

# Final report

Projectgroep A5

14-01-2015

# 1

## SAMENVATTING

Dit ontwerprapport is geschreven naar aanleiding van het project EPO-3 van het tweede jaar van de bacheloropleiding Electrical Engineering. In dit project staat het beschrijven van hardware centraal, met als doel het ontwerpen van een chip. In de keuze van een ontwerp zijn de studenten door de TU geheel vrij gelaten, er moet natuurlijk wel rekening worden gehouden met de grootte van de chip en het tijdsbestek waarin het ontwerp gerealiseerd kan worden. Er is hierdoor de EPO-3 groep A5 gekozen voor het ontwerpen van een spelcomputer, hierop kunnen meerdere spellen gespeeld worden. Om binnen het tijdsbestek te blijven is gekozen voor 'Pong', meerdere spellen kunnen eventueel achteraf simpel toegevoegd worden.

Van het uiteindelijke ontwerp wordt in de laatste periode van dit jaar een chip gemaakt, tijdens het ontwerpproces is er daarom met een FPGA development board gewerkt. Deze is geleverd door de TU Delft. Het grootste deel van het systeem bevindt zich op de chip zelf, toch zijn er nog drie losse componenten gebruikt om dit ontwerp te realiseren. Het idee achter de spelcomputer is dat hij meerdere spellen kan spelen, deze moeten extern aangeleverd worden. Er is hier gekozen voor een SD-kaart uitlezing met SPI, het spel is in assembly geschreven. Ook de besturing moet extern geregeld worden, met ultrasone sensoren bleek dit systeem het best te controleren. Als back-up plan is er hiernaast voor gekozen om ook enkele knoppen te implementeren. Om het spel te kunnen spelen is er tot slot ook een beeldscherm nodig.

Het belangrijkste deel van dit ontwerp bevindt zich op de chip. In de CPU wordt de informatie verwerkt die via SPI van de SD-kaart binnenkomt. In de CPU wordt daarnaast ook de besturing van het spel verwerkt. De CPU kan dus gezien worden als het brein van de chip, alles wordt hier verwerkt en doorgegeven aan de GPU. Deze maakt er vervolgens een beeld van. Om vertraging op het scherm te voorkomen is er een 'Black-box' geïmplementeerd, die zorgt ervoor dat er gewisseld kan worden tussen twee SRAM geheugens. Hierdoor kan er op de ene SRAM geschreven worden door de GPU en kan de ander worden uit gelezen door de VGA. De VGA produceert vervolgens het beeld op een beeldscherm. Alleen beide SRAM geheugens bevinden zich niet op de chip, maar zitten extern in het systeem verwerkt.

Tot in de laatste week is er hard gewerkt om alles op tijd werkend te krijgen, dit is helaas niet gelukt. Er is daarom besloten met de werkende onderdelen verder te gaan, van de ROM, CPU en de SPI decoder is een simpele rekenmachine ontworpen. De besturing loopt nog steeds via de sensoren in combinatie met enkele drukknoppen. De chip is nu in staat om enkele simpele berekeningen te doen, namelijk optellen, aftrekken en vermenigvuldigen.

# **INHOUDSOPGAVE**

# 2

## INLEIDING

EPO-3 is het derde project uit de propedeuse van de bachelor opleiding Electrical Engineering aan de TU delft. Het doel van dit project is het ontwerpen van een chip, deze chip kan bijvoorbeeld een spelletje bevatten of als besturingsdoel dienen. Als project groep A5 is er gekozen voor het bouwen van een spelcomputer, hierop kunnen meerdere games gespeeld worden. De verschillende games moeten van een SD-kaart worden geladen voordat ze gespeeld kunnen worden. Het eerste spel dat ontworpen wordt is Pong, mocht er aan het eind van dit project nog tijd overblijven zullen er meerdere spellen volgen. De besturing loopt via ultrasone sensoren die de locatie van iemands hand bepalen. Het ontwerpen van een simpele spelcomputer sluit nauw aan met de bachelor opleiding Electrical Engineering. De toekomst van de elektrotechniek ligt dan ook in dit werkgebied. Chips zijn in de loop der jaren de basis gaan vormen voor bijna elk systeem. In dit ontwerprapport zal eerst het doel van dit project verder toe worden gelicht. Vervolgens zal er na het geven van het systeemoverzicht, wordt ieder individueel onderdeel beschreven. Testresultaten en codes zullen voornamelijk terug te vinden zijn in de appendices, tenzij anders aangegeven.

# 3

## PROBLEEMSTELLING

### 3.1. PROJECTOPGAVE: 'ONTWERP EEN CHIP'

De opdracht meegegeven door de TU is het ontwerpen van een chip. Hiermee is de gehele opdracht eigenlijk al beschreven, de studenten mogen verder in alle vrijheid de opdracht verder bepalen. Er wordt natuurlijk wel eerst grondig gecontroleerd door de tutoren of de opdrachten aan de eisen voldoen en haalbaar zijn. Naast het ontwerpen van de chip mogen er extern onderdelen worden toegevoegd, bijvoorbeeld een beeldscherm of onderdelen voor de besturing.

Als project groep A5 is er gekozen om van de chip een spelcomputer te maken, hierop moeten meerdere spellen gespeeld kunnen worden. Als eerst moet het spelletje 'Pong' te spelen zijn op deze spelcomputer. Aan het einde van het project kunnen eventueel andere spellen ontworpen worden. Extern moet er gezorgd worden voor besturing, weergave en opslag van de spellen. Dit laatste wordt door middel van een SD-kaart gerealiseerd. De besturing vind plaats via ultrasone sensoren of mocht dit niet naar behoren werken door middel van drukknoppen.

### 3.2. BESCHIKBARE INFRASTRUCTUUR

In het 4e semester wordt de chip geproduceerd en getest, deze chip wordt door twee projectgroepen gebruikt. Elke groep heeft dus een halve chip voor zijn ontwerp, dit zijn ongeveer 200-250 flipflops. Daarnaast kan iedere groep wel gebruik maken van alle aansluitpinnen op de chip, dit zijn er 32. Voor het testen tussen door is er hardware beschikbaar, in de vorm van een Altera FPGA bord. Op dit Altera bord is veel randapparatuur aanwezig, daarnaast kunnen er nog andere externe onderdelen aan dit bord worden gekoppeld.

### 3.3. FUNCTIONELE EISEN

Als functionele eisen kan er kortom gezegd worden dat er op de chip meerdere spellen gespeeld kunnen worden. Deze spellen moeten bestuurd kunnen worden door middel van ultrasone sensoren. Om te beginnen wordt 'Pong' ontworpen, andere spellen volgen als de projecttijd dit toelaat.

### 3.4. RANDVOORWAARDEN

Aan het eindresultaat zitten een aantal randvoorwaarden verbonden. De externe onderdelen mogen maximaal 30 á 40 euro kosten, mochten deze niet vorhanden zijn. Er moet rekening worden gehouden met de grote van de chip. Tot slot dient aan alle tijdschema's te worden voldaan.

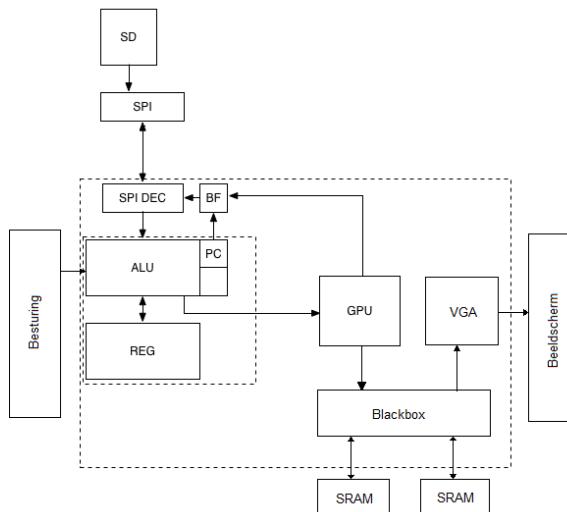
### 3.5. PLAN VAN AANPAK

Het plan van aanpak is een beschrijving van de aanpak van dit project, hierin staan onder andere de doelstellingen en wegen daar naar toe beschreven. Het plan van aanpak is te vinden in appendix A.

# 4

## SYSTEEMOVERZICHT

De spelcomputer ontworpen en gebouwd door EPO-3 groep A5 bestaat uit veel verschillende losse onderdelen. Om te voorkomen dat we het overzicht kwijt raken, wordt er in dit hoofdstuk het totale systeem beschreven. Er zal dus vooral worden gekeken naar de toplevel beschrijving, hierin worden alle onderdelen samengevoegd en gekoppeld tot een geheel. Ook wordt er in het kort even ingegaan op de externe componenten die ons systeem bevat.



Figuur 4.1: Totale systeem weergave

### 4.1. TOPLEVEL

De toplevel beschrijving bestaat uit een entity en een behaviour. In de code van de entity staan alle poorten beschreven die de chip verbinden met de externe componenten die het systeem bevat. In de behaviour worden alle interne componenten aan elkaar verbonden. In figuur ?? staat schematisch weergegeven op welke manier dit gebeurd. De losse onderdelen van dit systeem worden individueel beschreven in dit verslag.

### 4.2. EXTERNE COMPONENTEN

Het systeem dat ontworpen is door EPO-3 groep A5 bevat kort gezegd drie aparte externe componenten, twee daarvan zijn inputs en een is een output. De inputs van dit systeem zijn de besturing en de opslag van het spel. Dit laatste gebeurd door middel van SD uitlezing met SPI. De output van dit systeem wordt aangestuurd door de VGA en kan een beeld produceren op een willekeurig beeldscherm.

# 5

## BESTURING

Ons systeem wordt gespeeld doormiddel van 2 soorten besturing. De Ultrasone en de Button. De speler kan de modus selecteren doormiddel van een switch. Deze data van de buttons en de ultrasone worden verwerkt door een Arduino. Er is gekozen voor deze optie omdat de Arduino via SPI werkt. Hierdoor kan de SPI code getest worden en kunnen we het aantal pinnen dat gebruikt wordt laag houden.

### 5.1. ULTRASONE

Een unieke uitdaging van ons project is de Ultrasone besturing, de besturing werkt door middel van een (ultrasone)sensor aan de rechterzijde van de speler. Deze sensor meet met een speel ruimte van 75cm elke 11 milliseconde. Het aansturen van de ultrasone sensoren gaat doormiddel van een 2ms lange pulse op de IN-pin. Dit is de trigger van de SRF-04(de door ons gekozen ultrasone sensor). Hierna verzend de sensor zijn pulse, op dat moment wordt de OUT-pin ook hoog. Deze blijft hoog tot de gereflecteerde pulse weer binnen komt. Deze tijd wordt gedeeld door 2 omdat het geluid zowel de afstand heen als terug moet afleggen. Deze tijd wordt vervolgens geschaald naar afstand door hem door 29m/s(de snelheid van geluid in lucht) te delen. Hierna wordt de tijd teruggemapt(alles tussen de 0 en de 75 wordt terug geschaald naar 0 en 12). Deze waarde wordt voor player 2 4 bits geschoven naar links. En vervolgens wordt het signaal via de hierboven genoemde SPI verstuurd naar de chip.

