van deze methode is dat er per cel maar één flip-flop nodig is. Een groot nadeel is wel dat er per cel meer logica nodig is. Dit neemt weer een gedeelte van de kostbare ruimte op de chip in.

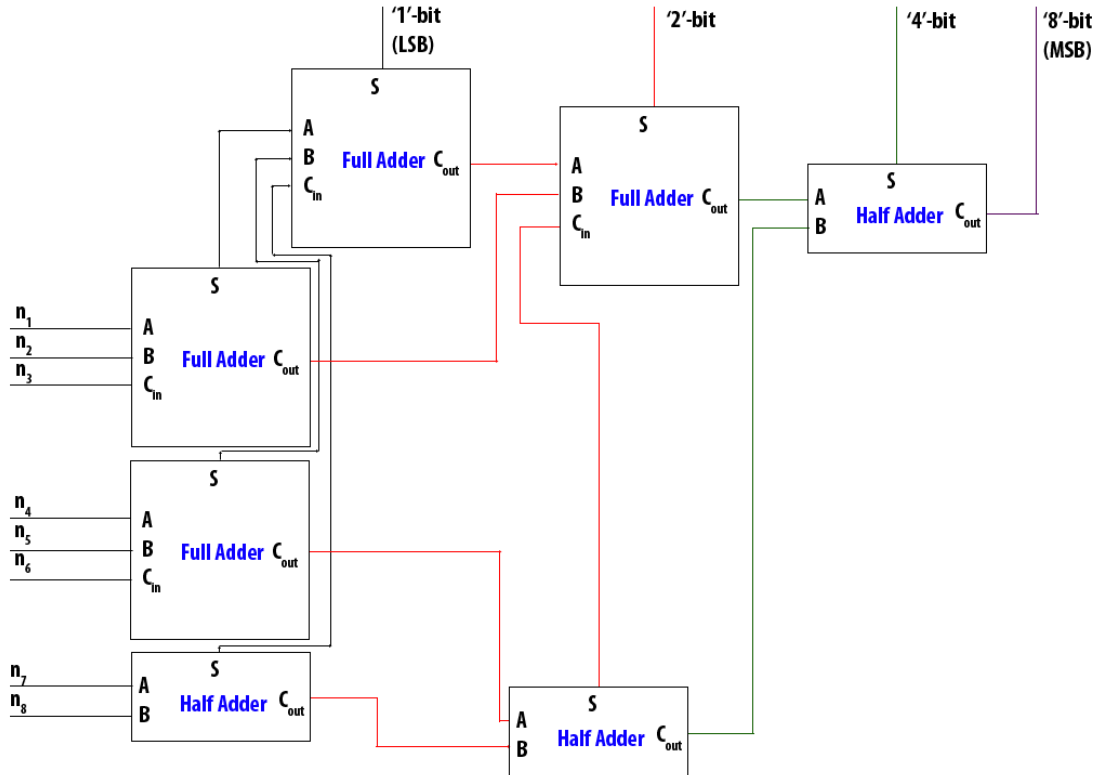
Uiteindelijk is er voor de tweede aanpak gekozen, omdat dit naar schatting minder ruimte op de chip in zal nemen dan de eerste aanpak. Daarnaast is het tweede systeem veel robuuster, omdat het voor het bereiken van een specifieke simulatiesnelheid een lagere klokfrequentie gebruikt kan worden dan in een sequentieel systeem. Een sequentieel systeem heeft namelijk vele klok-pulsen nodig voor het berekenen van een nieuwe iteratie, waar de parallele oplossing slechts één klok-puls nodig heeft. De lagere klok-frequentie stelt lagere eisen aan de delay van de logica en vermindert de kans op fouten. Tevens heeft een parallel systeem bij identieke logica een hogere maximale simulatie-frequentie waarbij het systeem nog functioneert.

De verwerkings-eenheden van de cellen bepalen de volgende staat van een cel door de regels van Game of Life te implementeren. De staat van een cel hangt af van het aantal levende burens van deze cel. De eerste taak van de logica is dan ook het berekenen van het aantal burens dat in het vorige tijdframe in leven was. Hiervoor krijgt de verwerkings-eenheid van een cel de staat van ieder van zijn burens binnen als een signaal. Uit deze acht signalen moet het aantal levende burens bepaald worden. Dit komt overeen met het berekenen van de hamming weight [4]: het aantal bits dat ongelijk aan nul is. Een standaard methode om de hamming weight te berekenen bestaat uit het aaneenschakelen van full-adders in verschillende lagen. De signalen worden eerst zelf bij elkaar opgeteld. Als deze optellingen carry-bits opleveren, worden deze verder verwerkt als signalen met waarde twee. Deze signalen worden weer verder opgeteld, waarbij een eventuele carry-bit waarde vier heeft, enzovoorts. Een dergelijk systeem is te zien in figuur 5.2.

Dit systeem kan nog geoptimaliseerd worden, aangezien een cel al sterft aan overbevolking zodra er meer dan vier burens in leven zijn. Het gedrag is hetzelfde voor vier, vijf, zes, zeven of acht levende burens, dus hoeft er niet verder geteld te worden dan een hamming weight van vier. Het gevolg is dat de full adder van de most significant bit vervangen kan worden door een OR-port. Daarnaast worden er poorten toegevoegd die de regels van de Game of Life implementeren aan de hand van het aantal levende burens. Het geheel aan logica per cel is te zien in figuur 5.3.

Tot slot worden er nog een stukje aansturingslogica en een flip-flop voor het onthouden van de staat van de cel aan het circuit toegevoegd. De aansturingslogica is een 2-1 multiplexer, die afhankelijk van het ‘enable’ signaal uit de controller (zie hoofdstuk over de systeemcontroller) de uitgangswaarde van het combinatorische circuit of de oude waarde uit de flip-flop kiest. Op deze manier kan er met behulp van het enable-signaal voor gekozen worden om het spel te pauzeren, zonder dat de klok van de flip-flop stil gezet hoeft te worden. Dit is nodig omdat anders de ‘set’ functionaliteit van de flip-flop niet werkt.

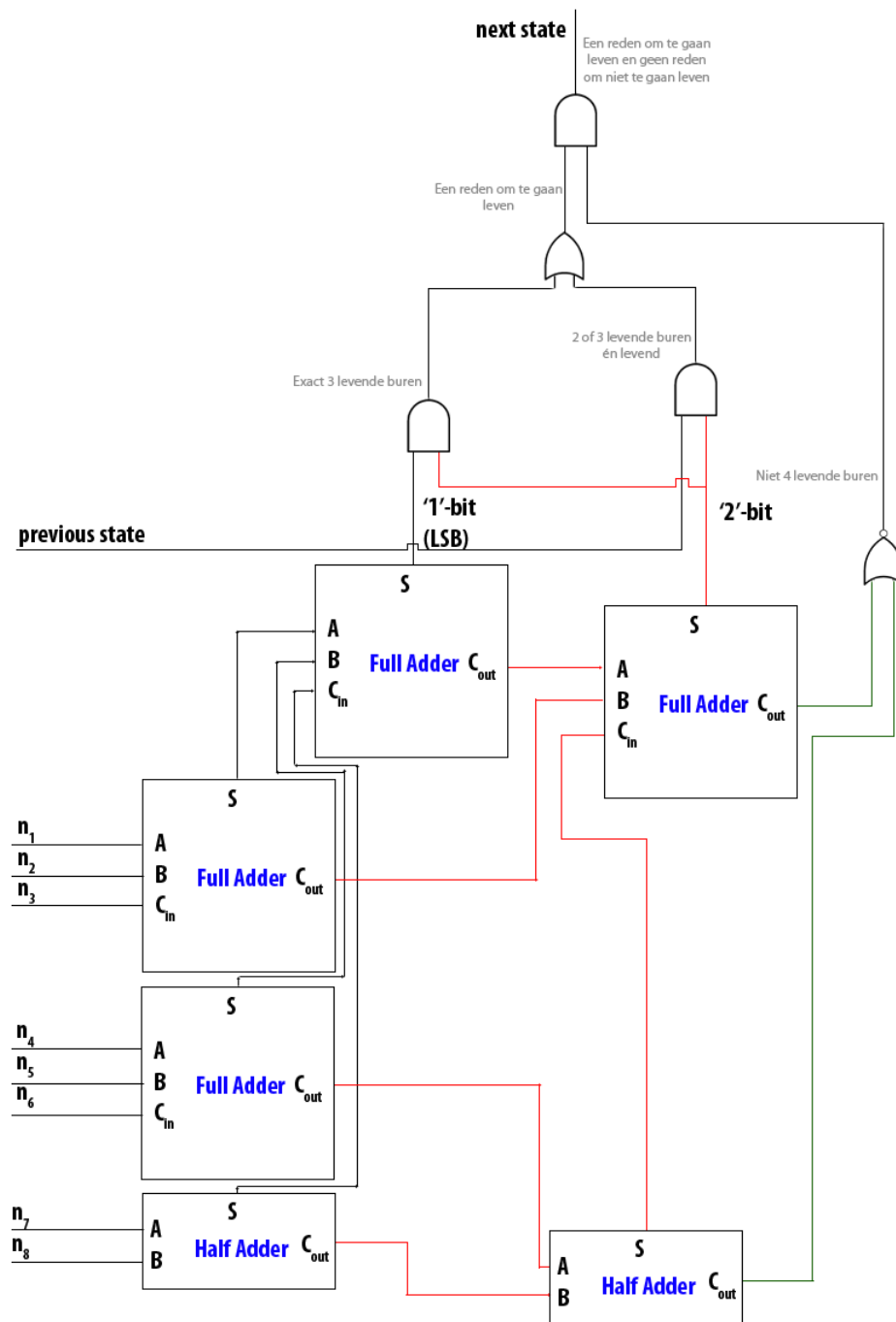
Door de modulaire opbouw van de cellen, kan een speelveld opgebouwd worden zo groot als de ruimte op chip dat toelaat. Er is gekozen voor een resolutie van 8x8 cellen. Dit omdat van dit formaat verwacht wordt dat het zeker op de chip past. De cellen worden op de chip in hetzelfde raster als ze zich in de simulatie bevinden. Omdat elke cel vooral verbindingen heeft met zijn burens, levert dit de kortste draden op. Om alle cellen aan te kunnen sturen zijn er nog nog twee



**Figuur 5.2:** Een standaard implementatie voor het berekenen van een hammingweight van 8 signalen

elementen naast de matrix van cellen toegevoegd: een klokdelers en een 6-64 bits demultiplexer. De klokdelers bepaalt op welke frequenties de flipflops in de cellen moeten werken. De gekozen frequentie is afhankelijk van twee signalen: het *enable* signaal afkomstig van desysteem- controller en het *switch* signaal, afkomstig van een switch van buiten de chip. Wanneer het enable signaal hoog is, zal het circuit in de simulatie modus staan. Dit houdt in dat het de simulatie van Game of Life gestart wordt met een frequentie van 0.94 of 1.88 Hz, afhankelijk van *switch*. Als *enable* hoog is, zal de klokdelers een frequentie van 7.5 Hz uitsturen naar de flip-flops. Dit is dezelfde frequentie waarop de controller werkt. Wanneer de controller dus een *set* signaal naar één van de cellen stuurt zal deze in theorie mooi op de klokflank aankomen, beide systemen werken immers op dezelfde klokfrequentie.

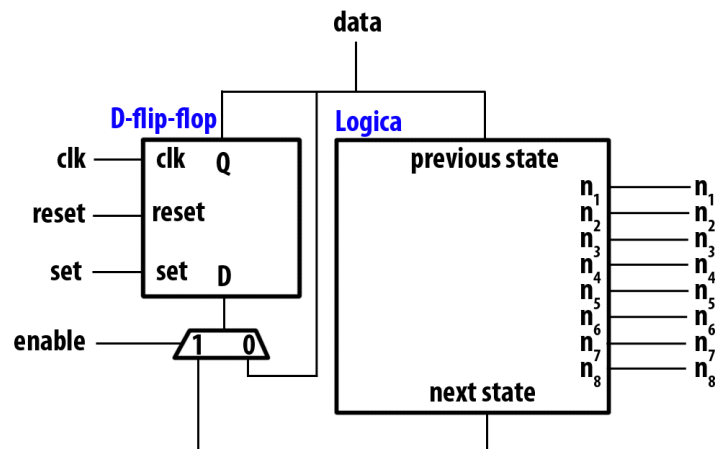
De 6-64 bits demultiplexer wordt gebruikt om het zes-bits *positie* signaal, afkomstig van de controller, naar de juiste flip-flop te leiden. Het moet namelijk mogelijk zijn om een individuele cel aan te zetten. Het uitgangssignaal van alle cellen wordt direct doorverbonden naar de VGA-timer. Er is hier geen gebruik gemaakt van een multiplexer vanwege het feit dat de VGA-timer een aantal pixels tegelijk uit wil kunnen lezen en op deze manier biedt de Evaluatie & Geheugen die mogelijkheid. Deze oplossing kost een aanzienlijke ruimte op de chip aan draden, maar door ervoor te zorgen dat de VGA-timer zich dicht in de buurt van het Evaluatie & Geheugen deelsysteem bevindt, nemen de draden zo min mogelijk ruimte in op de chip.



Figuur 5.3: De combinatorische logica per cel

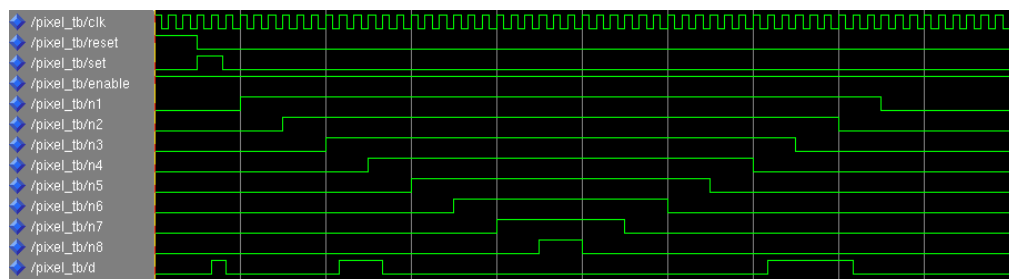
## 5.3 VHDL simulatie

De VHDL simulatie van het deelsysteem Evaluatie & Geheugen is opgedeeld in twee testbenches: één voor het testen van een individuele cel en één voor het testen van het gehele deelsysteem. De eerste simulatie is die van een enkele cel. De signalen die erin gestuurd worden zijn: *clk*, *set*,



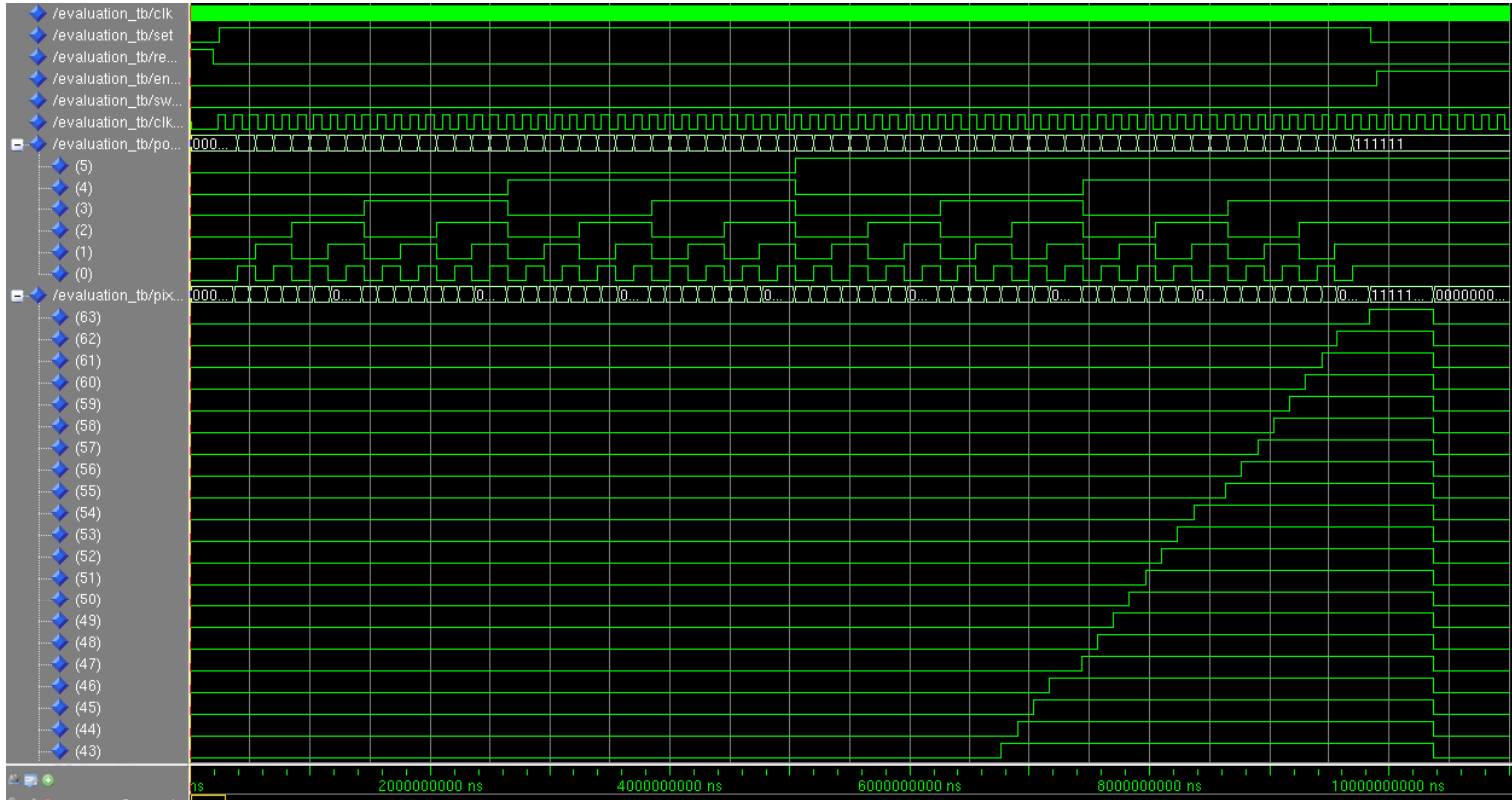
Figuur 5.4: De volledig schakeling voor één cel

*reset*, *enable* en de uitgangen van de acht burens ( $n_1$  t/m  $n_8$ ) die vertellen of deze burens leven of niet. Het uigangssignaal  $d$  geeft de staat van de cel aan, of deze levend of dood is. Zie figuur 5.5 voor het resultaat van de simulatie. De simulatie laat zien dat het blokje zich aan de basisregels van Game of Life houdt. Te zien is dat het blokje begint met leven als hij drie burens heeft. In het geval van twee levende burens, behoudt de cel dezelfde staat. Tot slot is zichtbaar dat in de andere gevallen het blokje geen teken van leven laat zien.