### 5.2. BUTTONS

Bij de buttons wordt een andere manier van werken gehanteerd, hier wordt de waarde van de plaatsvector onthouden(als integer). En naar gelang welke button geactiveerd wordt, wordt het signaal 1 verhoogd/verlaagd. Dit getal kan maximaal 12 bereiken en minimaal 0. Vervolgens wordt dit signaal voor player 2 ook verschoven, en daarna verzonden via SPI.

De getallen 0->12 voor player 1/2 zijn in gebruik, dit geeft ruimte om 13,14,15 te gebruiken voor andere doeleinde. 13 dient als getal om de Start van systeem aan te geven. 15 is de Reset van het systeem.

### 5.3. SPI-COMMUNICATIE

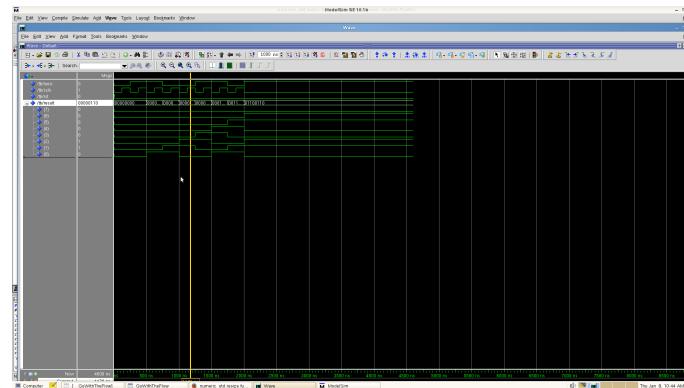
Alle variable die berekent worden in de arduino worden via SPI verzonden. SPI is een protocol over het verzenden van de data. Dit werkt doormiddel van 2 draden. Draad 1 verzend een klok van 8 pulse lang en een default low. Draad 2 verzend de waarde van 8 bits lang.

### 5.4. DETECTIE SPI

De chip detecteert op de opgaande flank van draad 1(klok) de waarde van draad 2(signaal). Deze waarde wordt in een Flip-flop(geheugen) gezet. Op de neergaande flank wordt de waarde geshift. Dit moet om te zorgen dat de data in de goede volgorde aankomt bij de CPU.

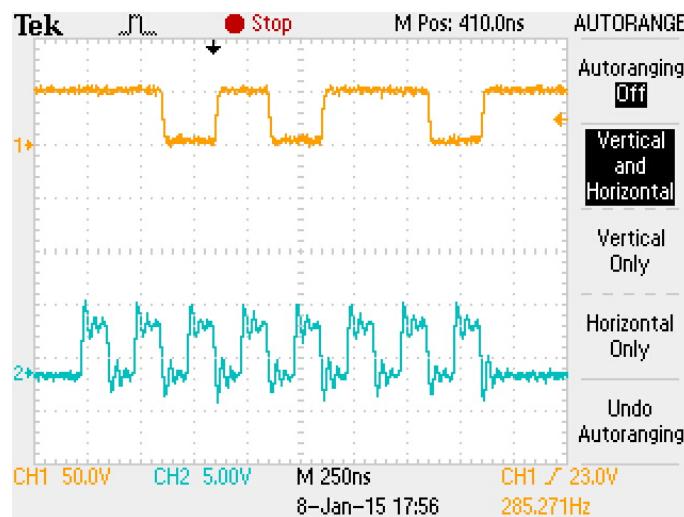
## 5.5. SIMULATIE

Na de ontwikkeling via Go-with-the-flow is het systeem getest. Hierbij is gebruik gemaakt van een dubbele 6 als positie. Het resultaat ziet u in:



Figuur 5.1: Twee maal de positie 6 gemeten met Modelsim

Vervolgens is het systeem gedownload op de FPGA, hierop is de arduino aangesloten. Het signaal van de arduino was:



Figuur 5.2: De positie 13(start) en 6

Deze waarde kwam precies zoals verwacht op de ledjes op de FPGA. Echter na ongeveer 5 seconde begon de waarde te shiften. Dit kwam door dat beide systemen geen gecombineerde aarde had. Dit is opgelost door een draad tussen beide aarde te zetten.

# 6

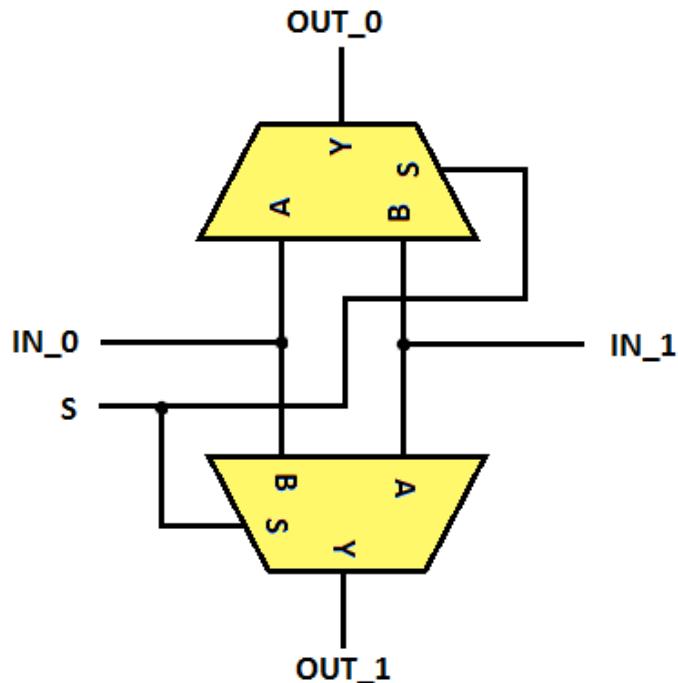
## BLACKBOX

### 6.1. FUNCTIONELE BESCHRIJVING

Voor het genereren van de beelden worden twee SRAM chips gebruikt. De ene chip wordt uitgelezen door de VGA generator en er wordt naar de andere chip geschreven door de GPU. Als een frame geschreven is naar de SRAM chip wordt de chip waar net naar geschreven is verbonden met de VGA generator zodat deze uitgelezen kan worden. De chip die voorheen verbonden was met de VGA wordt doorverbonden met de GPU zodat daar vervolgens naar geschreven kan worden. Door deze techniek toe te passen blijft het beeld op het scherm stabiel. De BlackBox is de schakeling die dat mogelijk maakt.

### 6.2. INPUTS EN OUTPUTS

De SRAM chips, de VGA en de GPU communiceren met elkaar met behulp van SPI. De schakeling heeft dus twee SPI master aansluitingen voor VGA en de GPU en twee SPI slave aansluitingen voor de twee SRAM chips. Een SPI aansluiting bestaat uit vier signaallijnen. Dit zijn de serial clock (SCLK), master in slave out (MISO), master out slave in (MOSI) en slave select (SS). Ook heeft de schakeling nog een ingang om te selecteren welke SRAM chip verbonden is met de VGA generator en welke met de GPU.



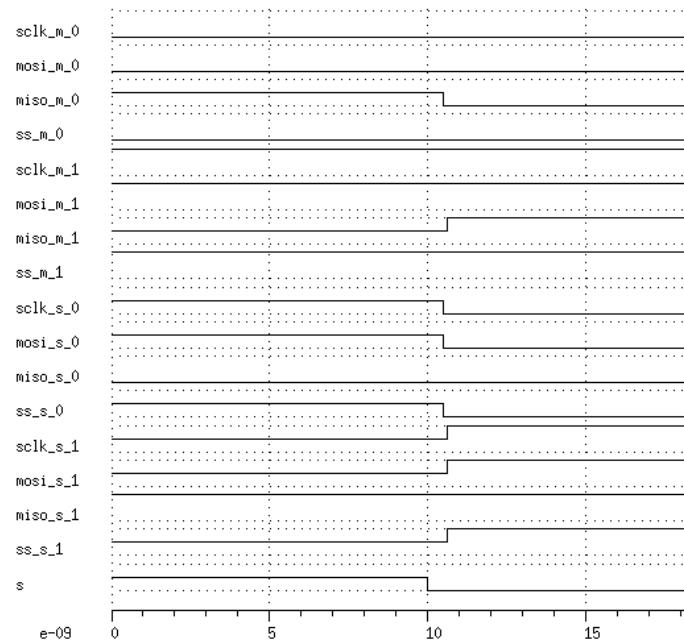
Figuur 6.1: Subschakeling van de blackbox

### 6.3. IMPLEMENTATIE

De schakeling bestaat uit vier subschakelingen. De subschakeling bestaat uit twee tweo naar 1 lijn multiplexers zoals weergegeven in figuur ???. Door de S laag te maken, wordt IN\_0 verbonden met OUT\_0 en IN\_1 met OUT\_1. Als S hoog is, is IN\_0 verbonden met OUT\_1 en vice versa. Vervolgens worden de SCLK lijnen van de GPU en VGA generator verbonden met de ingangen van de subschakeling en de uitgangen met de SCLK lijnen van de twee SRAM chips. Hetzelfde gebeurt ook voor de MOSI en de SS lijnen van de masters en de slaves. Bij de MISO lijnen is het omgekeerd omdat dit een inputlijn is voor de masters en een outputlijn voor de slaves. Hier worden dus de SRAM chips verbonden met de ingangen en de VGA generator en de GPU aan de uitgangen. De BlackBox bestaat nu uit vier subcircuits. Door de select (S) ingangen van de subschakelingen met elkaar te verbinden is de BlackBox nu compleet.

### 6.4. SIMULATIE

Om de werking van de schakeling te verifiëren is een switch-level simulatie gedaan zoals te zien is in figuur ??.



Figuur 6.2: Switch-Level Simulation van de blackbox

Er valt te zien dat voordat het signaal s hoog wordt de SPI signalen van master 0 en slave 0 overeenkomen. Ook de spi signalen van master 1 en slave 1 komen overeen. Nadat s laag wordt zien we dat de signalen van master 0 nu overeenkomen met de signalen van slave 1 en vice versa. Hieruit kan geconcludeerd worden dat de schakeling naar behoren zou moeten werken wanneer deze geïmplementeerd is in de chip. De schakeling kan getest worden door de signalen op de bijbehorende pinnen te bekijken en deze te vergelijken met de verwachte signalen van de SPI van de GPU en VGA circuits.