Figuur 5.5: De testbench voor een individueel blokje

Het tweede gedeelte van de simulatie omvat het testen van het gehele deelsysteem Evaluatie & Geheugen. Zie hiervoor figuur 5.6. De simulatie is zo opgezet dat eerst alle blokjes één voor één aan gezet worden op de manier zoals de Systeem-controller verwacht wordt dat ook te doen. Daarna zal het signaal *enable* op hoog gezet worden om het evalueren te beginnen. Als het goed is, levert het resultaat van de simulatie op dat alle blokjes in een enkele klokslag uit zullen gaan. In de figuur zijn niet alle 64 uitgangssignalen te zien. De overige bits van het signaal *data* gingen in de simulatie ook direct naar '0' toen *enable* op een hoog signaal gezet werd.



**Figuur 5.6:** De testbench voor het deelsysteem Geheugen & Evaluatie

## 5.4 Synthese & Layout

Bij het deelsysteem Evaluatie & Geheugen is veel gebruik gemaakt van structurele VHDL beschrijvingen. Hierdoor kan er voor elk klein onderdeel een aparte synthese en layout worden gemaakt. Dit zorgt voor een overzichtelijke layout waarbij een hoge efficiëntie kan worden gerealiseerd. Wel bevat het deelsysteem Evaluatie & Geheugen een groot aantal draden tussen de cellen en de multiplexer die gebruikt wordt om de pixels te ‘setten’. Deze draden zorgen er voor dat de Efficiëntie van het gehele deelsysteem niet bijzonder hoog is. Een bijzonder aspect van de layout van het deelsysteem is de structuur van de cellen. Omdat de cellen in de simulatie van Conway’s Game of Life vooral communiceren met de omliggende cellen, is het efficient om de cellen op de chip te plaatsen overeenkomstig met het raster zoals het op het scherm wordt weergegeven. Deze structuur is ook terug te zien in de layout van de gehele chip, te vinden in figuur 9.1 verderop in dit verslag.

Aantal transistors	Efficiëntie
39.039	58.6%

## 5.5 Switch-level simulatie

De switch-level simulatie van de submodule Evaluatie & Geheugen laat zien of het systeem zich gaat gedragen zoals verwacht. Wanneer figuur 5.6 en figuur 5.8 vergeleken worden met behulp van de ‘compare’ functionaliteit in GoWithTheFlow blijkt hieruit dat beide simulaties gelijk zijn aan elkaar. Hetzelfde geldt voor de figuren 5.5 en 5.7.





## 5.6 Conclusie

Voor het subsysteem Evaluatie & Evaluatie is er de keuze tussen twee oplossingen. De eerste is het geheugen in twee delen splitsen en er vervolgens sequentieel doorheen gaan om de staat van de volgende tijdstap door te rekenen. De andere oplossing is iedere cel een eigen stuk logica mee te geven om zijn volgende staat te bepalen. Uiteindelijk hebben we voor de tweede aanpak gekozen.

Het voordeel van deze tweede aanpak, is dat bij het synthetiseren en layouten van de chip per cel gedaan kan worden. Omdat elk blokje alleen met zijn burens verbonden hoeft te worden, kan er door de blokjes in de goede volgorde op de chip te plaatsen, behoorlijk wat ruimte worden bespaard. Dit heeft tot gevolg dat het onderdeel waarin alleen de blokjes zich bevinden zeer geoptimaliseerd kan worden.

De simulaties van het Evaluatie & Geheugen gedeelte verliepen goed. In het begin bleek het lastig om de code die voor de juiste aaneenschakeling van de verschillende blokjes zorgde goed te krijgen. Dit komt omdat er voor elke individueel blokje andere burens beschikbaar zijn. Uiteindelijk is dit prima gelukt. Vervolgens liep de switch-level simulatie ook goed.

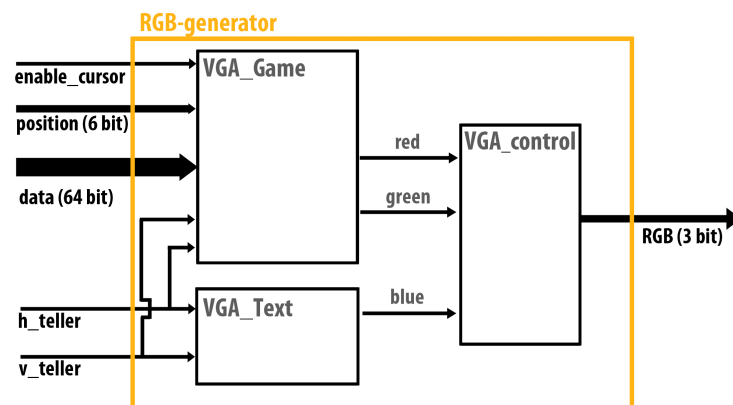
## Hoofdstuk 6

# RGB-generator

### 6.1 Specificaties

Het deelsysteem RGB-generator moet de volgende functies uitvoeren:

- Het scherm opsplitsen in actieve en non-actieve gebieden
- De actieve regio opsplitsen in een speelveld met 8x8 cellen
- Rondom de cellen een kader plaatsen
- Het juiste RGB-sigitaal doorsturen voor iedere cel
- In het non-actieve gebied de tekst 'Game of Life' laten zien



Figuur 6.1: Het volledige deelsysteem RGB-generator en zijn interne opdeling

### 6.2 Implementatie

Er is gekozen voor een implementatie met 3 entities. Er is een top-entity (VGA\_control) die bepaalt welke RGB-waarde er gekozen moet worden, er is een tweede entity (VGA\_game) voor het weergeven van het spel zelf (actieve gebied) en een derde entity (VGA\_text) voor het maken van de tekst (non-actieve gebied). Hoe deze entities aan elkaar gekoppeld zijn, is te vinden in figuur 6.1.

In de top-entity wordt gekeken in welk deel van het scherm de simulatie zich bevindt. Omdat het scherm rechthoekig en ons speelveld vierkant, betekent dit niet dat niet het hele scherm zal worden benut. Het deel van het veld dat niet als speelveld gebruikt wordt, wordt gebruikt om de letter weer te geven. De top-entity bepaalt of de VGA-timer zich in het actieve of in het non-actieve gebied bevindt. Er wordt daarvoor gekeken naar de signalen *h.teller* en *v.teller* die afkomstig zijn van de VGA-timer. Als het spel zich in het actieve gebied bevindt, wordt de RGB-waarde gekozen die geproduceerd wordt door de entity *VGA\_game*. Als het spel zich in het non-actieve gebied bevindt, wordt de RGB-waarde gekozen die geproduceerd wordt door de entity *VGA\_text*.

Als het systeem zich binnen het actieve gebied bevindt, wordt het spel weergegeven op een speelveld van 8x8 cellen. Iedere positie wordt opgesplitst in een x- en y-component. Zo is het eenvoudiger om de positie van de actieve cel te bepalen en te bepalen of hij actief moet worden of niet.




Om het juiste RGB-sigitaal naar de VGA-timer te sturen moet er voor iedere cel gekeken worden naar de positie van de tellers en of die cel levend of dood moet zijn. Voor het eerste RGB-bit wordt uit de combinatie van de x- en y-component voor elk van de 64-bits bepaald of er een 0 of een 1 op de ingang staat. Vervolgens wordt het juiste signaal op de uitgang gezet. Voor het tweede RGB-bit wordt er gekeken of de cursor aan is en of deze gelijk is aan de actieve cel. Als dit het geval is, wordt op de uitgang een hoog signaal gezet, anders een laag signaal. Het derde RGB-bit gebruiken wij in dit veld niet en is daarom continu verbonden met de aarde. Er is aan het gehele speelveld ook een offset, die gebruik maakt van constanten, toegevoegd zodat het speelveld gecentreerd kan worden.

Het implementeren van de tekst gebeurt in een andere entity. Allereerst is er in Excel een template gemaakt hoe de letters eruit moeten komen te zien. Daarna is er voor iedere regel van de tekst bepaald of er een 0 of een 1 doorgegeven moet worden aan het derde RGB-bit. Dit resulteerde in 23 lijnen van elk 160 tekens lang (de breedte van het scherm) die op deze manier de letters vormen. Ook werd er een offset aan toegevoegd die ervoor zorgt dat de tekst in zijn geheel naar boven of beneden verschoven kan worden.

## 6.3 VHDL simulatie

Om het testen van dit subonderdeel zijn er verschillende VHDL simulaties gemaakt. De werking van iedere entity is afzonderlijk gesimuleerd om te kijken of de onderdelen werken zoals deze ontworpen zijn.

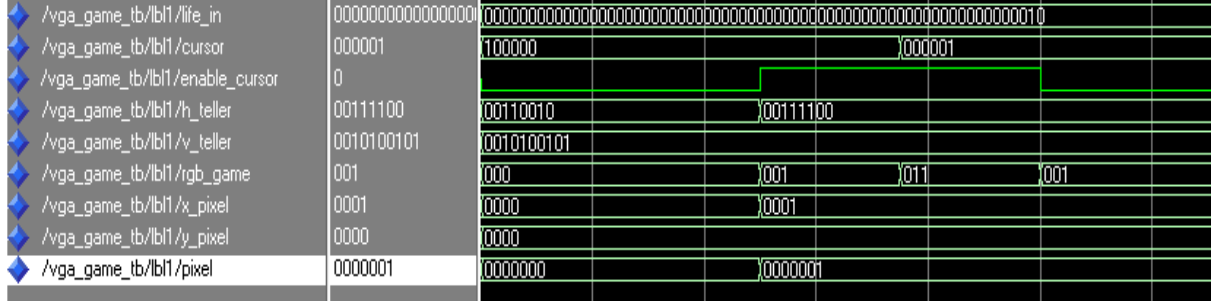
In figuur 6.2 is te zien dat de *VGA\_control* schakelt tussen het signaal van *VGA\_text* (000) en het signaal van *VGA\_game* (111). Wanneer er welk signaal wordt weergegeven, hangt af van de plaats in het speelveld waarop het spel zich op dat moment bevindt.

 /vga_control_tb/lb1/h_teller	00000000	01010000		00000000
 /vga_control_tb/lb1/v_teller	0000000000	0011111010		0000000000
 /vga_control_tb/lb1/rgb_out	111	000		111

**Figuur 6.2:** Simulatie van de entity *VGA\_control*

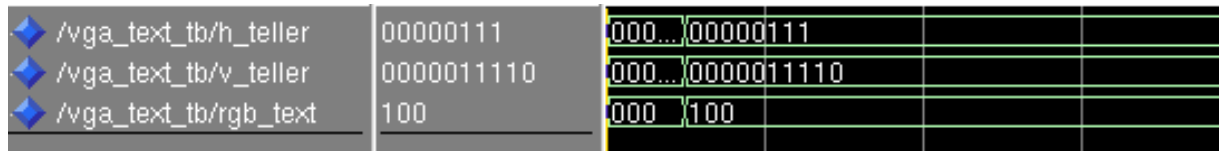
In figuur 6.3 worden een aantal verschillende signalen weergegeven. Aan het signaal *life\_in* is te zien dat alleen het tweede bit op dit moment actief is. Eerst staan *h.teller* en *v.teller* op positie 1. Aangezien dit bit niet actief is, zijn alle uitgangssignalen 0 en staan ook *x.pixel* en *y.pixel* op positie 0. Daarna veranderen *h.teller* en *v.teller* zodat ze op positie 2 staan. Daardoor verandert *x.pixel* van 0000 naar 0001 (naar positie 2) en verandert ook *pixel* naar positie 2. Dat vertaalt zich ook in een verandering van *rgb\_game* naar 001. Tot slot zijn er ook nog de signalen *cursor* en

*enable\_cursor*. De uitgang verandert pas als de cursor op de actieve positie staat en *enable\_cursor* hoog is zoals te zien is op de simulatie.



**Figuur 6.3:** Simulatie van de entity VGA\_game

Op figuur 6.4 is te zien dat als *v\_teller* en *h\_teller* op een letter staan de uitgang *rgb\_text* verandert naar 100. In alle andere gevallen is deze uitgang 000.



**Figuur 6.4:** Simulatie van de entity VGA\_text

## 6.4 Synthese & Layout

Aantal transistors	Efficiëntie
10 836	27.52%

Zoals te zien neemt dit onderdeel op de chip redelijk veel ruimte in beslag. Ook is het niet mogelijk om een hele hoge efficiëntie te behalen vanwege de vele verbindingen die nodig zijn voor het laten weergeven van de tekst.

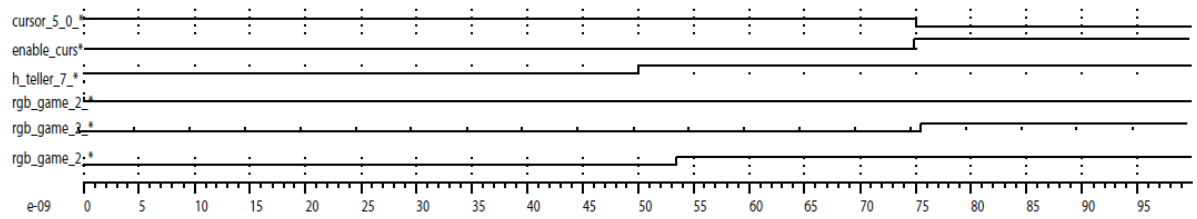
## 6.5 Switch-level simulatie

Er is een switch-level simulatie gedaan van het onderdeel VGA\_game (zie figuur 6.5). Daarbij is te zien dat er enkele kleine tijdsverschillen zijn tussen de testbench en de uiteindelijke switch-level simulatie. Deze verschillen zijn dermate klein dat ze geen problemen zullen geven op de uiteindelijke chip.

Van de andere entiteiten zijn ook switch-level simulaties gemaakt, maar deze zijn heel klein en hebben maar weinig variërende signalen. Op deze simulaties zijn geen relevante verschillen ten opzichte van de VHDL-simulaties aan te merken.

## 6.6 Conclusie

Voor de implementatie van de RGB-generator zijn een aantal mogelijkheden overlopen. Uiteindelijk hebben we voor de huidige implementatie gekozen waarbij we de *h\_teller* en *v\_teller* van de VGA-timer gebruiken om de signalen op het juiste tijdstip door te sturen. Als laatste is ook de



**Figuur 6.5:** Switch-level simulatie van de entity `VGA_game`

titel toegevoegd waardoor het onderdeel nog opgesplitst is in 3 entities; 1 topeentity die schakelt tussen 2 entities die elk een RGB-signaal uitsturen.

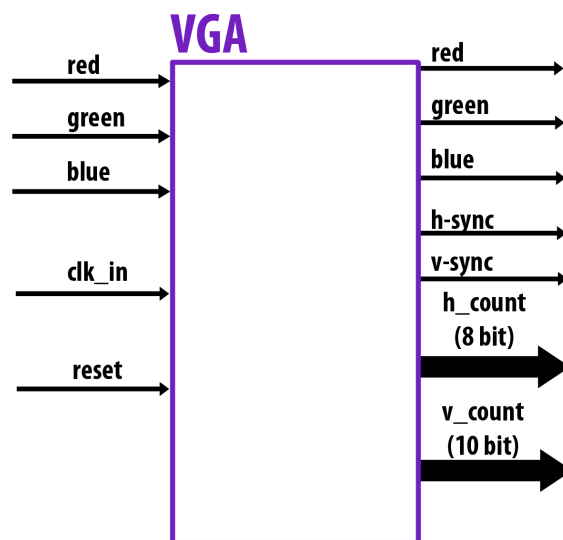
## Hoofdstuk 7

# VGA-timer

### 7.1 Specificaties

Het deelblok VGA-timer moet de volgende functies uitvoeren afhankelijk van de ingangsklok van  $6.144MHz$ :

- Op de juiste tijd een RGB-sigitaal, h-sync en v-sync doorsturen naar een VGA kabel
- De positie in het scherm doorgeven aan de RGB-generator door h\_count en v\_count
- Een  $60Hz$  signaal sturen naar de Klok voor het verloop van het spel door de v-sync



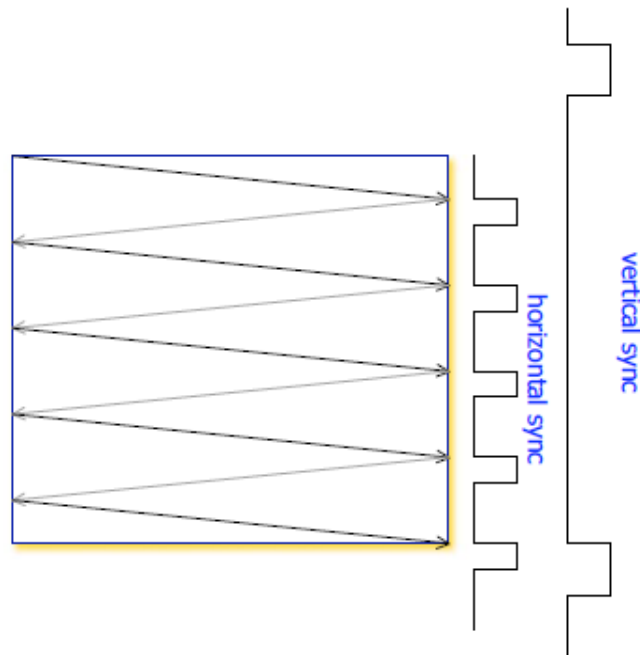
**Figuur 7.1:** Overzicht van ingangs- en uitgangssignalen

De uitgangssignalen van figuur 7.1 kunnen als volgt verklaard worden. Het spel wordt getoond op een VGA-monitor met behulp van de vijf output signalen: de beeldsignalen *red*, *green*, *blue* en de synchronisatietijden *h - sync* en *v - sync*. Een standaard VGA-monitor laat  $640 \times 480$  pixels zien, werkt met een verversingsfrequentie van  $60Hz$  en een kristalfrequentie van  $25MHz$ . Maar aangezien het kristal van de chip werkt op een frequentie van  $6.144MHz$  betekent dit dat de

klok te laag is om alle pixels te besturen. Daar komt bij dat een VGA-monitor alleen werkt als alle pixels afgegaan worden. Daarom doorlopen we de pixels horizontaal in blokken van meerdere pixels in plaats van één voor één. Aangezien de kristalaansturingsfrequentie ongeveer vier keer zo langzaam is als de frequentie waarop een standaard VGA-monitor werkt, gaan we slechts  $\frac{640}{4} = 160$  pixels aansturen. Dit gaan we op de monitor terug zien als we bijvoorbeeld een pixel aanzetten; het lijkt alsof er 4 pixels op een rij aanstaan. Of er een pixel aan of uit is, wordt bepaald met behulp van de ingangsignalen red, green, blue die gegenereerd worden door de RGB-generator. Of de pixel op het beeld te zien is, is afhankelijk van de waarde van  $h - sync$  en  $v - sync$ . Waar deze pixel te zien is, is weer afhankelijk van de waarden van  $h\_count$  en  $v\_count$  die doorgegeven worden aan de RGB-generator. De RGB-generator zal vervolgens op de juiste moment, het juiste RGB-signaal doorgeven. De reden waarom deze twee tellers 8 en 10 bits lang zijn, is te vinden in Implementatie.

## 7.2 Implementatie

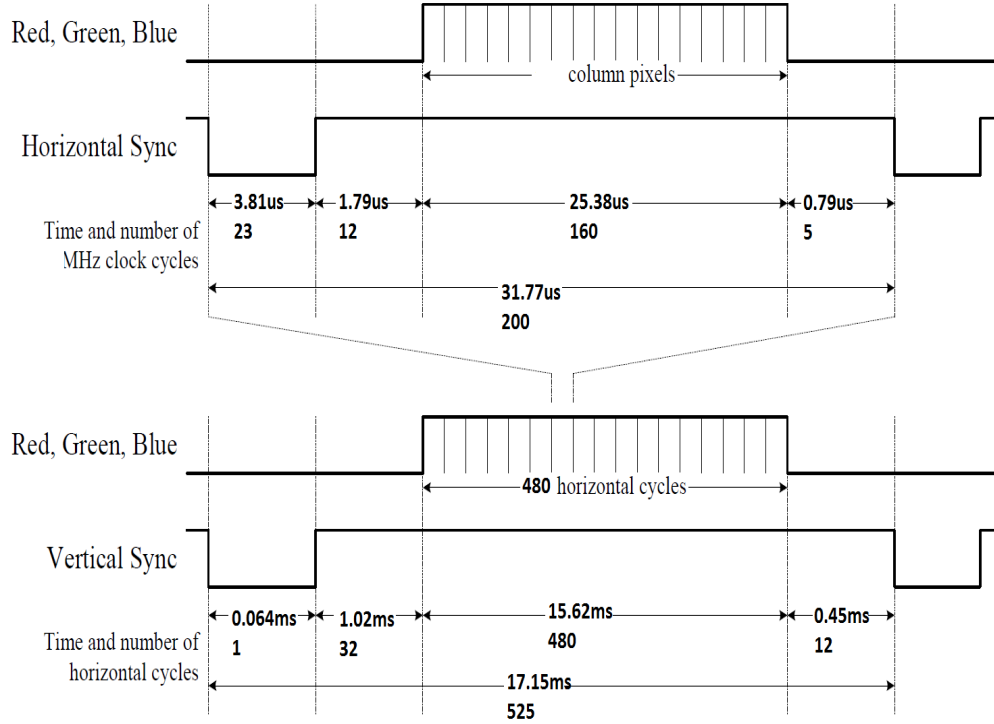
De manier van pixels doorlopen gaat als volgt; je begint linksboven het scherm en eindigt rechts-onder het scherm (zie figuur 7.2). Hierbij houdt  $h - sync$  de rijen bij en gaat  $v - sync$  aan het einde van een rij één keer omlaag om aan te geven dat er een volledige rij is doorlopen. Dit hele proces gebeurt zestig keer per seconde ( $60Hz$ ). Aangezien het menselijk oog op deze frequentie het verversen van het scherm niet meer waar kan nemen zal deze frequentie voldoen.



**Figuur 7.2:** H-sync wordt laag bij het eind van een rij

Zoals al gezegd is, zijn de  $h - sync$  en de  $v - sync$  afhankelijk van de  $h\_count$  en  $v\_count$ . In deze tellers zit namelijk de synchronisatietijden. Bij het verwerken van deze tijden moet vermeld worden dat de  $h\_count$  verhoogd wordt bij elke opgaande klokflank en de  $v\_count$  wordt verhoogd als de  $h\_count$  de waarde 199 bereikt en tevens op een opgaande klokflank (zie appendix VGA-timer). Verder is er te zeggen dat de klok  $6.144MHz$  is, wat neer komt op een periode van  $\frac{1}{6.144} \times 10^6 = 0.1627\mu s$ . Elke  $0.1627\mu s$  stijgt de waarde van  $h\_count$ . De VGA-monitor van  $640 \times 480$  met een

verversingsfrequentie van  $60Hz$  heeft een negatieve polariteit voor de  $h-sync$  en  $v-sync$  [5]. Dit betekent dat hij op een laag signaal naar de volgende rij of het volgende beeld gaat (zie figuur 7.2). De synchronisatietijden van een VGA-monitor van  $640 \times 480$  met verversingsfrequentie van  $60Hz$  hebben de volgende vaste waarden voor de  $h\_count$  en  $v\_count$ :



**Figuur 7.3:** In een v-sync zit 480 h-sync signalen

De periode van een  $v-sync$  duurt 525 waarden van  $v\_count$ . Hiermee kan de frequentie van  $v-sync$  te bepaald worden. Volgens [5] is deze  $\frac{1}{16.683ms} = 60Hz$ . Alle waarden en tijden worden verder uitgelegd aan de hand van figuur 7.3.

- De periode van  $h-sync$  begint met een laag signaal. Hier wordt het begin van de rij gezocht, zie figuur 7.2. Dit duurt 23  $h\_count$ .
- Voor het daadwerkelijk doorsturen van een RGB-sigitaal, wordt  $h-sync$  hoog gezet, zodat we na deze tijd kleuren op de monitor gaan zien. Dit duurt 12  $h\_counts$ .
- Aangezien er 160 pixels te besturen zijn in een rij, zijn er ook 160  $h\_counts$ . In deze tijd wordt er een RGB-sigitaal doorgelaten en is er een beeld te zien op de monitor.
- Daarna volgt er een periode waar er geen RGB-sigitaal meer wordt doorgestuurd, maar de  $h-sync$  nog wel hoog staat. Dan begint het proces opnieuw en gaat de  $h-sync$  weer omlaag om het begin van de rij te vinden. Dit duurt 5  $h\_counts$ .
- De  $h\_count$  heeft een totaalwaarde van 200. Om deze waarde te halen is er een teller nodig van minimaal 8 bits ( $2^8 = 256$ ).



- De periode van  $v - sync$  begint met een laag signaal. Hier wordt het begin van het beeld gezocht, zie figuur 7.2. Dit duurt slechts 1  $v\_count$ .
- Voor het daadwerkelijk doorsturen van een RGB-sigitaal wordt  $v\_sync$  hoog gezet, zodat na deze tijd kleuren op de monitor te zien zullen zijn. Dit duurt 32  $v\_counts$ .
- Aangezien er 480 rijen zijn in een beeld, telt de  $v\_count$  ook tot 480. In deze tijd wordt er een RGB-sigitaal doorgelaten en is er een beeld te zien op de monitor.
- Daarna volgt er een periode waarin er geen RGB-sigitaal meer wordt doorgestuurd, maar de  $v - sync$  staat nog wel hoge waarde heeft. Hierna begint het proces opnieuw en gaat de  $v - sync$  weer omlaag om het begin van het beeld te vinden. Dit duurt 12  $v\_counts$ .
- De  $v\_count$  heeft als totaalwaarde 525, om deze waarde te halen is een teller nodig van minimaal 10 bits,  $2^{10} = 1024$ .

Tijd( $\mu s$ )	3.81	1.95	25.38	0.82	32.54
$h\_count$	23	12	160	5	200

De periode van  $h\_sync$  duurt 200 waardes van  $h\_count$ , hiermee valt de frequentie van  $h\_sync$  te bepalen. Deze frequentie is  $31.5 kHz$  (zie: [5]), dit is tevens te controleren door  $\frac{1}{16.683 \mu s} = 31.5 kHz$ .

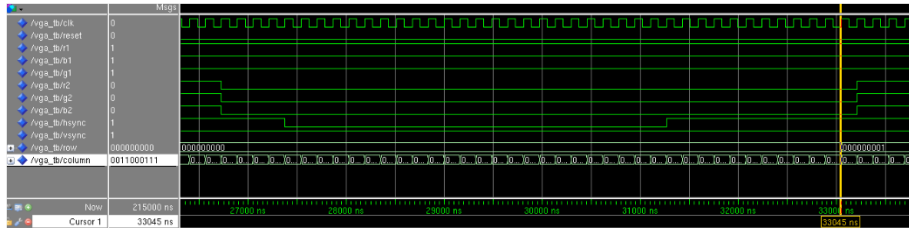
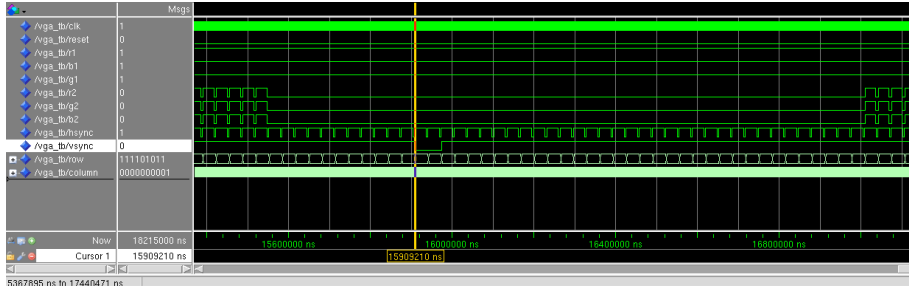
Tijd( $ms$ )	0.064	1.02	15.62	0.45	16.683
$v\_count$	1	32	480	12	525

Alleen in de tijd dat zowel de  $h - sync$  als de  $v - sync$  hoog zijn zal er daadwerkelijk een RGB-sigitaal worden doorgegeven. Wanneer gekeken wordt naar figuur 7.5 is te zien dat dit voornamelijk afhangt van  $v - sync$ . Bij het afstellen van de VGA-timing moest overwogen worden hoe strikt de synchronisatietijden [6] aangehouden zouden worden. Aangezien de periode van een  $h - sync$  standaard  $31.77 \mu s$  duurt kunnen er slechts 195 pixels aangestuurd worden. Een VGA-monitor is niet erg gevoelig voor verschillen in de synchronisatietijden en kan zelfs een foutmarge aan; kortere synchronisatietijden hebben weinig effect [6]. Aangezien 200 binnen de 5% procent marge ligt ten opzichte van de 195 zullen er geen problemen optreden.

## 7.3 VHDL simulatie

Het controleren van de VGA-timer wordt gedaan op basis van het bekijken van de  $h - sync$  en  $v - sync$ . Tevens wordt bekeken of het RGB-sigitaal op de goede manier aan en uit gaat; eerst de  $h - sync$  laag, daarna het RGB-sigitaal laag,  $h - sync$  hoog en als laatste het RGB-sigitaal hoog (zie figuur 7.4).

- Bij de 199<sup>ste</sup> (11000111) column gaat de row eentje omhoog, zo blijkt dat het eind van het lijn gehaald is en zal bij het row signaal 1 bit opgeteld moeten worden. Verder wordt er eerst geen RGB-sigitaal doorgegeven en daarna wordt er ook geen  $h - sync$  doorgegeven, dit is zoals verwacht (zie figuur: 7.4).
- Bij de 491<sup>ste</sup> (111101011)  $v\_count$  wordt het signaal  $v - sync$  laag, waarna er weer terug wordt gegaan naar de eerste rij (zie figuur: 7.5).