# 7

## CPU

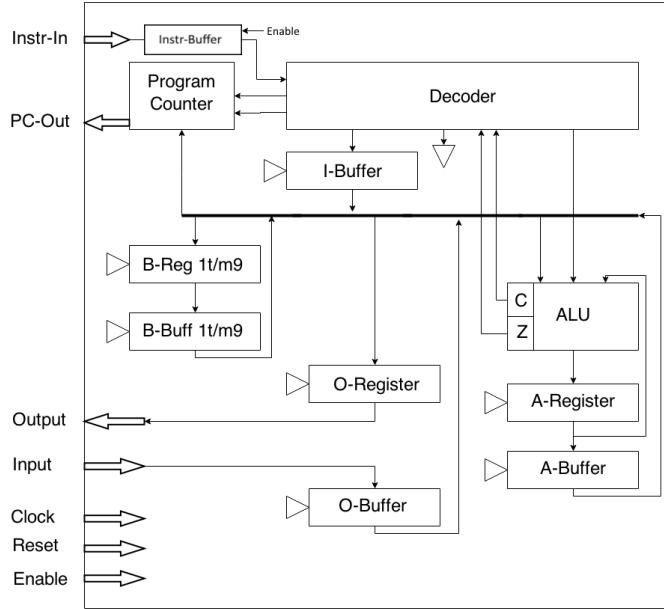
### 7.1. FUNCTIONELE BESCHRIJVING

Om programma's uit te kunnen voeren wordt gebruik gemaakt van een door de Delta-1 [INSERT BRON HIER] geïnspireerde microprocessor. Deze 8-bits processor krijgt instructies binnen die via SPI van een SD-kaart worden afgelezen. Vervolgens voert de processor bewerkingen uit, waarvan de resultaten doorgestuurd worden naar de overige componenten op de chip. De data die uit de processor komt wordt hierna door de VGA-controller omgezet naar beelden op het scherm.

### 7.2. INPUTS EN OUTPUTS

De processor heeft in totaal vijf ingangen. Dit zijn de 12-bits instructie-bus waarop de instructies staan die vanaf de SD kaart worden afgelezen, een 8-bits signaal waar alle externe inputs op komen te staan, de reset en het CPU-enable signaal. Het CPU-enable signaal geeft door aan de processor dat een nieuwe instructie klaar staat om uitgevoerd te worden. Op het 8-bits signaal waar alle inputsinformatie op staat, staat bijvoorbeeld welke knoppen ingedrukt zijn bij de besturing. We hebben besloten om de CPU niet te laten draaien op een kloksignaal, maar op het enable signaal. Op deze manier voert de CPU iedere keer dat er een instructie op de ingang wordt aangeboden één enkele instructie uit. Zo gaat de CPU geen ongewenste stappen ondernemen met verkeerde/verouderde ingangswaarden.

Verder beschikt de CPU over twee uitgangen: Eén 8-bits signaal dat data naar de VGA controller stuurt en een 8-bits signaal dat het adres van de volgende instructie doorgeeft aan de SD-kaart uitlezer.



Figuur 7.1: Toplevel beschrijving

### 7.3. IMPLEMENTATIE

De totale processor is opgebouwd uit vijf verschillende sub-schakelingen: Registers, buffers, een rekenkern, een program-counter en een instructie-decoder.

De processor beschikt in totaal over elf registers. Wanneer het enable-singaal vanaf de decoder van een register hoog is, slaat deze de waarde die op dat moment aan zijn ingang staat op en kan deze data bij een latere bewerking weer uitgelezen worden. Hiervan worden er negen gebruikt om waarden op te slaan voor volgende bewerkingen, één om data aan de data-uitgang van de processor te zetten en één aan de uitgang van de rekenkern om direct een volgende bewerking op uit te kunnen voeren. Het enable-singaal staat in een 5-bits vector, waarmee ook de registers waarin je data wilt opslaan geselecteerd worden. De most-significant bit bepaald of de register enabled is. De overige vier bits is het adres van het register dat je wilt selecteren.

Verder zijn er dertien buffers aanwezig. Deze buffers laten data van hun ingang door naar hun uitgang wanneer hun enable-singaal vanaf de decoder hoog is, en geven 'high-Z' aan de output door wanneer dit niet zo is. Ze maken dus data beschikbaar aan de bus, zodat er verder mee gerekend kan worden. Negen van deze 8-bits buffers staan tussen de uitgangen van de opslag-registers en de databus, één aan de controller-input, één tussen het rekenkern-register en de databus, één die vanuit de decoder een numerieke waarde op de databus kan zetten en één 12-bits buffer aan de instructie-ingang van de CPU. De negen buffers achter de opslag-registers worden op dezelfde manier geselecteerd als de registers, maar dan op hun eigen data-bus.

De Arithmetic Logic Unit (ALU) heeft één 3-bits instructie-ingang vanuit de decoder, twee 8-bits data ingangen waarvan er een van de databus komt en een het resultaat van de vorige bewerking bevat, een 1-bit carry uitgang die aangeeft wanneer er een overflow optreedt bij een optel-operatie, een 1-bit zero uitgang die aangeeft wanneer een 'AND' operatie enkel nullen oplevert en een 8-bits uitgang waar het resultaat van de bewerkingen op staat. Afhankelijk van het instructie-singaal wordt een andere operatie uitgevoerd met de data op de twee ingangen. Op basis van de Opcode selecteert de ALU middels 'when' statements welke functie hij moet uitvoeren.

De Program counter krijgt na elke instructie van de decoder een signaal om ofwel één instructie verder te gaan, of om naar een specifieke instructie te 'springen'. Dit wordt gedaan met behulp van een 1-bit increment en -jump signaal. Zolang de increment ingang hoog is wordt er telkens één opgeteld bij iedere instructie, en als het jump signaal hoog is wordt het nieuwe instructie-nummer vanaf de databus uitgelezen. Het 8-bits uitgangssignaal bevat het adres van de volgende instructie en wordt doorgegeven naar de SD-kaart uitlezer. De program counter zal bij elk opgaand enable signaal een stap verder gaan, dus bij iedere nieuwe instructie wordt gekeken wat de volgende instructie gaat zijn.

Aan het hoofd van de processor bevindt zich de instructie decoder. Deze decoder stuurt naar aanleiding van een 12-bits instructie de overige componenten binnen de CPU aan. Dit wordt gedaan door eerst het 12-bits signaal op te delen in een 4-bits instructie en een 8-bits argument. Hierna wordt afhankelijk van de

instructie een enable-singaal naar een relevant register of buffer, een instructie-code naar de rekenkern en een increment of jump signaal naar de program counter gestuurd. Het 8-bits argument kan ofwel als waarde op de databus gezet worden of als enable-signaal omgezet worden voor een register of buffer. De mogelijke instructies met bijbehorende uitgangssignalen van de decoder zijn te zien in tabel [figuur ??]. In de tabel is te zien bij welke ingangssignalen de output hoog moet zijn. Met dit in gedachten kan door middel van logische combinatoriek voor iedere uitgang een logische functie worden gedefinieerd.

De volledige layout is te zien in ??, en de mogelijke instructies met bijhorende interne signalen zijn te zien in ??

Instructioncode	Instructie	ALU	Prog_cntr_inc	Prog_cntr_ld	In_buff_oe	A_reg_ld	A_buff_oe
0000							
0001	Jp #	xxx	0	1	1	0	0
0010	Jp R <sub>s</sub>	xxx	0	1	0	0	0
0011	Bz	xxx	0 als z	1 als z	1	0	0
0100	Bc	xxx	0 als c	1 als c	1	0	0
0101	Ld #	000	1	0	1	1	0
0110	Ld R <sub>s</sub>	000	1	0	0	1	0
0111	St R <sub>s</sub>	xxx	1	0	0	0	1
1000	ADD #	101	1	0	1	1	0
1001	ADD R <sub>s</sub>	101	1	0	0	1	0
1010	XOR #	001	1	0	1	1	0
1011	XOR R <sub>s</sub>	001	1	0	0	1	0
1100	AND #	010	1	0	1	1	0
1101	AND R <sub>s</sub>	010	1	0	0	1	0
1110	Set c	011	1	0	0	0	0
1111	Clr c	100	1	0	0	0	0

Figuur 7.2: Instructies met bijbehorende uitgangssignalen

# 8

## GPU

### 8.1. DOELSTELLING

De GPU dient als vertaalslag tussen de 8-bit commando's welke worden ontvangen vanuit de output buffer van de CPU, en de pixelmap welke opgeslagen is in het SRAM. Door het gebruik van 2 wisselende SRAM chips kan de GPU in de ene SRAM chip schrijven, terwijl de VGA-module uit de andere SRAM chip aan het lezen is. De GPU ontvangt vanuit de CPU achtereenvolgens commando's met hierin data van de X en Y posities van de in te kleuren pixel. Tevens is er een commando aanwezig om het volledige scherm leeg te vegen.

### 8.2. I/O

De I/O van de GPU zal bestaan uit de communicatie met de CPU, SPI uitgang naar het SRAM en klok- en resetsignalen ten behoeve van het realiseren van de FSM's.

#### 8.2.1. INPUT CPU

De ingang van de CPU zal bestaan achtereenvolgens 8 bit commando's welke zullen worden opgesplitst in een 2 bit commando, en een 6 bit datavector. Zie onderstaande tabel.

7	6	5	4	3	2	1	0
instruction(1)	instruction(0)	data(5)	data(4)	data(3)	data(2)	data(1)	data(0)

De instruciteset van de GPU worst als volgt gedefinieerd:

**"01"** Read X: Deze instructie beschrijft dat de CPU in de databits de x-coördinaat van de in te kleuren pixel weergeeft.

**"10"** Read Y: Deze instructie beschrijft dat de CPU in de databits de y-coördinaat weergeeft van de in te kleuren pixel.