Figuur 7.4: Synchronisatietijden van de  $h - sync$ Figuur 7.5: Synchronisatietijden van de  $v - sync$ 

## 7.4 Synthese & Layout

Met behulp van GoWithTheFlow wordt de gemaakte VHDL-code gesimuleerd en geëxtraheerd. Hiermee kan er een layout gemaakt worden, die aangeeft hoeveel transistoren het geheel kost en de mate van efficiëntie. Voor de VGA-timer is dat als volgt:

Aantal transistors	Efficiëntie	Rijen	Afstand
146	52.9%	6	1
193	44.2 %	8	2

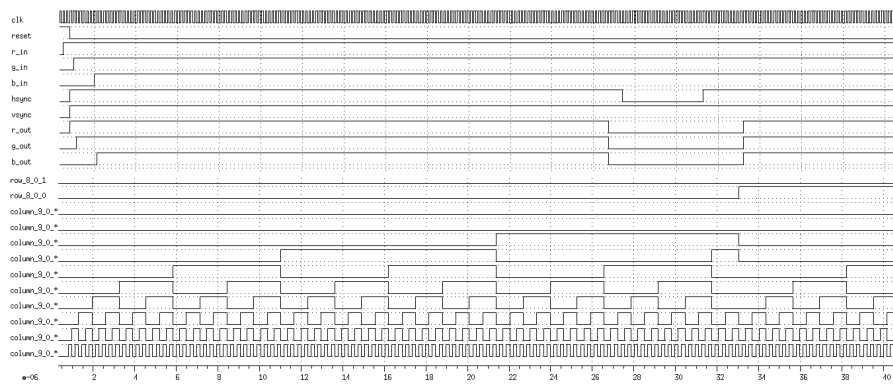
De layout is aangepast om genoeg ruimte te hebben voor de draden, er is gekozen voor 6 rijen met afstand 1. Dit is gedaan voor de grootst mogelijke efficiëntie en het minst mogelijke aantal transistors.

## 7.5 Switch-level simulatie

Uiteindelijk kan er met behulp van GoWithTheFlow ook een .ref file gemaakt worden die net zoals Modelsim een simulatie laat zien. Hiermee kan gecontroleerd worden of de VGA-timer ook daadwerkelijk werkt; bij lage  $h - sync$  en  $v - sync$  wordt er geen RGB-sigitaal doorgestuurd zie figuur 7.6.

## 7.6 Conclusie

Door de langzamere aansturingsfrequentie van het kristal is het aantal mogelijke horizontale pixels die te besturen zijn met een vierde afgenomen in vergelijking met een standaardfrequentie. Hiermee moet rekening gehouden worden om de synchronisatie van de VGA-monitor af te stellen. Opvallend is dat de uitgangssignalen  $h\_count$  en  $v\_count$  meer pixels tellen dan het beeld lang is. dit is om de synchronisatie te verwerken, deze tellers worden gebruikt in de RGB-generator om de correcte positie te bepalen.



**Figuur 7.6:** Oplopen van de row ook wel de h\_count

## Hoofdstuk 8

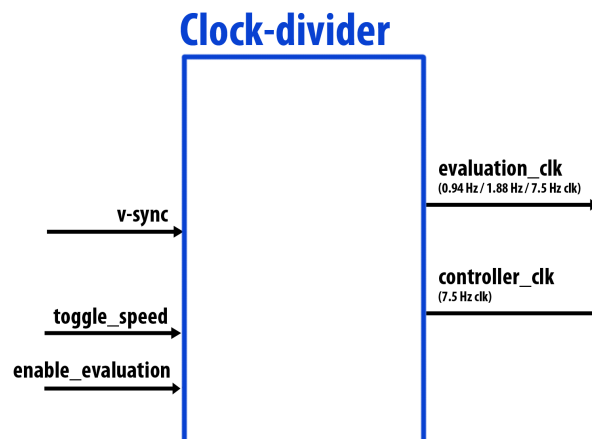
# Klok

Het gehele systeem maakt gebruik van hetzelfde kristal, maar de deelsystemen werken op verschillende frequenties. De lagere frequenties kunnen worden gegenereerd door klokdelers. Dit zijn in feite tellers, die bij een bepaalde waarde een signaal doorgeven. Deze tellers worden geïmplementeerd met flipflops en kosten dus een aanzienlijke hoeveelheid ruimte op de chip.

Om optimaal gebruik te maken van de beschikbare ruimte, worden de tellers die al in de VGA-timer geïmplementeerd zijn, ook gebruikt om een klok-sigitaal van circa  $60\text{ Hz}$  te genereren dat door de rest van de schakeling gebruikt kan worden. Dit signaal wordt vervolgens naar een centrale klokdeeler geleid dat alle andere klok-frequenties genereert. De controller ontvangt hier vandaan zijn signaal van ca  $7.5\text{Hz}$ , dat geschikt is om gebruikers-input te verwerken.

Ook de Evaluatie & Geheugen krijgt vanuit deze klokdeeler zijn kloksignaal. De frequentie hiervan is afhankelijk van de staat van de schakeling. Wanneer de initiële posities worden ingelezen, dan staat de *enable\_evaluation* laag, dan draait de Evaluatie & Geheugen op dezelfde  $7.5\text{Hz}$  (1) frequentie als de Systeem-controller. Tijdens het lopen van de simulatie, wanneer de *enable\_evaluation* hoog is, worden de cellen geëvalueerd op een frequentie van ca.  $0.94\text{Hz}$ (2) of ca.  $1.88\text{Hz}$ , afhankelijk van de ingestelde toggle speed; bij snel zal hij  $1.88\text{Hz}$ (3) zijn.

Onderdeel	Controller	Evaluatie(1)	Evaluatie(2)	Evaluatie(3)	RGB-gen	VGA-tim
Frequentie	$7.5\text{Hz}$	$7.5\text{Hz}$	$0.94\text{Hz}$	$1.88\text{Hz}$	geen	$6.144\text{MHz}$



Figuur 8.1: Een overzicht van het onderdeel clock-divider

---

In onderstaande tabel is ook te zien hoeveel transistoren de clock-divider nu uiteindelijk gebruikt.

Aantal transistors	Efficiëntie
684	71.05%

## Hoofdstuk 9

# Resultaat totaalontwerp

### 9.1 Inleiding

Het resultaat van het project is op verschillende manieren gecontroleerd. Door middel van simulaties en FPGA prototyping is geverifieerd dat de ontworpen schakeling naar behoren functioneert. Het uiteindelijke resultaat van het project is een layout waarmee een Sea-Of-Gates chip kan worden vervaardigd. De werking van deze layout is niet met volledige zekerheid vast te stellen, maar de uitkomsten van de eerdere controles maken een correcte werking zeer waarschijnlijk.

### 9.2 Simulatie

De VHDL-code van het gehele systeem is getest door een simulatie uit te voeren met behulp van een VHDL-testbench. In deze testbench worden achtereenvolgens een aantal cellen levend gemaakt en wordt daarna de simulatie gestart. De cellen zijn zo geplaatst dat er een oscillerend figuur ontstaat. De resultaten van de simulatie en een toelichten ervan zijn te vinden in appendix [A](#).

Met deze simulatie is het mogelijk een deel van de werking van het systeem te bevestigen, maar het is zeer lastig om het VGA-signaal goed te controlleren. Tevens kost het veel tijd om allerlei verschillende scenario's te simuleren met VHDL-testbenches.

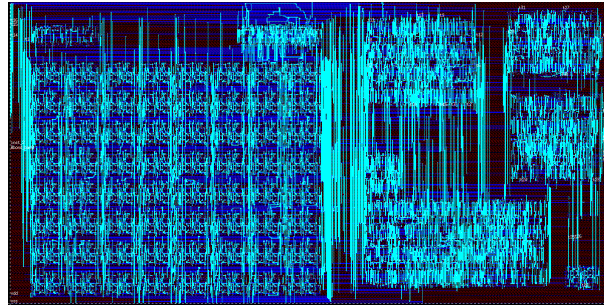
### 9.3 FPGA Prototyping

De functionaliteit van het VHDL-ontwerp van het systeem is uitgebreider getest door het op een FPGA te plaatsen. Hiervoor is het Altera DE1 FPGA-bord gebruikt. Bij deze test werd de VGA aangesloten op een externe VGA-monitor, en werden de inputs aangesloten op de fysieke schakelaars van de FPGA. Uit deze test bleek dat het systeem naar behoren functioneert op de FPGA. Hiermee is de juistheid van de VHDL-code bevestigd, maar is geen rekening gehouden met de synthese, de layout en andere aspecten die specifiek gerelateerd zijn aan de Sea-Of-Gates chip. Twee belangrijke aspecten die hierbij op de FPGA afwijken van de Sea-Of-Gates chip zijn de omzetting van het digitale RGB signaal naar een analoog RGB signaal en de generatie van een klok-puls. Deze aspecten worden door deze testmethode niet afgedekt.

### 9.4 Synthese en layout

Het VHDL-ontwerp is per deelsysteem gesynthetiseerd en er is ook per deelsysteem een layout gemaakt. De syntheses en layouts van de deelsystemen zijn beide samengevoegd tot een geheel. Het gehele ontwerp bleek synthetiseerbaar onder de gestelde eisen. Hierna is er een layout gemaakt waarbij de volledig gesynthetiseerde schakeling, inclusief testonderdelen, op de Sea-Of-Gates chip

kan worden gezet. De layout neemt een zeer groot deel van de beschikbare ruimte op de chip in beslag, maar het is wel mogelijk om alle benodigde signalen over de chip te leiden. Tevens worden hierbij de in- en uitgangspinnen succesvol naar de zijkant van de bond-bar geleid. De layout is te zien in figuur 9.1.



**Figuur 9.1:** De uiteindelijke layout van de schakeling op de Sea-Of-Gates Chip

Aantal transistoren	Efficiency
76400	40,93%

## 9.5 Switch-level-simulatie

Het gehele systeem is slechts beperkt te testen vanwege een beperking in de software met betrekking tot het aantal mogelijke klok-cycli. De testbench die gebruikt is om de VHDL-code is te lang om hier te gebruiken. Het is echter niet mogelijk om een korte testbench te maken die een goede indicatie geeft van een correcte werking. Dit ligt vooral aan het feit dat de Systeem-controller functioneert op een klok die veel langzamer is dan de klok die de schakeling binnenkrijgt. Er is geen mogelijkheid geïmplementeerd om de Systeem-controller op een andere klok te laten lopen, omdat dit de complexiteit en risico's te veel zou verhogen.

Bij de afzonderlijke deelsystemen zijn echter wel succesvolle switch-level-simulaties uitgevoerd, en korte simulaties van het geheel geven aan dat het systeem correct is verbonden. Tevens is van de losse cel die op de chip is geplaatst om te testen de werking bevestigd.

# Hoofdstuk 10

## Testplan voor de chip

De VHDL-code en, in een later stadium de chip, worden op meerdere momenten in het proces op verschillende manieren getest. Net als bij de deelsystemen wordt eerst de VHDL-code zelf gesimuleerd en na het synthetiseren wordt tevens de geëxtraheerde VHDL-code gesimuleerd. De resultaten van deze simulaties worden met elkaar vergeleken. Daarnaast wordt het gehele systeem ook getest met een FPGA. Wanneer de uiteindelijke chip geproduceerd is zal deze nog getest worden met behulp van de logic analyser.

### 10.1 Testen met de logic analyser

Na het daadwerkelijk realiseren van de chip wordt deze getest met een logic analyser. Voor het uitvoeren hiervan zijn verschillende testplannen overwogen. Hieronder worden de drie voornaamste besproken.

#### 10.1.1 Losse cel

Een goede mogelijkheid voor het testen is het kopiëren van één van de vele gesynthetiseerde blokjes die de waarde van een cel berekenen. Dit blokje wordt veel gebruikt, dus het is essentieel dat deze werkt. Een manier om een blokje te testen is hiervan één extra op de chip plaatsen en vervolgens de al aanwezige inputsignalen te gebruiken en aan te sluiten op dit blokje. Echter, er zijn 9 inputsignalen (inclusief reset) en het blokje heeft er 11 nodig. Van deze 11 is er één de klok en aangezien deze al aanwezig is op de chip kan deze worden gebruikt. Er is dus nog één extra signaal nodig, die moet dus extra aangemaakt worden. De output is een één-bit signaal die naar een pin kan worden geleid om uit te lezen.

Zodra we deze cel aan het testen zijn wordt de rest van het systeem ook aangestuurd aangezien hiervoor dezelfde aansturingsknoppen worden gebruikt. Dit is niet bezwaarlijk aangezien alle knoppen weer uit gezet kunnen worden en de reset knop het systeem weer in de oorspronkelijke toestand kan zetten.

Het nadeel aan deze vorm van testen is dat je maar één onderdeel van de chip test. Ook neemt deze manier van testen extra ruitme in beslag. Het voordeel is dat je, ook wanneer het gehele subsysteem Evaluatie & Geheugen niet werkt, kan laten zien dat de evaluatie van een losse cel wel correct functioneert.

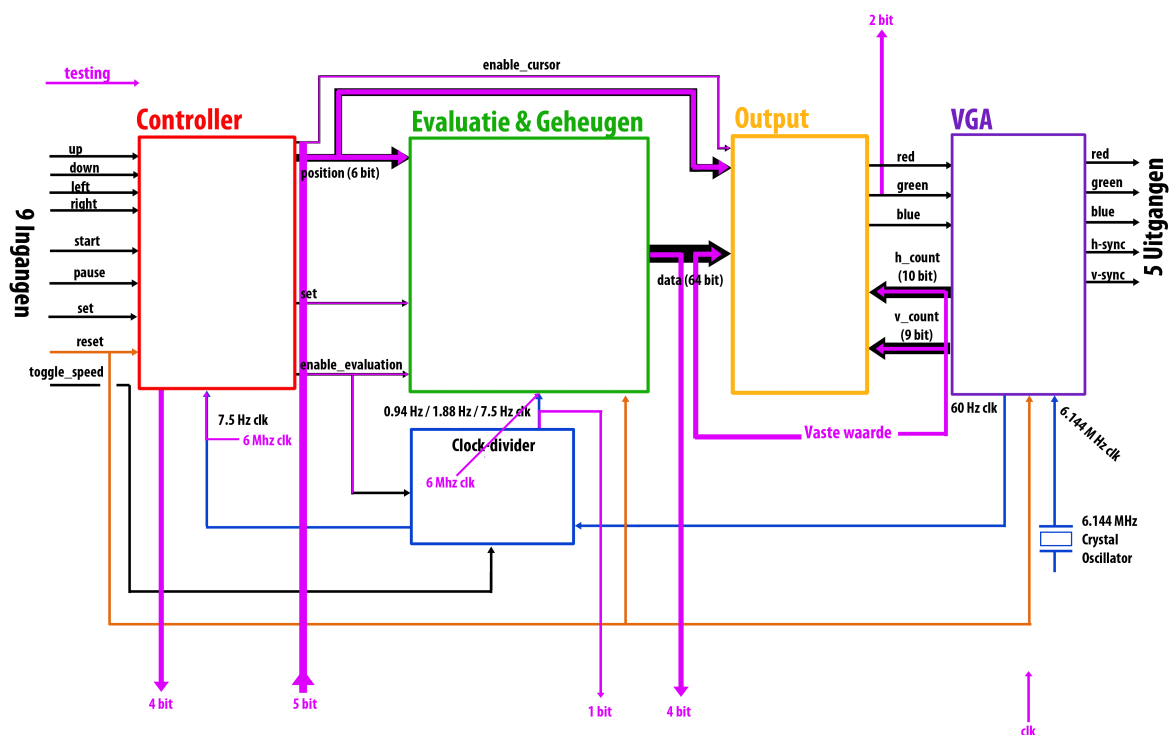
#### 10.1.2 Deelsystemen direct aansturen

Een andere teststructuur werkt op basis van multiplexers. Een extra testsignaal zal de verschillende multiplexers aansturen. Dit signaal zal bepalen welke signalen de multiplexers moeten doorgeven. De systeem-controller kan op zich worden getest, zonder het gebruik van dit systeem. Werkt echter de systeem-controller niet, dan kan je een nieuw signaal invoeren (met de logic analyser) en deze multiplexen op het subsysteem Evaluatie & Geheugen. De output van dit subsysteem wordt



(altijd) op enkele pinnen geschreven. Bij het derde subsysteem de RGB-generator wordt er door middel van een multiplexer vaste waarden aangeboden aan de ingang en de uitgang weer bekeken. Dit beschreven systeem is te vinden in figuur 10.1.

Het voordeel van dit systeem is dat elk subsysteem afzonderlijk getest kan worden. Als een voorgaand subsysteem niet functioneert, kan je het subsysteem erna nog wel goed testen. Een nadeel is dat deze teststructuur zeer uitgebreid is en daardoor veel ruimte inneemt op de chip. Tevens is een groot nadeel dat het systeem niet meer zal werken wanneer er in de teststructuur zelf fouten zitten.