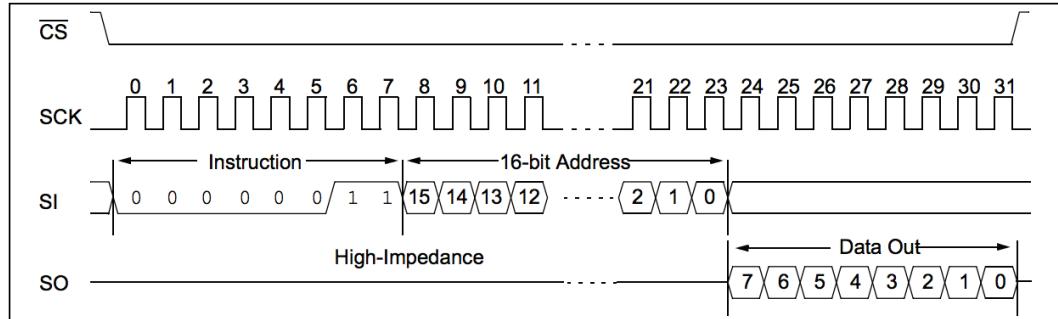
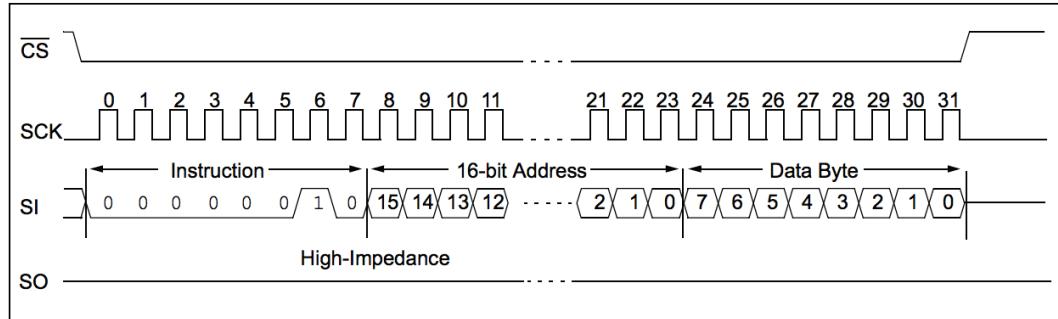
**"11"** Clear screen: Deze instructie geeft aan dat de volledige pixelmap in het SRAM leeggemaakt kan worden.

#### 8.2.2. SPI

De uitgang van de GPU zal bestaan uit SPI signalen om te comminuceren met het SRAM. Gedurende het project is gebleken dat het noodzakelijk blijkt te zijn dat de GPU zowel kan lezen als schrijven vanuit het SRAM. Dit betekend dat er een volledig bidirectionele comminucatie vereist is tussen SRAM en GPU. De comminucatie met het gekozen SRAM zal worden gerealiseerd volgens de door de datasheet van het SRAM opgegeven commando's. Voor de GPU zijn de lees en schrijf commando's van belang. Deze zijn te vinden in figuur ??.

**TABLE 2-1: INSTRUCTION SET**

Instruction Name	Instruction Format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register

**FIGURE 2-1: BYTE READ SEQUENCE****FIGURE 2-2: BYTE WRITE SEQUENCE**

Figuur 8.1: SPI specificatie volgens datasheet SRAM

## 8.3. RANDVOORWAARDEN

Tijdens het ontwerpen van de FSM ten behoeve van de GPU zal het noodzakelijk zijn om deze te laten voldoen aan een aantal gestelde randvoorwaarden met betrekking tot de communicatie met andere componenten.

### 8.3.1. INDELING BEELDSCHERM

De indeling van het uiteindelijke beeldscherm wordt deels bepaald door de GPU, gedurende het ontwerp-process is er uit gegaan van de volgende beeldscherm specificaties.

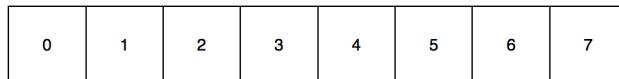
**Pixeldefinitie** De resolutie van het scherm is vastgesteld op 32 bij 24 pixels. Dit betekend dat de maximale ingangswaarde van de databits van de CPU 32 zal zijn, en dit getal prima in de 6 bit vector past.

**Kleuren** Er is tijdens het ontwerp besloten om geen gebruik te maken van kleuren, dus enkel zwart en witte pixels.

**Bits in Bytes** De pixelmap van het beeldscherm zal in het geheugen worden als bytes, waarin elke bit een ingekleurde pixel voorstelt. Bij de logische waarde '1' zal de pixel ingekleurd worden. Zie figuur ?? voor een grafische representatie van de pixelmap in bytes, en figuur ?? een representatie van de indeling van 1 individuele byte. In de pixelmap is te zien dat er exact 1 byte buiten het scherm valt in verband met de back porch van het beeldscherm.

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	64
65	66	67	68	69
70	71	72	73	74
75	76	77	78	79
80	81	82	83	84
85	86	87	88	89
90	91	92	93	94
95	96	97	98	99
100	101	102	103	104
105	106	107	108	109
110	111	112	113	114
115	116	117	118	119

Figuur 8.2: Pixelmap, rood kader is beelscherm



Figuur 8.3: Indeling byte

## 8.4. PROCESSES

De CHDL code van de GPU bestaat uit een drietal processes welke voor de verwerking van de gegevens zorgen.

### 8.4.1. PROCESS 1 : STATE-RENEW

Het state renew process heeft als taak om bij elke opgaande klokflank de nieuwe state toe te wijzen aan het register waar de huidige state in is opgeslagen. Tevens heeft dit proces als doel om te reageren op een actief reset signaal, waarin er voor de nieuwe state een specifieke resetstate wordt toegewezen (State[0]).

### 8.4.2. PROCESS 2 : COUNTER

De counter wordt gerealiseerd door een 11 bits unsigned waarde welke gebruikt kan worden binnen de FSM om bijvoorbeeld het aantal ontvangen bits via SPI te tellen. De teller reageert net zoals voorgaan process op de opgaande klokflank.

### 8.4.3. PROCESS 3 : FSM

De FSM van de GPU doet de daadwerkelijke berekeningen en vertaalslagen. In figuur ?? is een overzicht te vinden van de volledige FSM. Onderstaand zullen de verschillende functieblokken besproken worden.

**Blok A** Blok A is de resetstate van de volledige FSM. In blok A is de FSM gevoelig voor veranderingen op de instructieingang van de CPU. Zo kan de FSM overgaan naar state[13] wanneer het commando voor 'clear screen' gegeven wordt, of naar stat[1] wanneer deze een x-coordinaat ontvangt. Voor het tijdelijk opslaan van x en y waarden van de CPU wordt gebruik gemaakt van 2 8-bit buffers, welke gedurende state[0] worden gereset om te garanderen dat de volgende berekening geen oude resultaten meedraagt.

**Blok B** In blok B wordt het y-coordinaat ontvangen en zal de FSM berekenen in welke byte in het geheugen de pixel weggeschreven zal worden. Tevens wordt hier de instructie klaargemaakt om data vanuit het SRAM te lezen.

**Blok C** Blok C zal met behulp van de counter het sclk signaal genereren, en de lees instructie, met daarin tevens het adres van de te lezen byte verzenden naar het SRAM.

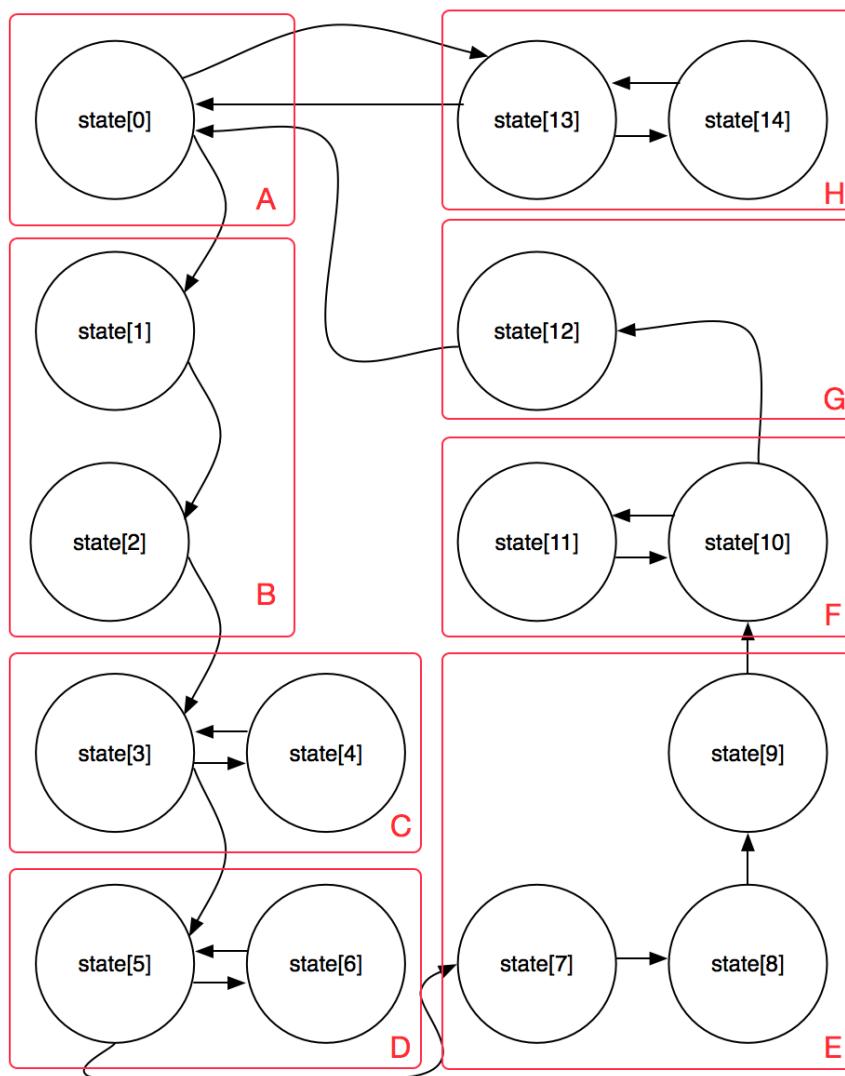
**Blok D** Blok D zal de 8-bit byte van het SRAM ontvangen en deze opslaan in een daarvoor bestemde buffer.

**Blok E** Blok E berekend op welke plaats in de byte een '1' geschreven zal moeten worden en vergelijkt dit met wat er reeds in de ontvangen byte staat. Op deze manier wordt weer een instructie klaargemaakt om de nieuwe data te schrijven naar het SRAM.

**Blok F** Blok F verzendt de nieuwe byte via SPI naar het SRAM. Tevens door gebruik te maken van de counter.

**Blok G** Gedurende de periode waarin SPI aan het schrijven is wordt het slave-select signaal naar '0' getrokken door de FSM. State[12] zet dit signaal weer terug op '1' en zorgt ervoor dat eventuele timers en buffers gereset worden.

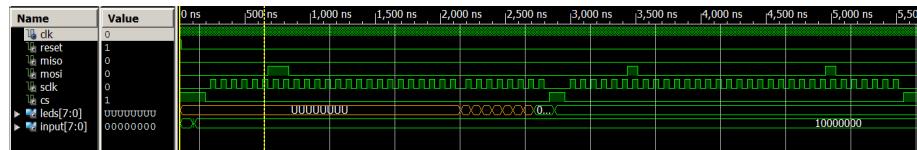
**Blok H** Blok H schrijft naar het SRAM de instructie om te mogen schrijven en daarna enkel '0' om het volledige scherm leeg te maken.



Figuur 8.4: FSM ten behoeve van de GPU

## 8.5. SIMULATIE

Om de volledige gpu te testen is gebruik gemaakt van een testbench welke SPI reactie simuleert. in figuur ?? is ze zien hoe de GPU data ontvangt, vergelijkt en weer wegschrijft naar het SRAM.

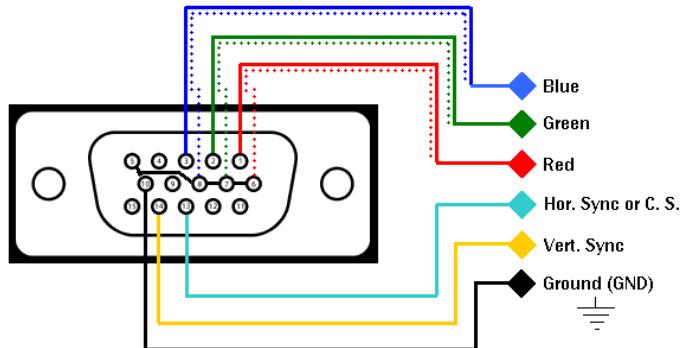


Figuur 8.5: Simulatie GPU

# 9

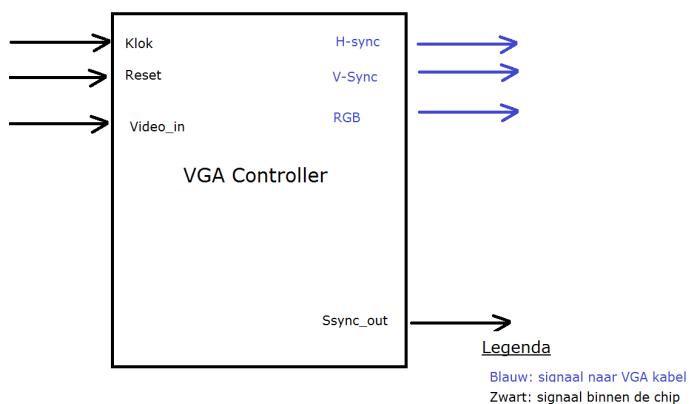
## VGA

De VGA-controller zorgt ervoor dat een beeldscherm van de juiste signalen wordt voorzien om een beeld te kunnen opbouwen. Hierbij wordt een serieel signaal vanuit het SRAM omgezet naar een signaal dat voor een monitor is te begrijpen. Hiervoor zijn vijf signalen nodig. Drie signalen voor respectievelijk rood, groen en blauw, een voor H-Sync en een voor V-Sync.



Figuur 9.1: Layout VGA-stekker

Kortom, de VGA controller zorgt voor een H-sync, een V-Sync en dat het RGB signaal op de juiste manier aan het scherm wordt doorgegeven.



Figuur 9.2: Grafische omschrijving blackbox

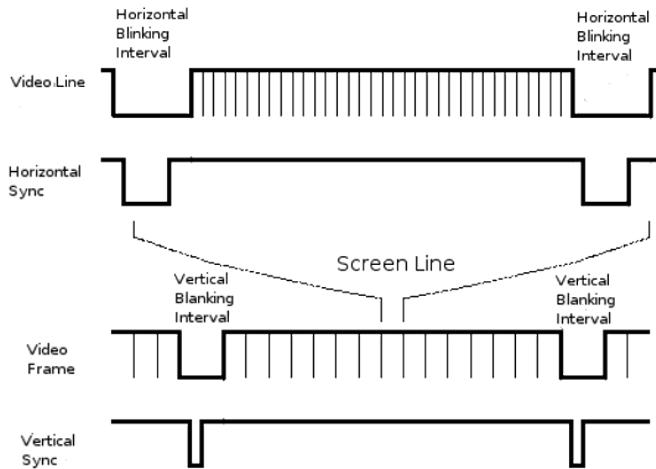
### 9.1. RGB

Op de pinnen voor rood, groen en blauw, staat een analoog signaal tussen de 0 en de 0.7 volt. Hoe hoger de spanning, hoe feller de kleur. Door middel van verschillende spanningsniveaus kan de kleur per pixel worden

bepaald. Wij maken echter alleen gebruik van zwart en wit. Daarom wordt er slechts een signaal naar de alle RGB pinnen verstuurd. Hierdoor ontstaat een witte pixel. Wanneer dit niet gebeurt, is de pixel zwart. Op de FPGA wordt een digitaal signaal omgezet naar een analog signaal tussen de 0 en 0.7 volt. Op de uiteindelijke chip zal er echter nog een schakeling nodig zijn om dit voor elkaar te krijgen.

## 9.2. H-SYNC

H-Sync vertelt de monitor wanneer een regel klaar is. Als H-Sync laag is, betekent dit voor de monitor dat er een nieuwe regel pixels geschreven kan worden. Het is echter niet zo dat wanneer H-Sync weer hoog is, dat er gelijk pixels geschreven kunnen worden. Er zit nogal enige vertragingstijd ingebouwd. Deze tijd was bij CRT-monitoren nodig om elektronenstraal terug naar het begin te richten.



Figuur 9.3

Video Line is het RGB signaal. Zoals te zien is in figuur ??, zit er tijd voor en na de dip van de H-sync. Deze tijd heet de front-porch en de back-porch.

## 9.3. V-SYNC

V-Sync is het signaal dat aan de monitor vertelt dat een volledig scherm is volgeschreven. Als V-Sync laag wordt, begint het scherm weer links bovenin met pixels vullen. Net als bij de H-Sync is er sprake van een Front-porch en een Back-porch. Deze zijn bij de V-Sync een stuk langer dan bij de H-sync. Dit komt omdat de monitor de elektronenstraal van rechts onderin naar links bovenin moet sturen. Deze afstand is een stuk groter dan terug van rechts naar links.

## 9.4. TIMING

De klok van de chip is 6.177 Mhz. Op de FPGA wordt echter een klok gebruikt van 6 Mhz. Deze wordt ook aangehouden in de VHDL code. Om de counters in de VGA-controller niet te groot te maken, wordt de klok eerst gedeeld door vijf. Hierdoor is de nieuwe klok 1.2 Mhz. Wanneer er wordt gesproken van een kloksignaal, wordt er gesproken over de klok van 1.2 Mhz.

### 9.4.1. H-SYNC

De Front-porch van de H-sync is precies één klok slag. Na de Front-porch is de H-sync vijf klokslagen laag. Daarna is de H-sync 34 klokslagen hoog. Het videosignaal naar de monitor is echter iets later pas beschikbaar. Dit komt doordat er nog een Back-porch is van twee klokslagen.

Tijd	Wat	H-sync	Videosignaal
0 tot 1	Front-porch	Hoog	Uit
1 tot 6	H-sync dip	Laag	Uit
6 tot 8	Back-porch	Hoog	Uit
8 tot 40	Video on	Hoog	Aan

Figuur 9.4: Timing voor de H-Sync

#### 9.4.2. V-SYNC

De V-sync werkt eigenlijk op eenzelfde manier als de H-Sync. Hier wordt echter niet de klok geteld, maar het aantal regels. Omdat het aantal regels een stuk minder is dan het aantal klokslagen, wordt de counter voor de V-sync (vidcounter) een stuk kleiner. Iedere keer dat het signaal voor de H-sync weer opnieuw begint, krijgt de vidcounter een signaal. Dit signaal heet nline-out. De Front-porch voor de V-Sync is 11 regels. De V-sync dip is 2 regels en de Back-porch is 31 regels. Vervolgens zijn er 480 regels beschikbaar om een videosignaal uit te zenden.

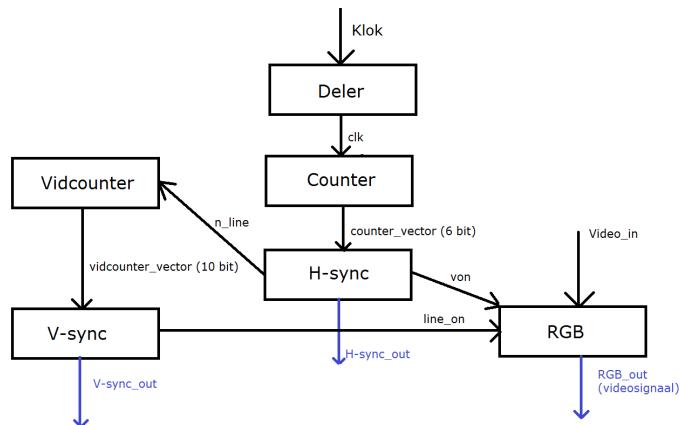
Regels	Wat	V-Sync	Videosignaal
0 tot 11	Front-porch	Hoog	Uit
11 tot 13	H-sync dip	Laag	Uit
13 tot 44	Back-porch	Hoog	Uit
44 tot 524	Video on	Hoog	Aan

Figuur 9.5: Timing voor de V-Sync

Er wordt echter maar een keer per twintig regels een nieuw signaal op het videosignaal gezet. Hierdoor zijn er dus twintig maal zo weinig regels beschikbaar. De resolutie wordt hierdoor 32 bij 24 pixels.

### 9.5. VHDL

Om alle signalen op het juiste moment naar de monitor te sturen, is de VGA-controller in delen opgedeeld.  
Deler Counter H-sync Vidcounter V-sync RGB



Figuur 9.6: Schema van de componenten

#### 9.5.1. DELER

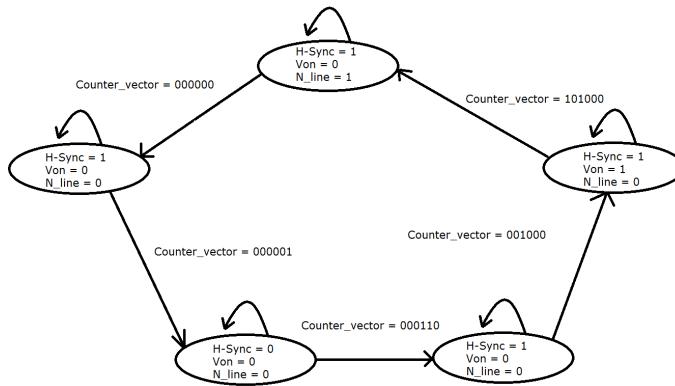
Dit component deelt de klok door vijf. Hierdoor is de klok 1.2 Mhz. Wanneer er over een klok gesproken wordt binnen de VGA-controller, gaat het over de klok van 1.2 Mhz.