Figuur 10.1: Testplan: Deelsystemen direct aansturen

### 10.1.3 Tussensignalen uitlezen

Het uitlezen van de tussensignalen is de meest eenvoudige manier van testen. Bij deze teststuctuur worden alle tussensignalen die beschikbaar zijn naar buiten geleid om deze uit te lezen. Er zijn echter niet genoeg pinnen beschikbaar om al deze signalen tegelijkertijd naar buiten te voeren. Daarom wordt er een multiplexer geïmplementeerd die een keuze maakt tussen de vele verschillende signalen. Op deze manier worden 8 signalen tegelijkertijd naar buiten geleid en aangestuurd door een 4 bits selector signaal, ookwel testsignaal genoemd.

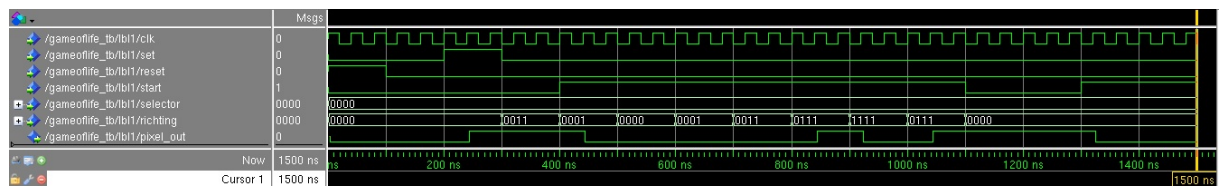
Een fout in het testsysteem, bijvoorbeeld de multiplexer, zal geen gevolgen hebben voor de rest van het systeem. Echter een cruciale fout in de Systeem-controller of de Evaluatie & Geheugen zal de rest van het systeem ontestbaar maken. Een kleinere fout zal de functie wel beïnvloeden, maar zal de volledige functionaliteit van de achterliggende subsystemen niet ontestbaar maken.

### 10.1.4 Conclusie

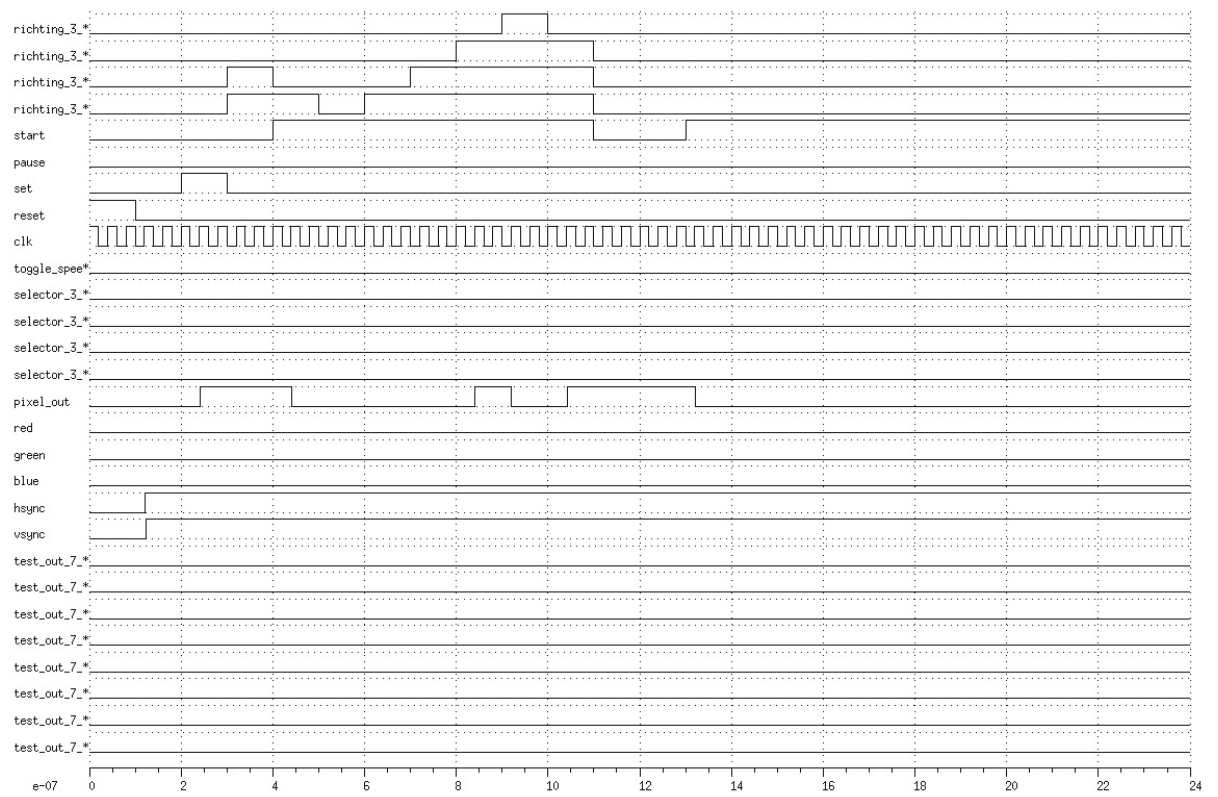
De teststructuur van *Deelstystemendirectaansturen* brengt veel risico's met zich mee. Daarnaast kost het onderdeel veel ruimte in beslag en is zeer gecompliceerd. Er is besloten om de losse cel test te combineren met de *tussensignalenuitlezen*.

Deze losse cel is geïmplementeerd en tevens gesimuleerd, zie figuur: 10.2. Hierin moet het richting-signaal gezien worden als het aantal burens. Wanneer het start-sig-naal hoog staat, zal het test-sig-naal aan gaan. De waarde van pixel\_aan geeft aan of de pixel levend of dood is. Uiteindelijk is er een switch-level simulatie gemaakt van de testcel, zie fig 10.3.

Het combineren van deze twee manieren van testen omvat de losse functie van een cel en de gehele functie van het totale ontwerp. Een kleine fout is met deze structuur nog goed te achterhalen, omdat we kunnen kijken waar de signalen afwijken van het te verwachten gedrag. Grote fouten in signalen, zo groot dat achterliggende blokken niet meer functioneren, zijn met deze keuze wel lastig te traceren. Dit maakt het moeilijk om eventuele problemen op de chip op te lossen met een verbeterde versie. Binnen dit project is het echter slechts mogelijk om één maal een chip te vervaardigen. Dit betekent dat eventuele fouten niet kunnen worden verbeterd, en dus heeft het localiseren van fouten minder prioriteit. Het voorkomen van fouten is echter des te belangrijker. Hierom is gekozen om de deelsystemen niet direct aan te sturen.



**Figuur 10.2:** De simulatie van de losse cel



**Figuur 10.3:** De switch level simulatie van de losse cel

# Hoofdstuk 11

## Verloop van het project

Op maandag 1 oktober is er, na het bespreken van een aantal opties, gemeenschappelijk besloten om als hoofdpdracht een chip te maken die een simulatie volgens de regels van Conway's Game of Life uitvoert.

Het project is verdeeld in vier verschillende onderdelen met de volgende taakverdeling:

- Niels, Patrick en Arjan ; Systeem-controller
- Pim en Leon ; Evaluatie & Geheugen
- Lisa en Bas ; RGB-generator
- Ahmet en Gert-Jan ; VGA-timer

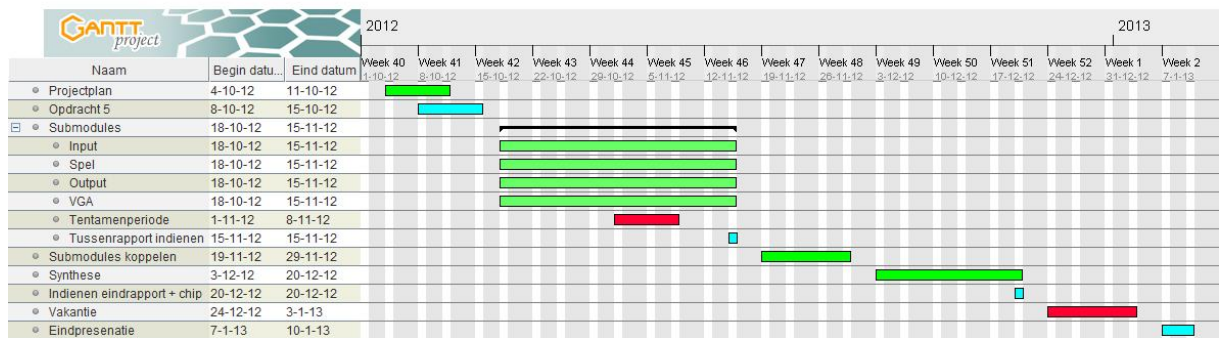
Toen de losse deelsystemen klaar waren, konden enkelen beginnen met het samenvoegen van het geheel. De anderen konden nu beginnen met het schrijven van het tussenverslag. In de planning was aangegeven dat we al 5 weken voor de deadline voor het inleveren van de code, zouden beginnen met het samenvoegen van de losse onderdelen. Dit kon toen nog niet helemaal, omdat een enkel onderdeel wat vertraging op had gelopen. Ook na het samenvoegen werden er soms nog enkele onderdelen aangepast en later werd nog de teststructuur toegevoegd. Het heeft echter wel veel voordeel opgeleverd om zo vroeg met samenvoegen te starten, aangezien daardoor redelijk op schema gebleven werd.

Het beoogde doel daarna was het eindverslag typen, wat ons eigenlijk stuitte op een nieuw probleem: de testbench van het gehele systeem. Deze testbench werd onderschat, er kwam meer bij kijken dan in eerste instantie gedacht werd. Zo waren er enkele unidentified signalen in het testresultaat die geen problemen opleverde bij de losse simulaties, maar wel bij de simulaties van het geheel. Desondanks hebben we alles gekregen voor de deadline.

Door de strakke planning, te vinden in figuur 11.1, hebben we nauwelijks extra ochtenden of middagen nodig gehad om onze code en verslag op tijd in te kunnen leveren. Dit is niet alleen te danken aan de planning, maar ook aan de goede inzet en motivatie van de projectleden.

### 11.1 FPGA Prototyping

Met een directe feedback en daadwerkelijke functionaliteit van het spel is vooral naar de Graphical User Interface (GUI) gekeken om te kijken of het eruit zag zoals wij verwachtte. Zo konden er problemen met de aansturing ontdekt worden, zoals gebruikersfouten als systeemfouten. Al snel is tijdens de FPGA Prototyping besloten om een titel boven het speelveld te zetten en een raster te maken. Verder bleek dat de reset niet aan zijn bedoelde functies voldeed, wat natuurlijk onprettig



Figuur 11.1: Projectplanning

was. Na heel wat debuggen bleek het probleem de klok. De klok werd uitgeschakeld bij een reset! Hierdoor kon de systeemcontroller niet resetten, want deze had een synchrone reset. Gelukkig was dit probleem makkelijk op te lossen door de reset overal asynchroon te ontwerpen.

## 11.2 Synthetiseren

Doordat er bij het totaalplaatje wat extra bedrading bijkomt, door bijvoorbeeld een 64-bit multiplexer voor testdoeleinden, moet er bij de synthese vooral op gelet worden dat de gigantische hoeveelheid bedrading van de chip allemaal naar behoren verloopt. Dit heeft als bijkomend voordeel dat mogelijke klokvertragingen geminimaliseerd blijft wanneer delen die met elkaar moeten communiceren bij elkaar in de buurt staan.

Al bij het maken van de eerste layout bleek dat de Evaluatie & Geheugen module een aan de hoge kant gesynthetiseerd was, waardoor deze in de hoogte net wel, of net niet op twee bondbars zou passen. Dit hebben wij dan ook aangepast in de deelmodule.

## Hoofdstuk 12

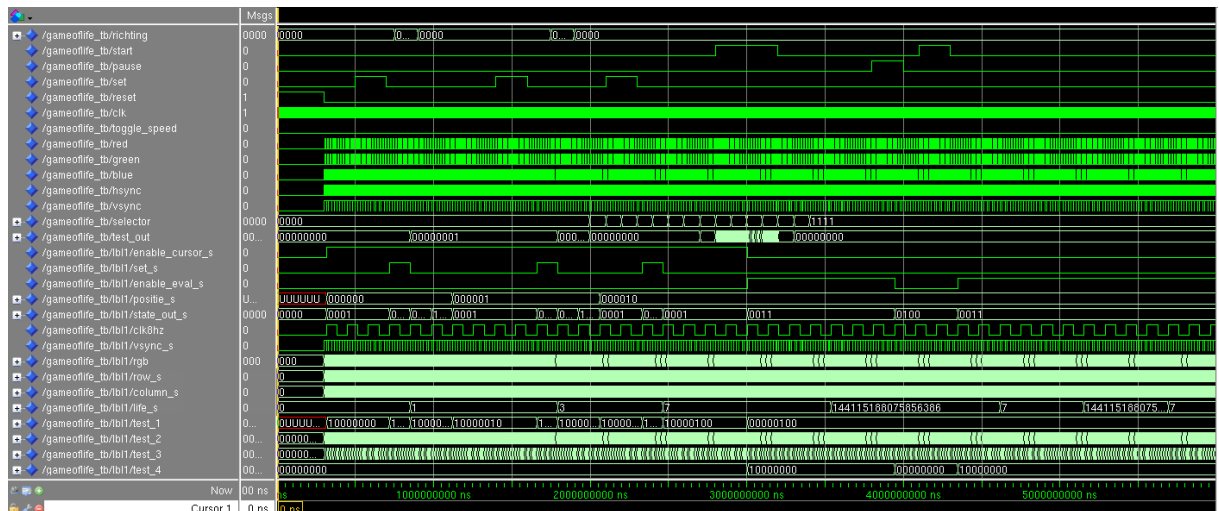
# Conclusie

Bij de start van het project is het doel gesteld een chip te ontwerpen die Conway's Game of Life kan simuleren. Er is er een implementatie gemaakt van deze functionaliteit en deze plementatie is verwerkt tot een layout gemaakt die op een Sea-of-Gates chip geplaatst kan worden. De resultaten van verschillende testen die tijdens het project zijn uitgevoerd, wijzen er op dat deze layout ook daadwerkelijk zal functioneren. Indien dit ook daadwerkelijk zo is, zijn de doelstellingen van het project binnen de gestelde tijd gehaald. Verder verwijzen we graag naar de resultaten.

# Bibliografie

- [1] Wikipedia. (2012) Conway's game of life. [Online]. Available: [http://en.wikipedia.org/wiki/Conway%27s\\_Game\\_of\\_Life](http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life) 1, 5
- [2] E. Hendriks, A. Bakker, A. Frehe, P. Groeneveld, R. Nouta, C. Verhoeven, S. de Graaf, J. Liedorp, and A. van Genderen, *Projecthandleiding ontwerp een chip*, 2012. 4, 7
- [3] Wikipedia. (2012) Rgb color model. [Online]. Available: [http://en.wikipedia.org/wiki/RGB\\_color\\_model](http://en.wikipedia.org/wiki/RGB_color_model) 11
- [4] ——. (2012) Hamming weight. [Online]. Available: [http://en.wikipedia.org/wiki/Hamming\\_weight](http://en.wikipedia.org/wiki/Hamming_weight) 19
- [5] J. Valcarce. (2011) Vga video signal format and timing specification. [Online]. Available: [http://www.javiervalcarce.eu/wiki/VGA\\_Video\\_Signal\\_Format\\_and\\_Timing\\_Specifications](http://www.javiervalcarce.eu/wiki/VGA_Video_Signal_Format_and_Timing_Specifications) 32, 33
- [6] E. Hwang. (2004) Build a vga monitor controller. [Online]. Available: <http://faculty.lasierra.edu/~ehwang/public/mypublications/VGA%20Monitor%20Controller.pdf> 33
- [7] J. Rabaey, *Digital integrated circuits*. Pearson Education, 2003.
- [8] J. D. Erwin, *Basic Engineering Circuit Analysis*. Willey, 2010.
- [9] J. Lewis. (2012) A modular single cell for conway's game of life. [Online]. Available: <http://jordanlewis.org/files/jalewis-proj.pdf>
- [10] epanorama.net. (2012) Vga timing information. [Online]. Available: [http://www.epanorama.net/documents/pc/vga\\_timing.html](http://www.epanorama.net/documents/pc/vga_timing.html)
- [11] E. Sanchez. (2012) A vga display controller. [Online]. Available: [http://lslwww.epfl.ch/pages/teaching/cours\\_lsl/ca.es/VGA.pdf](http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca.es/VGA.pdf)

## Simulatie van het complete systeem



**Figuur A.1:** Simulatieresultaten van het gehele systeem

De simulatie start in de reset-state. Hierna worden de eerste drie cellen van het raster levend gemaakt. Dit gebeurt door afwissellend het signaal *rechts* en het signaal *set* te sturen. Het signaal *rechts* is hierbij verwerkt in de *position*-vector. Na het instellen van de beginsituatie wordt de simulatie gestart. Even later wordt de simulatie gepauzeerd, om vervolgens te worden hervat. Dit is te zien aan de *pause* en *start* signalen. Dat de simulatie wordt gepauzeerd is vervolgens te zien aan het interne signaal *enable\_evaluation*. Het 64-bits signaal *life\_s* toont de huidige staat van het raster van cellen. Dit signaal is hier weergegeven als een unsigned integer. Hierbij is de eerste cel de minst significante bit en de laatste cel de meest significante bit. Het levend maken van de eerste drie cellen levert de waarden 1,3 en 7 op, respectievelijk 00(..)0001, 00(..)0011 en 00(..)0111 binair. Bij het simuleren draaien deze drie cellen een kwartslag, waarbij een van de cellen over de rand valt, en dus in de onderste rij bij een van de laatste pixels terecht komt. Deze situatie komt overeen met de integerwaarde 144115188075856386. In de volgende simulatiestap van Game of Life draaien de cellen wederom een kwartslag, om weer op dezelfde positie terug te komen. Dit is te zien aan de waarde 7. Dit patroon herhaald zich dan. Ook de signalen *clk8hz*, *hsync* en *vsync* zien er in de simulatie zoals we verwachtten.



## Bijlage B

# VHDL-code

### B.1 Top-level

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  entity gameoflife is
5      port(richting      :in      std_logic_vector(3 downto 0); -- Input system
           controller
6          start         :in      std_logic;                    -- Input system
           controller
7          pause         :in      std_logic;                    -- Input system controller
8          set           :in      std_logic;                    -- Input system controller
9          reset         :in      std_logic;                    -- Input system controller
10     xi                :in      std_logic;                    -- Crystal
11     toggle_speed      :in      std_logic;                    -- Toggle evaluation speed
12     selector          :in      std_logic_vector(3 downto 0); -- Select debug
           outputs
13     xo                :inout    std_logic;                    -- Crystal
14     pixel_out          :out      std_logic;                    -- Pixelwaarde van de
           testpixel
15     red               :out      std_logic;                    -- VGA out voor de chip
16     green              :out      std_logic;                    -- VGA out
17     blue               :out      std_logic;                    -- VGA out
18     hsync              :out      std_logic;                    -- VGA out
19     vsync              :out      std_logic;                    -- VGA out
20     test_out          :out      std_logic_vector(7 downto 0)); -- Output van de
           testmux
21 end gameoflife;
22
23 architecture structural of gameoflife is
24 component input
25     port (clk          : in      std_logic;
26           richting     : in      std_logic_vector(3 downto 0);
27           start        : in      std_logic;
28           pauze        : in      std_logic;
29           reset        : in      std_logic;
30           in_set       : in      std_logic;
31           positie      : out      std_logic_vector(5 downto 0);
32           enable_cursor : out      std_logic;
33           out_set      : out      std_logic;
34           enable_eval   : out      std_logic;
```

```

35     state_out    : out    std_logic_vector(3 downto 0));
36 end component;
37
38 component vga
39     port(clk, reset    : in    std_logic;
40           r_in         : in    std_logic;
41           g_in         : in    std_logic;
42           b_in         : in    std_logic;
43           hsync, vsync : out    std_logic;
44           r_out        : out    std_logic;
45           g_out        : out    std_logic;
46           b_out        : out    std_logic;
47           row          : out    std_logic_vector(9 downto 0);
48           column       : out    std_logic_vector(7 downto 0));
49 end component;
50
51 component vga_top
52     port(life_in       : in    std_logic_vector(63 downto 0);
53           cursor       : in    std_logic_vector(5 downto 0);
54           enable_cursor : in    std_logic;
55           h_teller     : in    std_logic_vector(7 downto 0);
56           v_teller     : in    std_logic_vector(9 downto 0);
57           rgb_out      : out    std_logic_vector(2 downto 0));
58 end component;
59
60 component evaluation
61     port(clk          : in    std_logic;
62           set         : in    std_logic;
63           reset       : in    std_logic;
64           enable      : in    std_logic;
65           switch      : in    std_logic;
66           position    : in    std_logic_vector(5 downto 0);
67           clk8hz      : out    std_logic;
68           data_out    : out    std_logic_vector(63 downto 0));
69 end component;
70
71 component test_selector is
72     port(selector      : in    std_logic_vector(3 downto 0);
73           data_out     : out    std_logic_vector(7 downto 0);
74           data_in_0    : in    std_logic_vector(7 downto 0);
75           data_in_1    : in    std_logic_vector(7 downto 0);
76           data_in_2    : in    std_logic_vector(7 downto 0);
77           data_in_3    : in    std_logic_vector(7 downto 0);
78           data_in_4    : in    std_logic_vector(7 downto 0);
79           data_in_5    : in    std_logic_vector(7 downto 0);
80           data_in_6    : in    std_logic_vector(7 downto 0);
81           data_in_7    : in    std_logic_vector(7 downto 0);
82           data_in_8    : in    std_logic_vector(7 downto 0);
83           data_in_9    : in    std_logic_vector(7 downto 0);
84           data_in_10   : in    std_logic_vector(7 downto 0);
85           data_in_11   : in    std_logic_vector(7 downto 0);
86           data_in_12   : in    std_logic_vector(7 downto 0);
87           data_in_13   : in    std_logic_vector(7 downto 0);
88           data_in_14   : in    std_logic_vector(7 downto 0);
89           data_in_15   : in    std_logic_vector(7 downto 0));
90 end component;
91
92 component pixel is

```

---

```

93     port(clk      : in    std_logic;
94           reset   : in    std_logic;
95           set     : in    std_logic;
96           enable  : in    std_logic;
97           n1      : in    std_logic;
98           n2      : in    std_logic;
99           n3      : in    std_logic;
100          n4      : in    std_logic;
101          n5      : in    std_logic;
102          n6      : in    std_logic;
103          n7      : in    std_logic;
104          n8      : in    std_logic;
105          d       : out   std_logic);
106 end component;
107
108 component osc10 is
109     port(e       : in    std_logic;
110           f       : out   std_logic;
111           xi      : in    std_logic;
112           xo      : inout std_logic
113     );
114 end component;
115
116 component buf40 is
117     port(a       : in    std_logic;
118           y       : out   std_logic
119     );
120 end component;
121
122 signal enable_cursor_s, set_s, enable_eval_s, xi_buffered, clk8hz, vsync_s
123       , clk_6mhz : std_logic;
124 signal rgb       : std_logic_vector(2 downto 0);
125 signal state_out_s : std_logic_vector(3 downto 0);
126 signal positie_s : std_logic_vector(5 downto 0);
127 signal column_s, test_1, test_2, test_3, test_4, test_5, test_6 :
128       std_logic_vector(7 downto 0);
129 signal row_s      : std_logic_vector(9 downto 0);
130 signal life_s     : std_logic_vector(63 downto 0);
131
132 begin
133     buf: buf40
134     port map (
135         a      => xi,
136         y      => xi_buffered
137     );
138     osc: osc10
139     port map (
140         e      => '1',
141         f      => clk_6mhz,
142         xi     => xi_buffered,
143         xo     => xo
144     );
145     controllbl: input
146     port map (
147         clk    => clk8hz,

```

```

149     richting    => richting,
150     start       => start,
151     pauze       => pause,
152     reset       => reset,
153     in_set      => set,
154     positie     => positie_s,
155     enable_cursor => enable_cursor_s,
156     out_set     => set_s,
157     enable_eval  => enable_eval_s,
158     state_out   => state_out_s
159 );
160
161 vgalbl: vga
162     port map (
163         clk        => clk_6mhz ,
164         reset      => reset,
165         r_in       => rgb(0),
166         g_in       => rgb(1),
167         b_in       => rgb(2),
168         hsync      => hsync,
169         vsync      => vsync_s,
170         r_out      => red,
171         g_out      => green,
172         b_out      => blue,
173         row        => row_s,
174         column     => column_s
175     );
176
177 outputlbl: vga_top
178     port map(
179         life_in     => life_s,
180         cursor      => positie_s,
181         enable_cursor => enable_cursor_s,
182         h_teller    => column_s,
183         v_teller    => row_s,
184         rgb_out     => rgb
185     );
186
187 evaluationlbl: evaluation
188     port map (
189         clk         => vsync_s,
190         set         => set_s,
191         reset       => reset,
192         enable      => enable_eval_s,
193         switch      => toggle_speed,
194         position    => positie_s,
195         clk8hz      => clk8hz,
196         data_out    => life_s
197     );
198
199 testerlbl: test_selector
200     port map(
201         selector    => selector,
202         data_out    => test_out,
203         data_in_0   => life_s(7 downto 0),
204         data_in_1   => life_s(15 downto 8),
205         data_in_2   => life_s(23 downto 16),
206         data_in_3   => life_s(31 downto 24),

```

```

207     data_in_4    => life_s(39 downto 32),
208     data_in_5    => life_s(47 downto 40),
209     data_in_6    => life_s(55 downto 48),
210     data_in_7    => life_s(63 downto 56),
211     data_in_8    => test_1,
212     data_in_9    => test_2,
213     data_in_10   => row_s(7 downto 0),
214     data_in_11   => test_3,
215     data_in_12   => column_s(7 downto 0),
216     data_in_13   => test_4,
217     data_in_14   => "00000000",
218     data_in_15   => "00000000"
219 );
220
221 pixeltestlbl: pixel
222     port map(
223         clk        => clk_6mhz,
224         reset      => reset,
225         set        => set,
226         enable     => start,
227         n1         => richting(0),
228         n2         => richting(1),
229         n3         => richting(2),
230         n4         => richting(3),
231         n5         => selector(0),
232         n6         => selector(1),
233         n7         => selector(2),
234         n8         => selector(3),
235         d          => pixel_out
236     );
237
238 test_1 <= (enable_cursor_s & positie_s & set_s);
239 test_2 <= (rgb & "00000");
240 test_3 <= (row_s(9 downto 8) & "000000");
241 test_4 <= (enable_eval_s & "0000000");
242
243 vsync <= vsync_s;--tussensignaal omdat de vsync op de chip meermaals
    gebruikt wordt, maar ook uitgangssignaal is
244
245 end structural;

```

./vhdl/gameoflife.vhd

## B.2 Systeem-controller

```

1  -----
2  -- VHDL Beschrijving voor de systeemcontroller van Conway's Game of Life
3  -- December 2012, EE2821 Project "maak een chip" (2012-2013)
4  -- Laatste aangepast op 17 December, door Patrick Fuchs
5  -----
6  library IEEE;
7  use IEEE.std_logic_1164.ALL;
8  use IEEE.numeric_std.ALL;
9
10 entity input is
11     port(clk        : in    std_logic;
12          richting    : in    std_logic_vector(3 downto 0);
13          start        : in    std_logic;

```

```

14     pauze      : in      std_logic;
15     reset      : in      std_logic;
16     in_set     : in      std_logic;
17     positie    : out     std_logic_vector(5 downto 0);
18     enable_cursor : out   std_logic;
19     out_set     : out     std_logic;
20     enable_eval : out     std_logic;
21     state_out  : out     std_logic_vector(3 downto 0));
22 end input;
23
24 architecture behaviour of input is
25 -- state definities
26 type input_state is (reset_state, positie_state, up, down, left, right,
27     plaats, spel, stop);
28 -- ervoor zorgen dat reset als 0 bits gecodeerd wordt (want deze is
29     asynchroon)
30 attribute enum_encoding : string;
31 attribute enum_encoding of input_state : type is "0000 0001 0010 0011 0100
32     0101 0111 1000 1001";
33
34 -- signalen voor de states
35 signal state, new_state : input_state;
36 -- signalen voor de pointer die bijhoudt waar op het scherm de cursor is
37 signal new_positieptr, positieptr : unsigned(5 downto 0);
38 -- tussensignalen voor input-buffer
39 signal start_s, pauze_s, in_set_s : std_logic;
40 signal richting_s, state_val : std_logic_vector(3 downto 0);
41
42 begin
43
44 process(clk)
45 begin
46     if(reset = '1') then
47         state <= reset_state;
48     elsif (clk'event and clk = '1') then
49         state <= new_state;
50         positieptr <= new_positieptr;
51     end if;
52 end process;
53
54 --state definities, zie FSM schema voor meer uitleg
55 process(state, start_s, pauze_s, reset, in_set_s, richting_s)
56 begin
57     case state is
58         when reset_state =>
59             enable_cursor <= '0';
60             out_set <= '0';
61             enable_eval <= '0';
62             new_positieptr <= "000000";
63             new_state <= positie_state;
64             state_val <= "0000";
65         when positie_state =>
66             enable_cursor <= '1';
67             out_set <= '0';
68             enable_eval <= '0';
69             state_val <= "0001";
70             if( in_set_s = '1' and
71                 richting_s = "0000" and

```

```
69     start_s = '0'      and
70     pauze_s = '0'      ) then
71     new_state <= plaats;
72 elsif( in_set_s = '0'  and
73     richting_s = "0001" and
74     start_s = '0'      and
75     pauze_s = '0'      ) then
76     new_state <= up;
77 elsif( in_set_s = '0'  and
78     richting_s = "0010" and
79     start_s = '0'      and
80     pauze_s = '0'      ) then
81     new_state <= right;
82 elsif( in_set_s = '0'  and
83     richting_s = "0100" and
84     start_s = '0'      and
85     pauze_s = '0'      ) then
86     new_state <= down;
87 elsif( in_set_s = '0'  and
88     richting_s = "1000" and
89     start_s = '0'      and
90     pauze_s = '0'      ) then
91     new_state <= left;
92 elsif( in_set_s = '0'  and
93     richting_s = "0000" and
94     start_s = '1'      and
95     pauze_s = '0'      ) then
96     new_state <= spel;
97 else
98     new_state <= state;
99 end if;
100 new_positieptr <= positieptr;
101 when plaats =>
102     state_val <= "0010";
103     enable_cursor <= '1';
104     out_set <= '1';
105     enable_eval <= '0';
106     new_state <= positie_state;
107     new_positieptr <= positieptr;
108 when spel =>
109     state_val <= "0011";
110     enable_cursor <= '0';
111     out_set <= '0';
112     enable_eval <= '1';
113     if( in_set_s = '0'      and
114         richting_s = "0000" and
115         start_s = '0'      and
116         pauze_s = '1'      ) then
117         new_state <= stop;
118     else
119         new_state <= state;
120     end if;
121     new_positieptr <= positieptr;
122 when stop =>
123     state_val <= "0100";
124     enable_cursor <= '0';
125     out_set <= '0';
126     enable_eval <= '0';
```

```

127     if( in_set_s = '0'      and
128         richting_s = "0000" and
129         start_s = '1'      and
130         pauze_s = '0'      ) then
131         new_state <= spel;
132     else
133         new_state <= state;
134     end if;
135     new_positieptr <= positieptr;
136     when up =>
137         state_val <= "0101";
138         new_positieptr <= positieptr - 8;
139         enable_cursor <= '1';
140         out_set <= '0';
141         enable_eval <= '0';
142         new_state <= positie_state;
143     when down =>
144         state_val <= "0110";
145         new_positieptr <= positieptr + 8;
146         enable_cursor <= '1';
147         out_set <= '0';
148         enable_eval <= '0';
149         new_state <= positie_state;
150     when left =>
151         state_val <= "0111";
152         if(positieptr(2 downto 0) = "000") then
153             new_positieptr <= positieptr + 7;
154         else
155             new_positieptr <= positieptr - 1;
156         end if;
157         enable_cursor <= '1';
158         out_set <= '0';
159         enable_eval <= '0';
160         new_state <= positie_state;
161     when right =>
162         state_val <= "1000";
163         if(positieptr(2 downto 0) = "111") then
164             new_positieptr <= positieptr - 7;
165         else
166             new_positieptr <= positieptr + 1;
167         end if;
168         enable_cursor <= '1';
169         out_set <= '0';
170         enable_eval <= '0';
171         new_state <= positie_state;
172     end case;
173 end process;
174
175 -- input buffer: 1 flip-flop buffer (zonder reset),
176 -- aangezien een klein beetje skew tijdens het uitlezen
177 -- van het ingangssignaal triviaal is voor de verdere
178 -- werking van het circuit. Als er een verkeerde uitlezing
179 -- was, is deze een klokslag later opgelost -> dwz, het
180 -- knopje moet dan een fractie langer ingedrukt blijven.
181 process(clk)
182 begin
183     if (clk'event and clk = '1') then
184         richting_s <= richting;

```