#### 9.5.2. COUNTER

Dit component telt klokslagen. Bij een resolutie van 32 bij 24 pixels, is de pixelklok gelijk aan de gedeelde klok van 1.2 Mhz. Iedere  $1/1.2 \times 10^6$  seconde (mits von en line-on hoog zijn) komt er een nieuwe pixel op het scherm. De counter telt tot en met 40 en gaat daarna weer terug naar nul.

#### 9.5.3. H-SYNC

Dit component is een FSM. De input is de counter-vector (6 bit). De outputs zijn: Von, N-line en H-sync. Von en N-line blijven binnen de chip en H-sync wordt op de VGA uitgang gezet.



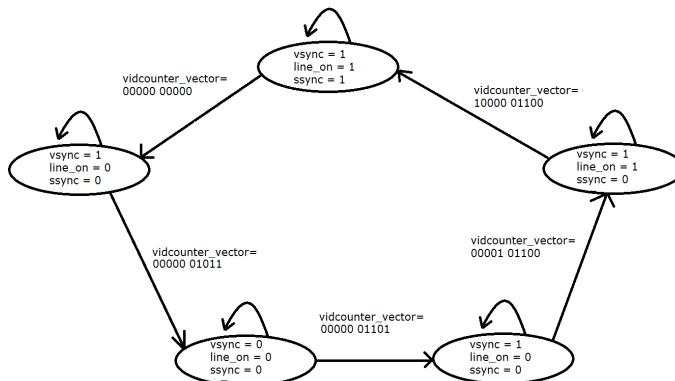
Figuur 9.7: FSM H-Sync

#### 9.5.4. VIDCOUNTER

Dit component werkt hetzelfde als de counter. Het enige verschil is dat nu niet de klok wordt geteld, maar het signaal N-line. N-line is 1 op het moment dat er een nieuwe regel begint. De vidcounter is de counter voor de V-sync. De counter telt tot en met 524 en gaat daarna weer terug naar nul.

#### 9.5.5. V-SYNC

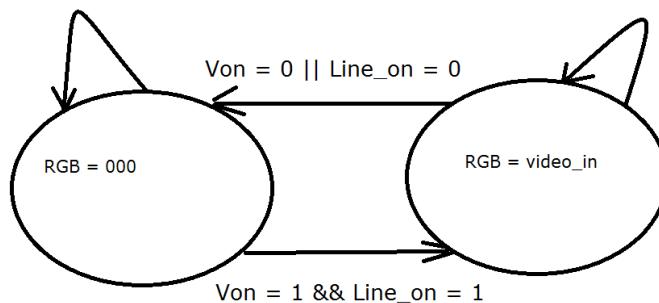
Dit is een FSM. De input is vid-counter-vector (10 bit). De outputs zijn V-sync, line-on en Ssync. Ssync en line-on blijven binnen de chip en V-sync wordt op de VGA uitgang gezet.



Figuur 9.8: FSM V-Sync

#### 9.5.6. RGB

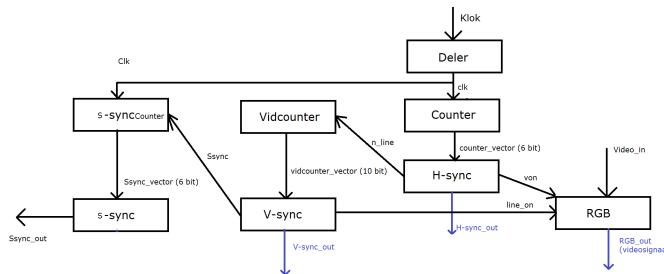
Dit component zorgt ervoor dat er pixels op het scherm ingekleurd kunnen worden. Op het moment dat Von en Line-on beide 1 zijn, zit video-in aan de uitgang RGB. Dit signaal wordt de op VGA poort gezet en daarmee rechtstreeks naar het scherm gestuurd.



Figuur 9.9: FSM RGB

## 9.6. VIDEOSIGNAAL

Het videosignaal komt uit het SRAM. Om het SRAM de juiste waarden naar de VGA-controller te laten sturen, zijn er een aantal dingen nodig. Adres Leescommando Sklok. De Sklok is vrij eenvoudig. Dit is namelijk de klok waarop het SRAM bits moet gaan sturen. Aangezien er binnen de VGA controller wordt gewerkt met een klok van 1.2 Mhz, hoeft dit signaal alleen te worden doorgestuurd. Het leescommando voor het SRAM is 0000 0011. Na het leescommando, moet het adres worden gegeven. Dit adres is 0000 0000 0000 0000. Kortom, er dient een signaal van 24 bit te worden verstuurd op de klok naar het SRAM. Na dit signaal, gaat het SRAM direct waarden gegeven. Het is dus belangrijk dat op de laatste regel de 24 bits worden verstuurd zodat er op de eerste regel direct kan worden uitgelezen. Hiervoor is het signaal Ssync. Dit komt uit de V-sync.



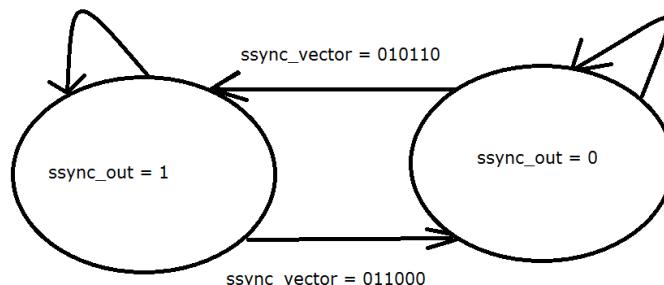
Figuur 9.10: Schema van de componenten

### 9.6.1. S-SYNCCOUNTER

Deze counter begint met tellen als Ssync hoog is. Vervolgens telt de counter tot 40. Daarna begint hij weer opnieuw.

### 9.6.2. S-SYNC

Net als de H-Sync en V-Sync is dit een FSM met ongeveer dezelfde werking.



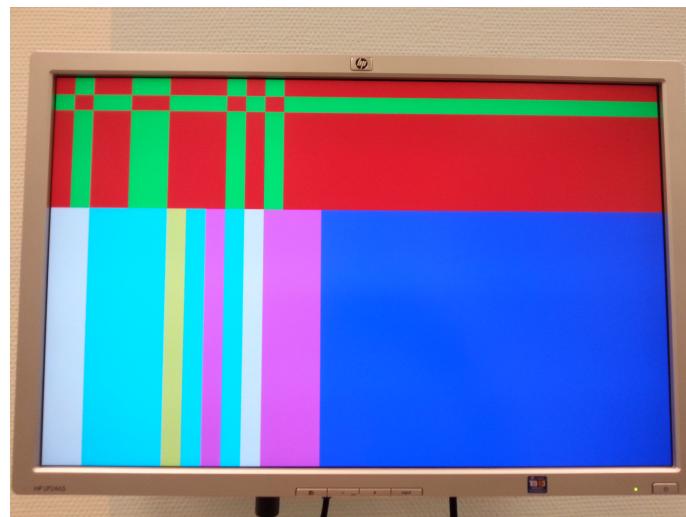
Figuur 9.11: FSM Ssync

## 9.7. CONCLUSIE

Om alle signalen op het juiste moment naar de monitor te sturen, is de VGA-controller in zes delen opgedeeld. Daarnaast zijn er nog twee componenten nodig om het SRAM aan te sturen. Hierdoor komt het totaal op acht componenten uit. Deler Counter H-sync Vidcounter V-sync RGB Ssync-counter Ssync

## 9.8. RESULTATEN

Op de foto is een testscherm te zien. Hierbij zijn enkele pixels te zien in verschillende kleuren.



Figuur 9.12: Testscherm vanaf de FPGA

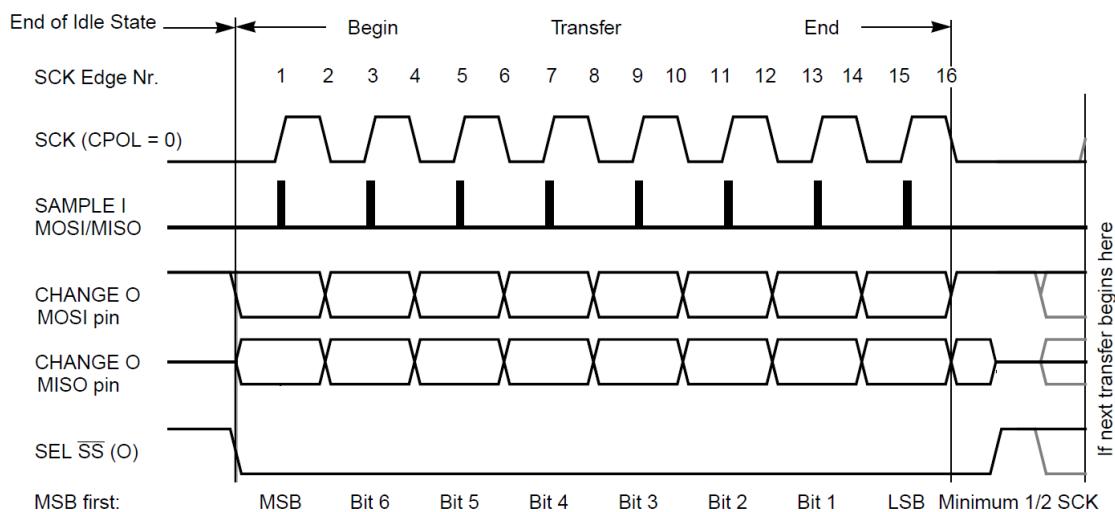
# 10

## SPI

Serial Peripheral Interface in het kort ook SPI genoemd is een de facto-standaard die ooit door Motorola bedacht is voor het communiceren met randapparatuur, zoals embedded systems, sensors en memory cards. Omdat SPI niet een officiële standaard is er geen specificatie om aan te houden bij het ontwerpen, om toch een refentie te hebben is er gebruikt gemaakt van [?] waarin het ontwerp van een SPI module wordt omschreven. Bij SPI heb je de master en de slave deze communiceren met elkaar via twee seriële datalijnen daarnaast is er nog een kloklijn die aangeeft wanneer er data overgedragen wordt. De kloklijn wordt aangestuurd door de master, dit houdt in dat de slave niet kan pauzeren en altijd data aan de master aan moet bieden. De standaard definieert enkel hoe de data van de master naar de slave en vice versa over wordt gedragen, niet hoe de data verwerkt wordt, dit heeft als gevolg dat randapparatuur zelf nog moet definiëren hoe de communicatie verloopt.