```

185 start_s <= start;
186 in_set_s <= in_set;
187 pauze_s <= pauze;
188 end if;
189 end process;
190
191 --uitgangen
192 positie <= std_logic_vector(positieptr);
193 state_out <= std_logic_vector(state_val);
194
195 end behaviour;

```

./vhdl/input-behaviour.vhd

## B.3 Evaluatie & Geheugen

### B.3.1 Evaluatie & Geheugen Toplevel

```

1  -----
2  -- VHDL Beschrijving voor de evaluatie van Conway's Game of Life,
   inclusief clock-divider
3  -- Door Leon Noordam & Pim Veldhuisen
4  -- November 2012, EE2821 Project "maak een chip" (2012-2013)
5  -----
6
7  library IEEE;
8  use IEEE.std_logic_1164.ALL;
9
10 entity evaluation is
11     port( clk      :in    std_logic;
12           set       :in    std_logic;
13           reset     :in    std_logic;
14           enable    :in    std_logic;
15           switch    :in    std_logic;
16           position  :in    std_logic_vector(5 downto 0);
17           lk8hz     :out   std_logic;
18           data_out  :out   std_logic_vector(63 downto 0));
19 end evaluation;
20
21 architecture behaviour of evaluation is
22     component demux_6_to_64 is
23         port( enable :in    std_logic;
24               position :in    std_logic_vector(5 downto 0);
25               data_out :out   std_logic_vector(63 downto 0));
26     end component;
27
28     component pixels is
29         port( set      :in    std_logic_vector(63 downto 0);
30               reset    :in    std_logic;
31               enable   :in    std_logic;
32               clk      :in    std_logic;
33               data     :out   std_logic_vector(63 downto 0));
34     end component;
35
36     component clock_divider is
37         port( clk_in   :in    std_logic;
38               reset    :in    std_logic;
39               switch   :in    std_logic;
40               enable   :in    std_logic;

```

```

41         clk8hz      :out    std_logic;
42         clk_out     :out    std_logic
43     );
44 end component;
45
46 signal clk_internal: std_logic;
47 signal demux_out: std_logic_vector (63 downto 0);
48 begin
49     pix: pixels port map (demux_out, reset, enable, clk_internal, data_out);
50     demux: demux_6_to_64 port map (set, position, demux_out);
51     clk_div: clock_divider port map (clk, reset, switch, enable, clk8hz,
        clk_internal);
52 end behaviour;

```

./vhdl/evaluation.vhd

### B.3.2 Clock-divider

```

1  -----
2  -- VHDL Beschrijving van een clock-divider voor gebruik in Game of Life
3  -- Door Leon Noordam & Pim Veldhuisen
4  -- November 2012, EE2821 Project "maak een chip" (2012-2013)
5  -----
6
7
8  library IEEE;
9  use IEEE.std_logic_1164.ALL;
10
11 entity clock_divider is
12     port(clk_in :in    std_logic;
13          reset  :in    std_logic;
14          switch :in    std_logic;
15          enable  :in    std_logic;
16          clk8hz :out    std_logic;
17          clk_out:out    std_logic
18     );
19 end clock_divider;
20
21 architecture behaviour of clock_divider is
22     signal clk1, clk2, clk3: std_logic;
23     signal counter : integer range 0 to 31 := 0;
24 begin
25     process(reset, clk_in)
26     begin
27         if (reset = '1') then
28             clk1 <= '0';
29             clk2 <= '0';
30             clk3 <= '0';
31             counter <= 0;
32         elsif (clk_in'event and clk_in = '1') then
33             if (counter mod 32 = 1) then
34                 clk1 <= not clk1;
35             end if;
36             if (counter mod 16 = 1) then
37                 clk2 <= not clk2;
38             end if;
39             if (counter mod 4 = 1) then
40                 clk3 <= not clk3;
41             end if;

```

```

42     if (counter = 31) then
43         counter <= 0;
44     else
45         counter <= counter + 1;
46     end if;
47 end if;
48 end process;
49
50 process(switch, enable, clk1, clk2, clk3)
51 begin
52     if (enable = '1') then
53         if (switch = '1') then
54             clk_out <= clk1;
55         else
56             clk_out <= clk2;
57         end if;
58     else
59         clk_out <= clk3;
60     end if;
61 end process;
62
63 clk8hz <= clk3;
64
65 end behaviour;

```

./vhdl/clock\_divider.vhd

### B.3.3 Pixels

```

1  -----
2  -- VHDL beschrijving van het grid van pixels van Game of Life, gebruik
   -- maken van een generate statement
3  -- Door Leon Noordam & Pim Veldhuisen
4  -- November 2012, EE2821 Project "maak een chip" (2012-2013)
5  -----
6
7  library IEEE;
8  use IEEE.std_logic_1164.ALL;
9  use IEEE.numeric_std.all;
10
11 entity pixels is
12     port(set      :in      std_logic_vector(63 downto 0);
13          reset    :in      std_logic;
14          enable   :in      std_logic;
15          clk      :in      std_logic;
16          data     :out      std_logic_vector(63 downto 0));
17 end pixels;
18
19 architecture structural of pixels is
20     component pixel is
21         port(clk, reset, set, enable, n1, n2, n3, n4, n5, n6, n7, n8: in
22             std_logic;
23             d : out std_logic);
24     end component;
25     signal internal: std_logic_vector(63 downto 0);
26     constant SIZE: natural := 8;
27 begin
28     -- Gebruikte grid voor de blokjes, in het generate statment is i de

```

```
        positie van het blokje
29  -- 0  1  2  3  4  5  6  7
30  -- 8  9 10 11 12 13 14 15
31  -- 16 17 18 19 20 21 22 23
32  -- 24 25 26 27 28 29 30 31
33  -- 32 33 34 35 36 37 38 39
34  -- 40 41 42 43 44 45 46 47
35  -- 48 49 50 51 52 53 54 55
36  -- 56 57 58 59 60 61 62 63
37  gen_pixels: for i in 0 to (SIZE*SIZE-1) generate
38      signal n1, n2, n3, n4, n5, n6, n7, n8: natural;
39      signal s1, s2, s3, s4, s5, s6, s7, s8: std_logic;
40
41  begin
42      --Linksboven
43      n1 <= i-SIZE-1 when (i-SIZE > 0 and i mod SIZE /= 0) else
44          i+SIZE*SIZE-1 when (i = 0) else
45          i+SIZE*(SIZE-1)-1 when (i-SIZE < 0) else
46          i-1;
47      s1 <= internal(n1);
48
49      -- Boven
50      n2 <= i-SIZE when (i-SIZE+1 > 0) else i+SIZE*(SIZE-1);
51      s2 <= internal(n2);
52
53      -- Rechtsboven
54      n3 <= i-SIZE+1 when (i-SIZE+2 > 0 and (i+1) mod SIZE /= 0) else
55          i+SIZE*(SIZE-2)+1 when (i = SIZE-1) else
56          i+SIZE*(SIZE-1)+1 when (i < SIZE-1) else
57          i-SIZE*2+1;
58      s3 <= internal(n3);
59
60      -- Links
61      n4 <= i-1 when (i mod SIZE /= 0) else i+SIZE-1;
62      s4 <= internal(n4);
63
64      -- Rechts
65      n5 <= i+1 when ((i+1) mod SIZE /= 0) else i-SIZE+1;
66      s5 <= internal(n5);
67
68      -- Linksonder
69      n6 <= i+SIZE-1 when (i+SIZE <= SIZE*SIZE and i mod SIZE /= 0) else
70          i-SIZE*(SIZE-2)-1 when (i = SIZE*(SIZE-1)) else
71          i+SIZE*2-1 when (i mod SIZE = 0) else
72          i-SIZE*(SIZE-1)-1;
73      s6 <= internal(n6);
74
75      -- Onder
76      n7 <= i+SIZE when (i+SIZE+1 <= SIZE*SIZE) else i-SIZE*(SIZE-1);
77      s7 <= internal(n7);
78
79      -- Rechtsonder
80      n8 <= i+SIZE+1 when (i+SIZE+2 <= SIZE*SIZE and (i+1) mod SIZE /= 0)
81          else
82          i-SIZE*SIZE+1 when (i = (SIZE*SIZE)-1) else
83          i-SIZE*(SIZE-1)+1 when (i > SIZE*(SIZE-1)-1) else
84          i+1;
85      s8 <= internal(n8);
```

```

85
86     pix: pixel port map (clk, reset, set(i), enable, s1, s2, s3, s4, s5,
87                          s6, s7, s8, internal(i));
88     end generate gen_pixels;
89     data <= internal;
90 end structural;

```

./vhdl/pixels.vhd

### B.3.4 Pixel

```

1  -----
2  -- VHDL beschrijving een cel van Game of Life, inclusief combinatorische
3  -- Door Leon Noordam & Pim Veldhuisen
4  -- November 2012, EE2821 Project "maak een chip" (2012-2013)
5  -----
6
7  library IEEE;
8  use IEEE.std_logic_1164.ALL;
9
10 architecture structural of pixel is
11     component d_flipflop is
12         port(d, clk, reset, set : in std_logic;
13              q : out std_logic);
14     end component;
15     component full_adder is
16         port(A, B, Cin : in std_logic;
17              s, Cout : out std_logic);
18     end component;
19     component half_adder is
20         port(A, B : in std_logic;
21              s, Cout : out std_logic);
22     end component;
23     component selector is
24         port(A, B, S : in std_logic;
25              data : out std_logic);
26     end component;
27     signal s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15,
28            s16, s17: std_logic;
29 begin
30     fa1: full_adder port map (n1, n2, n3, s2, s1);
31     fa2: full_adder port map (n4, n5, n6, s4, s3);
32     ha1: half_adder port map (n7, n8, s6, s5);
33
34     fa3: full_adder port map (s1, s3, s7, s9, s10);
35     ha2: half_adder port map (s2, s4, s8, s7);
36
37     fa4: full_adder port map (s5, s9, s11, s13, s12);
38     ha3: half_adder port map (s6, s8, s14, s11);
39
40     sel: selector port map (s15, s16, enable, s17);
41     dff: d_flipflop port map (s17, clk, reset, set, s16);
42     s15 <= (s10 nor s12) and ((s13 and s14) or (s13 and s16));
43     d <= s16;
44 end structural;

```

./vhdl/pixel.vhd

## B.4 RGB-generator

### B.4.1 VGA\_top

```

1  -----
2  -- VHDL beschrijving van de vga-top entitiy die alle in- en
   -- uitgangssignalen doorverwijst naar de juiste entities
3  -- Door Bas Generowicz & Lisa Audenaert
4  -- December 2012, EE2821 Project "maak een chip" (2012-2013)
5  -----
6
7  library IEEE;
8  use IEEE.std_logic_1164.ALL;
9  use IEEE.numeric_std.ALL;
10
11 entity vga_top is
12 port (
13     life_in      : in  std_logic_vector(63 downto 0);
14     cursor       : in  std_logic_vector(5  downto 0);
15     enable_cursor : in  std_logic;
16     h_teller     : in  std_logic_vector(7  downto 0);
17     v_teller     : in  std_logic_vector(9  downto 0);
18     rgb_out      : out std_logic_vector(2  downto 0)
19 );
20 end vga_top;
21
22 architecture structural of vga_top is
23
24 component vga_control is
25     port (
26         rgb_game      : in  std_logic_vector(1  downto 0);
27         rgb_text      : in  std_logic;
28         h_teller      : in  std_logic_vector(7  downto 0);
29         v_teller      : in  std_logic_vector(9  downto 0);
30         rgb_out       : out std_logic_vector(2  downto 0)
31     );
32
33 );
34 end component;
35
36 component vga_game is
37     port (
38         life_in      : in  std_logic_vector(63 downto 0);
39         cursor       : in  std_logic_vector(5  downto 0);
40         enable_cursor : in  std_logic;
41         h_teller     : in  std_logic_vector(7  downto 0);
42         v_teller     : in  std_logic_vector(9  downto 0);
43         rgb_game     : out std_logic_vector(1  downto 0)
44     );
45 end component;
46
47 component vga_text is
48     port (
49         h_teller     : in  std_logic_vector(7  downto 0);
50         v_teller     : in  std_logic_vector(9  downto 0);
51         rgb_text     : out std_logic
52     );
53 end component;

```

```

54
55 signal rgb_game_s: std_logic_vector(1 downto 0);
56 signal rgb_text_s: std_logic;
57 begin
58
59 lbl1: vga_control port map(
60     rgb_game => rgb_game_s,
61     rgb_text => rgb_text_s,
62     h_teller => h_teller,
63     v_teller => v_teller,
64     rgb_out  => rgb_out
65 );
66
67 lbl2: vga_game port map(
68     life_in  => life_in,
69     cursor   => cursor,
70     enable_cursor => enable_cursor,
71     h_teller => h_teller,
72     v_teller => v_teller,
73     rgb_game => rgb_game_s
74 );
75
76 lbl3: vga_text port map(
77     h_teller => h_teller,
78     v_teller => v_teller,
79     rgb_text => rgb_text_s
80 );
81
82
83 end structural;

```

./vhdl/vga\_top.vhd

## B.4.2 VGA\_control

```

1  -----
2  -- VHDL beschrijving van de vga controlentitiy die bepaalt welk rgb-
3  -- signaal wordt doorgestuurd en of het spel zich in het speelveld bevindt
4  -- Door Bas Generowicz & Lisa Audenaert
5  -- December 2012, EE2821 Project "maak een chip" (2012-2013)
6  -----
7
8 library IEEE;
9 use IEEE.std_logic_1164.ALL;
10 use IEEE.numeric_std.ALL;
11
12 entity vga_control is
13     port (
14         rgb_game      :in  std_logic_vector(1 downto 0);
15         rgb_text       :in  std_logic;
16         h_teller       :in  std_logic_vector(7 downto 0);
17         v_teller       :in  std_logic_vector(9 downto 0);
18         rgb_out        :out std_logic_vector(2 downto 0)
19     );
20 end vga_control;
21
22
23 architecture behaviour of vga_control is

```

```

24
25     constant offset_y : integer := 150;
26
27 begin
28
29 process (h_teller, v_teller)
30     variable v_tel: unsigned (9 downto 0);
31     variable h_tel: unsigned (7 downto 0);
32 begin
33
34     h_tel := unsigned(h_teller);
35     v_tel := unsigned(v_teller);
36
37     if (((v_tel <= (offset_y) and v_tel >= (offset_y - 2)) and (h_tel <= 112
38         and h_tel >= 48)) or
39         ((v_tel <= (258 + offset_y) and v_tel >= (256 + offset_y)) and (
40             h_tel <= 112 and h_tel >= 48)) or
41         ((v_tel <= (256 + offset_y) and v_tel >= (offset_y)) and (h_tel =
42             112 or h_tel = 48))) then
43         rgb_out <= "111";
44
45         --test if binnen speelveld
46     elsif ((v_tel <= (256 + offset_y) and v_tel >= (offset_y)) and (h_tel <
47         112 and h_tel > 48)) then
48         rgb_out <= '0' & rgb_game;
49     else
50         rgb_out <= rgb_text & "00";
51     end if;
52
53 end process;
54 end behaviour;

```

./vhdl/vga\_control.vhd

### B.4.3 VGA\_game

```

1  -----
2  -- VHDL beschrijving van de vga-game entitiy die het rgb-signaal genereert
3  -- als het spel zich in het speelveld bevindt
4  -- Door Bas Generowicz & Lisa Audenaert
5  -- December 2012, EE2821 Project "maak een chip" (2012-2013)
6  -----
7
8  library IEEE;
9  use IEEE.std_logic_1164.ALL;
10 use IEEE.numeric_std.ALL;
11
12 entity vga_game is
13     port (
14         life_in      : in  std_logic_vector(63 downto 0);
15         cursor       : in  std_logic_vector(5  downto 0);
16         enable_cursor : in  std_logic;
17         h_teller     : in  std_logic_vector(7  downto 0);
18         v_teller     : in  std_logic_vector(9  downto 0);
19         rgb_game     : out std_logic_vector(1  downto 0)
20     );
21 end vga_game;
22

```



```

23 architecture behaviour of vga_game is
24
25 signal x_pixel: unsigned (3 downto 0);
26 signal y_pixel: unsigned (3 downto 0);
27 signal pixel: unsigned (6 downto 0);
28
29 constant offset_x : integer := 48;
30 constant offset_y : integer := 150;
31
32
33 begin
34
35 process (h_teller, v_teller)
36 variable v_tel: unsigned (9 downto 0);
37 variable h_tel: unsigned (7 downto 0);
38 begin
39
40     h_tel := unsigned(h_teller);
41     v_tel := unsigned(v_teller);
42
43
44     if ((h_tel = (0 + offset_x)) or (h_tel = (8 + offset_x)) or (h_tel =
45         (16 + offset_x)) or (h_tel = (24 + offset_x)) or (h_tel = (32 +
46         offset_x)) or (h_tel = (40 + offset_x)) or (h_tel = (48 + offset_x))
47         or (h_tel = (56 + offset_x)) or (h_tel = (64 + offset_x))) then
48         x_pixel <= "1100";
49
50         elsif (h_tel < (8 + offset_x)) then -- pixel 1, x < 8
51             x_pixel <= "0000";
52
53         elsif (h_tel < (16 + offset_x)) then -- pixel 2, x < 16
54             x_pixel <= "0001";
55
56         elsif (h_tel < (24 + offset_x)) then -- pixel 3, x < 24
57             x_pixel <= "0010";
58
59         elsif (h_tel < (32 + offset_x)) then -- pixel 4, x < 32
60             x_pixel <= "0011";
61
62         elsif (h_tel < (40 + offset_x)) then -- pixel 5, x < 40
63             x_pixel <= "0100";
64
65         elsif (h_tel < (48 + offset_x)) then -- pixel 6, x < 48
66             x_pixel <= "0101";
67
68         elsif (h_tel < (56 + offset_x)) then -- pixel 7, x < 56
69             x_pixel <= "0110";
70
71         elsif (h_tel < (64 + offset_x)) then -- pixel 8, x < 64
72             x_pixel <= "0111";
73
74     else
75         x_pixel <= "1000"; -- buiten pixels (zou nooit mogen gebeuren)
76
77 end if;

```

```

78   if ((v_tel =(0  + offset_y) and v_tel < (2 + offset_y)) or (v_tel > (30
    + offset_y) and v_tel < (34 + offset_y)) or (v_tel > (62  +
    offset_y) and v_tel < (66 + offset_y)) or (v_tel >(94 + offset_y)
    and v_tel < (98 + offset_y)) or (v_tel > (126 + offset_y) and v_tel
    < (130 + offset_y)) or (v_tel > (158 + offset_y) and v_tel < (162 +
    offset_y)) or (v_tel > (190 + offset_y) and v_tel < (194 + offset_y)
    ) or (v_tel > (222 + offset_y) and v_tel < (226 + offset_y)) or (
    v_tel > (254 + offset_y) and v_tel < (258 + offset_y))) then
79   y_pixel <= "1100";
80
81   elsif (v_tel < (32 + offset_y)) then -- pixel 1, y < 32
82     y_pixel <= "0000";
83
84   elsif (v_tel < (64 + offset_y)) then -- pixel 2, y < 64
85     y_pixel <= "0001";
86
87   elsif (v_tel < (96 + offset_y)) then -- pixel 3, y < 96
88     y_pixel <= "0010";
89
90   elsif (v_tel < (128+ offset_y)) then -- pixel 4, y < 128
91     y_pixel <= "0011";
92
93   elsif (v_tel < (160 + offset_y)) then -- pixel 5, y < 160
94     y_pixel <= "0100";
95
96   elsif (v_tel < (192 + offset_y)) then -- pixel 6, y < 192
97     y_pixel <= "0101";
98
99   elsif (v_tel < (224 + offset_y)) then -- pixel 7, y < 224
100     y_pixel <= "0110";
101
102   elsif (v_tel < (256 + offset_y)) then -- pixel 8, y < 256
103     y_pixel <= "0111";
104
105   else
106     y_pixel <= "1000"; -- buiten pixels (zou nooit mogen gebeuren)
107   end if;
108
109 end process;
110
111
112
113 process (x_pixel, y_pixel)
114 begin
115   if (x_pixel = "1100" or y_pixel = "1100" or x_pixel = "1000" or y_pixel =
    "1000") then
116     pixel <= "00000000";
117   else
118     if (y_pixel = "0000") then -- pixel waarde bepalen ( x_pixel opschalen
    naar 7 bitjes)
119       pixel <= ("000" & x_pixel);
120
121     elsif (y_pixel = "0001") then
122       pixel <= ("000" & x_pixel) + 8;
123
124     elsif (y_pixel = "0010") then
125       pixel <= ("000" & x_pixel) + 16;
126

```

```

127     elsif (y_pixel = "0011") then
128         pixel <= ("000" & x_pixel) + 24;
129
130     elsif (y_pixel = "0100") then
131         pixel <= ("000" & x_pixel) + 32;
132
133     elsif (y_pixel = "0101") then
134         pixel <= ("000" & x_pixel) + 40;
135
136     elsif (y_pixel = "0110") then
137         pixel <= ("000" & x_pixel) + 48;
138
139     elsif (y_pixel = "0111") then
140         pixel <= ("000" & x_pixel) + 56;
141
142     else
143         pixel <= ("000" & x_pixel); -- buiten speelveld ( pixel wordt verder
                                   niet gebruikt)
144
145     end if;
146 end if;
147
148 end process;
149
150
151 process (pixel, x_pixel, y_pixel)
152 begin
153
154     if (x_pixel = "1100" or y_pixel = "1100") then -- raster
155         rgb_game <= "00";
156     elsif (x_pixel = "1000" or y_pixel = "1000") then
157         rgb_game <= "00";
158     else --binnen speelveld
159         rgb_game(0) <= life_in(to_integer(pixel)); --life pixel
160         if(enable_cursor = '1') then
161             if (('0' & cursor) = std_logic_vector(pixel)) then --cursor
162                 rgb_game(1) <= '1';
163             else
164                 rgb_game(1) <= '0';
165             end if;
166         else
167             rgb_game(1) <= '0'; -- enable cursor = 0
168         end if;
169     end if;
170
171 end process;
172 end behaviour;

```

./vhdl/vga\_game.vhd

#### B.4.4 VGA\_text

```

1  -----
2  -- VHDL beschrijving van de vga textentitij die de tekst 'Game of Life'
   doorstuurt naar het scherm
3  -- Door Bas Generowicz & Lisa Audenaert
4  -- December 2012, EE2821 Project "maak een chip" (2012-2013)
5  -----
6

```

```

7
8 library IEEE;
9 use IEEE.std_logic_1164.ALL;
10 use IEEE.numeric_std.ALL;
11
12
13 entity vga_text is
14     port (
15         h_teller      :in  std_logic_vector(7 downto 0);
16         v_teller      :in  std_logic_vector(9 downto 0);
17         rgb_text      :out std_logic
18     );
19 end vga_text;
20
21 architecture behaviour of vga_text is
22
23     constant offset_y : integer := 30;
24
25
26     constant v1      : std_logic_vector(159 downto 0) := "
27         0000111111111100111111111100001100000000001100000000000001111111111000011111111110000
28         ";
29     constant v2      : std_logic_vector(159 downto 0) := "
30         0000111111111100111111111100001100000000001100000000000001111111111000011111111110000
31         ";
32     constant v3      : std_logic_vector(159 downto 0) := "
33         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
34         ";
35     constant v4      : std_logic_vector(159 downto 0) := "
36         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
37         ";
38     constant v5      : std_logic_vector(159 downto 0) := "
39         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
40         ";
41     constant v6      : std_logic_vector(159 downto 0) := "
42         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
43         ";
44     constant v7      : std_logic_vector(159 downto 0) := "
45         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
46         ";
47     constant v8      : std_logic_vector(159 downto 0) := "
48         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
49         ";
50     constant v9      : std_logic_vector(159 downto 0) := "
51         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
52         ";
53     constant v10     : std_logic_vector(159 downto 0) := "
54         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
55         ";
56     constant v11     : std_logic_vector(159 downto 0) := "
57         0000000000001100000000001100001100000000001100000000000000000000011000011000000110000
58         ";
59     constant v12     : std_logic_vector(159 downto 0) := "
60         0000001111111110000111111110000110000000000110000000000000001111111000011000000110000
61         ";
62     constant v13     : std_logic_vector(159 downto 0) := "
63         0000001111111110000111111110000110000000000110000000000000001111111000011000000110000
64         ";

```

```

39  constant v14          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
40  constant v15          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
41  constant v16          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
42  constant v17          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
43  constant v18          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
44  constant v19          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
45  constant v20          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
46  constant v21          : std_logic_vector(159 downto 0) := "
    00000000000001100000000000110000110000000000011000000000000000000000000110000110000000110000
    ";
47  constant v22          : std_logic_vector(159 downto 0) := "
    0000111111111100000000000110000110000111111110000000000000000000000000011000011111111110000
    ";
48  constant v23          : std_logic_vector(159 downto 0) := "
    0000111111111100000000000110000110000111111110000000000000000000000000011000011111111110000
    ";
49  -----
50  begin
51
52  process (h_teller, v_teller)
53      variable h_tel: unsigned (7 downto 0);
54      variable v_tel: unsigned (9 downto 0);
55  begin
56
57      h_tel := unsigned(h_teller);
58      v_tel := unsigned(v_teller);
59
60      if (h_tel < 160) then
61          if (v_tel = offset_y or v_tel = (offset_y + 1)) then
62              rgb_text <= v1(to_integer(h_tel));
63          elsif (v_tel = (offset_y + 2) or v_tel = (offset_y + 3)) then
64              rgb_text <= v2(to_integer(h_tel));
65          elsif (v_tel = (offset_y + 4) or v_tel = (offset_y + 5)) then
66              rgb_text <= v3(to_integer(h_tel));
67          elsif (v_tel = (offset_y + 6) or v_tel = (offset_y + 7)) then
68              rgb_text <= v4(to_integer(h_tel));
69          elsif (v_tel = (offset_y + 8) or v_tel = (offset_y + 9)) then
70              rgb_text <= v5(to_integer(h_tel));
71          elsif (v_tel = (offset_y + 10) or v_tel = (offset_y + 11)) then
72              rgb_text <= v6(to_integer(h_tel));
73          elsif (v_tel = (offset_y + 12) or v_tel = (offset_y + 13)) then
74              rgb_text <= v7(to_integer(h_tel));
75          elsif (v_tel = (offset_y + 14) or v_tel = (offset_y + 15)) then
76              rgb_text <= v8(to_integer(h_tel));

```

```

77     elsif (v_tel = (offset_y + 16) or v_tel = (offset_y + 17)) then
78         rgb_text <= v9(to_integer(h_tel));
79     elsif (v_tel = (offset_y + 18) or v_tel = (offset_y + 19)) then
80         rgb_text <= v10(to_integer(h_tel));
81     elsif (v_tel = (offset_y + 20) or v_tel = (offset_y + 21)) then
82         rgb_text <= v11(to_integer(h_tel));
83     elsif (v_tel = (offset_y + 22) or v_tel = (offset_y + 23)) then
84         rgb_text <= v12(to_integer(h_tel));
85     elsif (v_tel = (offset_y + 24) or v_tel = (offset_y + 25)) then
86         rgb_text <= v13(to_integer(h_tel));
87     elsif (v_tel = (offset_y + 26) or v_tel = (offset_y + 27)) then
88         rgb_text <= v14(to_integer(h_tel));
89     elsif (v_tel = (offset_y + 28) or v_tel = (offset_y + 29)) then
90         rgb_text <= v15(to_integer(h_tel));
91     elsif (v_tel = (offset_y + 30) or v_tel = (offset_y + 31)) then
92         rgb_text <= v16(to_integer(h_tel));
93     elsif (v_tel = (offset_y + 32) or v_tel = (offset_y + 33)) then
94         rgb_text <= v17(to_integer(h_tel));
95     elsif (v_tel = (offset_y + 34) or v_tel = (offset_y + 35)) then
96         rgb_text <= v18(to_integer(h_tel));
97     elsif (v_tel = (offset_y + 36) or v_tel = (offset_y + 37)) then
98         rgb_text <= v19(to_integer(h_tel));
99     elsif (v_tel = (offset_y + 38) or v_tel = (offset_y + 39)) then
100         rgb_text <= v20(to_integer(h_tel));
101     elsif (v_tel = (offset_y + 40) or v_tel = (offset_y + 41)) then
102         rgb_text <= v21(to_integer(h_tel));
103     elsif (v_tel = (offset_y + 42) or v_tel = (offset_y + 43)) then
104         rgb_text <= v22(to_integer(h_tel));
105     elsif (v_tel = (offset_y + 44) or v_tel = (offset_y + 45)) then
106         rgb_text <= v23(to_integer(h_tel));
107     else
108         rgb_text <= '0';
109     end if;
110 else
111     rgb_text <= '0';
112 end if;
113
114 end process;
115
116
117 end behaviour;

```

./vhdl/vga\_text.vhd

## B.5 VGA-timer

```

1  -----
2  -- VHDL Beschrijving voor de VGA controller van Conway's Game of Life
3  -- November 2012, EE2821 Project "maak een chip" (2012-2013)
4  -- Laatste aangepast op 19 November, door Ahmet Gercekcioglu en Gert-Jan
   van Raamsdonk
5  -----
6  library IEEE;
7  use IEEE.std_logic_1164.ALL;
8  use IEEE.std_logic_unsigned.ALL;
9
10 entity vga is
11 port(

```

```

12 clk,      : in std_logic;
13 reset     : in std_logic;
14 r_in      : in std_logic;
15 g_in      : in std_logic;
16 b_in      : in std_logic;
17 hsync     : out std_logic;
18 vsync     : out std_logic;
19 r_out     : out std_logic;
20 g_out     : out std_logic;
21 b_out     : out std_logic;
22 row       : out std_logic_vector(9 downto 0);
23 column    : out std_logic_vector(7 downto 0));
24 end vga;
25
26
27 architecture behaviour of vga is
28
29 signal videoon, videov, videoh : std_logic;
30 signal hcount : std_logic_vector(7 downto 0);
31 signal vcount : std_logic_vector(9 downto 0);
32
33 begin
34 hcounter: process (clk, reset)
35 begin
36     if reset='1' then
37         hcount <= (others => '0');
38     else if (clk'event and clk='1') then
39         if hcount=199 then --eind van het horizontale beeld
40             hcount <= (others => '0');
41         else
42             hcount <= hcount + 1;
43         end if;
44     end if;
45 end if;
46 end process;
47
48
49 vcounter: process (clk, reset)
50 begin
51     if reset='1' then
52         vcount <= (others => '0');
53     else if (clk'event and clk='1') then
54         if hcount=199 then
55             if vcount = 524 then --eind van het gehele beeld
56                 vcount <= (others => '0');
57             else
58                 vcount <= vcount + 1;
59             end if;
60         end if;
61     end if;
62 end if;
63 end process;
64
65
66     end if;
67 end process;
68 sync: process (clk, reset)
69 begin

```

```

70     if reset='1' then
71         hsync <= '0';
72         vsync <= '0';
73     else if (clk'event and clk='1') then
74         if (hcount<=187 and hcount>=164) then    --volgende rij, 31.5kHz
75             hsync <= '0';
76         else
77             hsync <= '1';
78         end if;
79         if (vcount<=492 and vcount>=491) then    --volgend beeld, 60Hz
80             vsync <= '0';
81         else
82             vsync <= '1';
83         end if;
84     end if;
85 end if;
86 end process;
87
88 process (hcount)
89 begin
90     column <= hcount;
91     if hcount <= 159 then    --resolutie horizontaal, eind rechts
92         videoh <= '1';
93     else
94         videoh <= '0';
95     end if;
96 end process;
97
98 process (vcount)
99 begin
100     row <= vcount;
101     if vcount <= 479 then    --resolutie verticaal, eind linksonder
102         videov <= '1';
103     else
104         videov <= '0';
105
106     videoon <= videoh and videov;    --beeldscherm wat te zien is
107
108 colors: process (clk, reset)
109 begin
110     if reset='1' then
111         r_out <= '0';
112         g_out <= '0';
113         b_out <= '0';
114     elsif (clk'event and clk='1') then
115         r_out <= r_in and videoon;
116         g_out <= g_in and videoon;
117         b_out <= b_in and videoon;
118     end if;
119
120 end process;
121
122 end behaviour;

```

./vhdl/vga-compleet.vhd