### 10.1. FUNCTIONELE BESCHRIJVING

De SPI module begint met zenden/ontvangen zodra het send signaal hoog is en stopt zodra er acht bits zijn verzonden/ontvangen, daarna kunnen de ontvangen bits uitgelezen worden en nieuwe bits in worden geladen om te verzenden. Het overbrengen van het shift register dat in de master zit naar de slave en vice versa deze shift registers zijn in een kring op elkaar aangesloten, waarbij altijd de meest significante bit wordt verzonden. Voor de communicatie met de slave wordt een slave klok signaal gegenereerd, de data overdracht is gesynchroniseerd ten opzicht van dit signaal. Data wordt gesampled op de opgaande klokflank en de data wordt geshift op de neergaande klokflank, een verduidelijking van hoe deze timing precies werkt is te zien in figuur ??

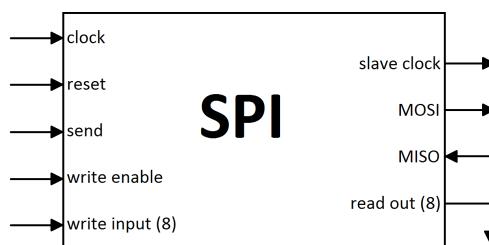


Figuur 10.1: Diagram van SPI timing

## 10.2. INPUTS EN OUTPUTS

Hieronder een beschrijving van alle in- en outputs van de SPI module zoals te zien in figuur ??.

- Clock: het kloksignaal waarop de SPI draait, het slave clock signaal zal dezelfde frequency hebben als dit kloksignaal
- Reset: de hoofd reset van de SPI
- Send: de input die aangeeft wanneer er begonnen moet worden met zenden/ontvangen
- Write\_enable: als deze hoog is zal write\_in(8) in het shift register geladen worden
- Write\_in(8): de bits die naar het shift register worden geschreven als write\_enable hoog is
- Read\_out(8): de waarde die in het shift register staat
- Slave clock: het slave kloksignaal die de communicatie met de slave aanstuurt
- MOSI: de datalijn van de master naar de slave
- MISO: de datalijn van de slave naar de master



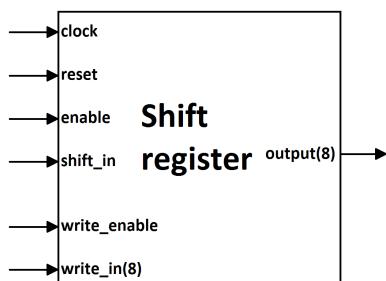
Figuur 10.2: Diagram van SPI circuit

## 10.3. IMPLEMENTATIE

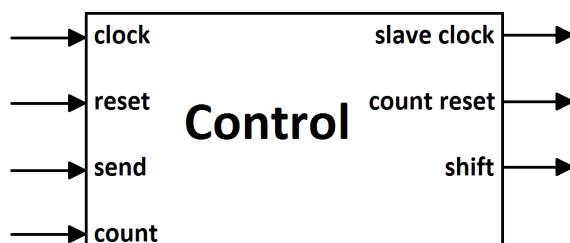
Voor de implementatie van de SPI is er voor gekozen om het in drie subsystemen op te delen:

- Counter: een simpele teller die de opgaande klokslagen van het slave clock signaal telt.
- Shift register: een shift register van acht bits die shift op de neergaande klokflank als het enable signaal hoog is en nieuwe waarden inlaad als de write enable hoog is. Het blokschema van het shift register is te zien in figuur ??.
- Control: een statemachine die er voor zorgt dat de SPI stopt met shiften na 8 klokslagen van de slave klok, zodat er tijd is om het register uit te lezen of nieuwe waarden in te laden. Het blokschema van Control is te zien in figuur ??.

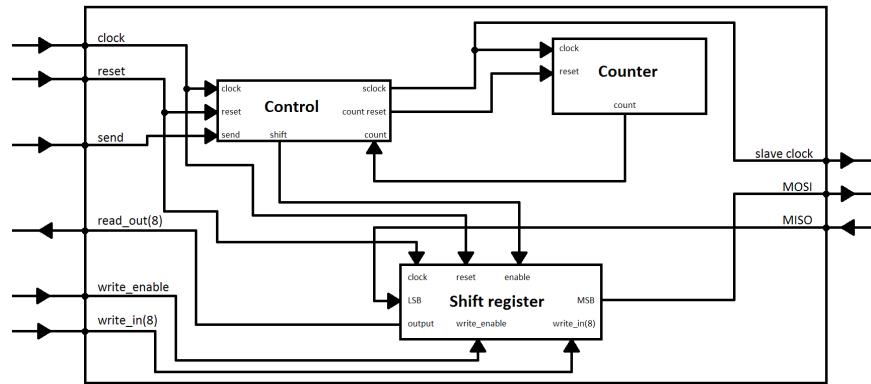
Deze drie subsystemen zijn aan elkaar verbonden volgens het schema in figuur ??.



Figuur 10.3: Diagram van Shift register



Figuur 10.4: Diagram van Control



Figuur 10.5: Diagram van de verbinding van de componenten

# 11

## SD-KAART

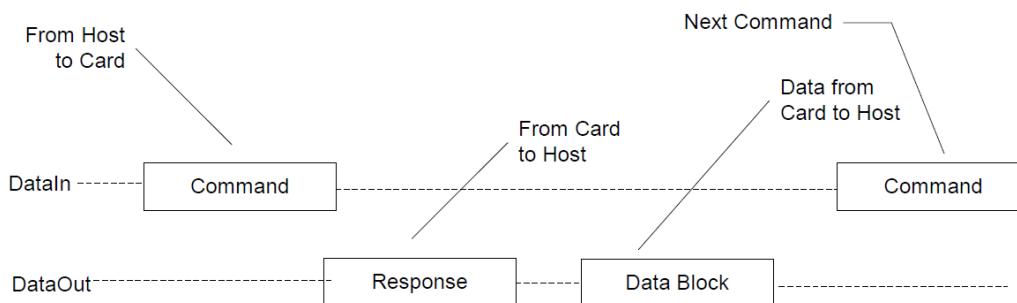
Voor het opslaan van de instructies voor de CPU is gekozen voor een SD-kaart. Voor het uitlezen van een SD-kaart zijn er drie verschillende modes, twee hiervan zijn echter gebaseerd op een zelf ontworpen systeem van SanDisk. De derde mode is echter gebaseerd op SPI waarvan al een module is die al ontworpen is, daarom is hier ook voor gekozen.

### 11.1. SD COMMUNICATIE

Aangezien SPI niets zegt over hoe de communicatie verloopt of wat de betekenis van de overgebrachte data is, definieert SanDisk in de SD specificatie een aantal commando's die naar de SD-kaart gestuurd kunnen worden. De specificatie die gebruikt is is uit 2003 [?], dit is omdat dit de enige volledige specificatie is die te vinden was. Dit is omdat je een licentie bij SanDisk moet hebben om toegang te hebben tot de meest recente en volledige specificaties.

### 11.2. SD COMMANDO'S & RESPONSES

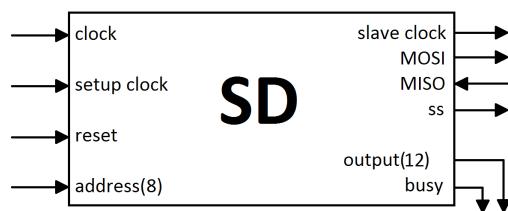
De communicatie met de SD-kaart verloopt als volgt: er wordt een commando naar de SD-kaart verstuurt, de SD-kaart geeft antwoordt of het een correct commando is en of het commando verwerkt kan worden, daarna komt eventueel nog, bij het lezen, data van de SD-kaart of, bij het schrijven, data naar de SD-kaart. Zodra alles is afgehandeld kan er een nieuw commando gestuurd worden. Hoe dit precies verloopt is te zien in figuur ?? hierin is data versturen naar de SD-kaart weggelaten omdat hier geen gebruik van wordt gemaakt.



Figuur 11.1: Commando communicatie van SD-kaart

### 11.3. FUNCTIONELE OMSCHRIJVING

De SD module is zo ontworpen dat deze een adres binnen krijgt van de program counter van de CPU deze uitleest vanaf de SD-kaart en deze vervolgens aanbiedt aan de CPU. Zolang de SD module nog bezig is met het uitlezen van het adres zal het busy signaal hoog zijn.



Figuur 11.2: Diagram van SD-kaart

## 11.4. INPUTS EN OUTPUTS

Hieronder een beschrijving van alle in- en outputs van de SD module zoals te zien in figuur ??.

- Clock: het kloksignaal waar de SD module op draait, dit is ook het kloksignaal waar de SPI na initialisatie op zal draaien
- Setup clock: het kloksignaal dat tijdens de initialisatie gebruikt zal worden, deze moet tussen de 100 en 400 kHz liggen
- Reset: het algemene reset signaal
- Address: het adres dat van de SD-kaart zal worden afgelezen
- Output(12): dit zal de waarde zijn die op het adres staat zodra deze is uitgelezen
- Busy: dit signaal geeft aan of de SD module bezig is
- Slave clock: het slave kloksignaal dat de SD-kaart aanstuurd
- MOSI: de datalijn van de SD module naar de SD-kaart
- MISO: de datalijn van de SD-kaart naar de SD module
- SS: het slave select signaal, dit signaal wordt gebruikt om de SD-kaart in SPI mode te krijgen en om aan te geven dat de SD-kaart geselecteerd is.

# 12

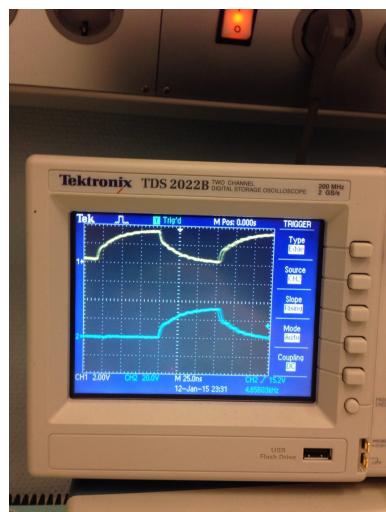
## CONCLUSIE EN AANBEVELINGEN

### 12.1. CONCLUSIE

Het EPO-3 project van de bachelor opleiding Electrical Engineering begon in tegen stelling tot voorgaande projecten vrij soepel. Als groep waren we het snel met elkaar eens, het zou een spelcomputer worden met een ultrasone besturing. Gaande weg liepen we tegen steeds meer complicaties aan.

#### 12.1.1. COMPLICATIES

Gedurende de uitvoering van het project is gebleken dat er een aantal problemen zijn ontstaan met betrekking tot de communicatie met externe geheugenmodules. Zo is de communicatie met de SD kaart niet op gang gekomen wegens een incorrecte VHDL code. Tevens het SRAM heeft op geen enkele manier gereageerd op de SPI signalen. Mogelijke oorzaken van het niet functioneren van het SRAM en de SD-kaart zouden kunnen liggen bij het uitgangssignaal van de FPGA. In figuur ?? is een foto te zien met daarop de uitgangssignalen van de FPGA. Deze signalen vertonen duidelijk geen correcte blokvorm. Voor onderzoek naar het oplossen van dit probleem bleek te weinig tijd te zijn.



Figuur 12.1: Uitgang FPGA

#### 12.1.2. RESULTATEN

Door het wegvallen van het SRAM en de SD-kaart is besloten om de volledige VGA output te laten vallen, tevens de externe ROM (in de vorm van de SD kaart) is niet gebruikt bij het uiteindelijke design. Er is besloten om met de werkende componenten een rekenmachine te maken. Deze zal zijn ingang ontvangen via een externe microcontroller over SPI. Er zal gebruik worden gemaakt van de 8-bit CPU en on-board ROM. In de

bijlage is de gebruikte assambly code te vinden welke de operaties optellen, aftrekken en vermenigvuldigen realiseert.

## 12.2. AANBEVELINGEN

Achter af is het natuurlijk simpeler gezegd als gedaan, maar het is bij dit EPO-3 project belangrijk een duidelijk doel te stellen. Er moet daarbij vooral gelet worden op de haalbaarheid van het ontwerp. Het is met deze vrijheid in keuze dan ook gevaarlijk om te veel te willen realiseren. Dit heeft bij onze project groep voor veel stress gezorgd, wat zeker niet ideaal is aan het einde van een periode.

# 13

## DISCUSSIE

### 13.1. REFLECTIE OP HET EINDONTWERP

Zoals eerder in het verslag al aangegeven is het ons niet gelukt om een werkende spelcomputer te realiseren. Met de werkende onderdelen zijn we vervolgens verder gegaan en hebben een simpele rekenmachine ontworpen. De problemen die we tijdens het proces tegen kwamen zaten vooral in de communicatie tussen de FPGA en de externe componenten. Het lukte ons niet om de SD-kaart uitlezing werkend te krijgen. Deze leek telkens te werken, maar was onmogelijk stabiel te krijgen. Ditzelfde probleem ondervonden we met beide SRAM geheugens. Deze onderdelen leken het hele project haalbaar, pas in de laatste week kwamen de problemen naar voren. Dit had tot gevolg dat het onmogelijk geworden was iets sterker te produceren dan dat ons nu gelukt is.

### 13.2. REFLECTIE OP HET GROEPSPROCES

EPO-3 groep A5 begon dit project sterk en georganiseerd, dit was grotendeels te danken aan onze voorzitter Erné Bronkhorst. Beslissingen konden gezamenlijk en snel worden genomen, dit gaf ons een goede voorsprong. Ook de taakverdeling en planning leek erg soepel te lopen. Gaande weg het project werd steeds meer duidelijk dat niet alle onderdelen evenveel tijd kosten, dit had tot gevolg dat er achteraf in moest worden gesprongen in onderdelen en dat koste erg veel tijd. De besturing was daar een goed voorbeeld van, deze kon eigenlijk vrij snel gerealiseerd worden. Hierdoor kwamen er twee mensen zonder taak te zitten, gelukkig kon dit vrij snel worden opgelost. Er moest nog een plan van aanpak worden geschreven en een toplevel beschrijving ontwerpen. In de laatste fase van het project heeft de taakverdeling ervoor gezorgd dat we dit toch nog met een goed resultaat konden beëindigen.

## BIBLIOGRAFIE

- [1] SPI Block Guide, <http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf>, benaderd in oktober en november. 2014.
- [2] SanDisk Secure Digital Card Product Manual Version 1.9, <http://alumni.cs.ucr.edu/~amitra/sdcard/Prod-ManualSDCardv1.9.pdf>, benaderd in november en december. 2014.

# A

## PLAN VAN AANPAK

### A.1. ACHTERGRONDEN

Door onze aanmelding bij de studie Electrical Engineering aan de TU-Delft hebben wij van deze organisatie opdracht gekregen om een eigen chip te ontwerpen, verder is alles geheel voor eigen keuze. De opdracht zal door de opdrachtgevers gekozen naam 'Ontwerp een chip' benoemd worden. Dit project is aangenomen door alle leden van de projectgroep A5, tevens zal dit project worden begeleid door Dr. R. Ishihara. Dr. R. Ishihara zal hiernaast fungeren als onze tutor. Deze opdracht zal binnen de Technische Universiteit Delft worden verwerkt. Dit project speelt in op de in college naar voren gekomen onderdelen, waaronder 'Programmeren in C' en 'Digitale systemen'. Het dient als doel onze kennis te verdiepen en te verbreden in de elektrotechniek. De uiteindelijke stakeholders zijn behalve de opdrachtgevers, daardoor ook de projectnemers. Deze opdracht is een onderdeel en vereiste voor het behalen van de Bachelor opleiding 'Electrical Engineering'.

### A.2. PROJECTRESULTAAT

Tijdens het EPO-3 project 'Ontwerp een chip' dient een chip ontworpen te worden, dit kan bijvoorbeeld het implementeren van een spelletje of een Infrarood-Besturing op een chip zijn. Deze opdracht mag door de projectgroep gekozen worden. Bij het ontwerp van de chip dient gebruik gemaakt te worden van Sea-of-gates technologie.

#### A.2.1. DOELSTELLINGEN VAN HET PROJECT

Allereerst is het doel van het project, het succesvol uitvoeren van de opdracht gekregen van de opdrachtgever. Belangrijk hierbij is het ontwerpen aan de hand van globale productspecificaties met randvoorwaarden. Daarnaast, zeker niet onbelangrijk, zijn de vaardigheden die universiteit ons mee wilt geven. Deze vaardigheden komen gaandeweg het project steeds meer naar voren. Hierbij speelt projectmatig werken in een groep een grote rol. Ook worden de projectnemers middels dit project getraind in vergaderen, overleggen en plannen. Als laatst worden verschillende technische vaardigheden in de praktijk getraind waaronder programmeren, simuleren en het bouwen van verschillende schakelingen. Er moet tijdens dit EPO-3 project strikt worden gelet op het combineren en testen van de verschillende ontwerpen, hierbij kan worden gebruik gemaakt van moderne computerhulpmiddelen. De projectleden zullen gaande weg dit project hierin gesteund en getest worden.

#### A.2.2. EINDRESULTAAT

Als eindresultaat zal er een chip moeten worden ontworpen, daarnaast moet er een verslag worden geschreven. De chip moet aan een aantal eisen voldoen, namelijk:

1. Op de chip moet een spelcomputer gebouwd worden, deze spelcomputer dient spelletjes van een SD-kaart te kunnen weergeven op een beeldscherm.
2. Ultrasone sensoren moeten de besturing vormen voor de spelcomputer.
3. Er moet minimaal een spel ontworpen worden (Pong). Mocht er tijd over zijn dan kunnen er meer spelletjes ontworpen worden.

4. Het beeld dient te worden weergegeven met VGA op een beeldscherm.

Ook het verslag moet aan een aantal eisen voldoen:

1. Het verslag bestaat uit twee delen, het technische verslag en het procesverslag.
2. Het technisch verslag documenteert de ontwerpkeuzes en de prestaties van de chip.
3. De weekverslagen/notulen vormen een basis voor het procesverslag.
4. Het verslag dient te worden geschreven volgens het format van de projecthandleiding.

### A.3. PROJECTACTIVITEITEN

Het project kan worden opgedeeld in het uitvoeren van de volgende taken:

1. Opstellen plan van aanpak
  - verzamelen en bestuderen informatie
  - maken van concept plan van aanpak
  - bespreken plan van aanpak met opdrachtgever
2. Ontwerpen en bouwen besturing
3. Ontwerpen microprocessor
4. Ontwerpen en bouwen geheugenelement (SD)
5. Ontwerpen VGA + SRAM module
6. Schrijven Pong (spelletje)

### A.4. PROJECTGRENZEN

In dit project is veel vrijheid gegeven door de opdrachtgever. Dat neemt niet weg dat er aan een aantal eisen moet worden voldaan, voornamelijk de beperkingen die de chip met zich mee brengt. Deze eisen staan beschreven in de projecthandleiding, en zijn hieronder even kort samengevat.

1. Het ontwerp dient klaar te zijn binnen het 2e kwartaal.
2. De schakeling dient te werken op een klokfrequentie van 6.144 MHz.
3. Elke projectgroep heeft  $0.4 \text{ cm}^2$  ruimte op de chip, dit komt overeen met 40.000 transistor paren.
4. Per groep zijn er 32 aansluitpinnen beschikbaar.
5. Voor de FSM's mogen alleen die van het type Moore gebruikt worden, daarnaast moet er altijd een reset aanwezig zijn.
6. Het streven is om zo weinig mogelijk componenten extern te gebruiken.

#### A.4.1. TUSSENRESULTATEN

De tussenresultaten van dit project worden uitgebreid beschreven in het gehele tussentijdse ontwerprapport en dus niet verder behandeld in het plan van aanpak.

## A.4.2. KWALITEITSBEWAKING

### Eindproduct

Het eindproduct van het derde project van de bachelor Electrical Engineering moet een goed werkende chip zijn die aan enkelen eisen voldoet, deze eisen zijn door de projectgroep zelf opgesteld:

- Op de chip moet een spelcomputer gebouwd worden, deze spelcomputer dient spelletjes van een SD-kaart te kunnen weergeven op een beeldscherm.
- Ultrasone sensoren moeten de besturing vormen voor de spelcomputer.
- Er moet minimaal een spel ontworpen worden (Pong). Mocht er tijd over zijn dan kunnen er meer spelletjes ontworpen worden.
- Het beeld dient te worden weergegeven met VGA op een beeldscherm.

### Controles

Tussendoor zullen er een aantal controles worden uitgevoerd om de kwaliteit te blijven bewaken, namelijk:

- Is het haalbaar om spelletjes extern te laden?
- Werkt de uitlees module van de SD-kaart?
- Is de ultrasone sensor besturing bruikbaar voor het eindresultaat?
- Lukt het om met de VGA een beeld op een scherm te printen?

Deze punten zullen voornamelijk worden gecontroleerd door de groep zelf. Hierbij is er duidelijk een leider aangewezen die het overzicht behoud en zo de voortgang van het project controleert. Daarnaast vindt er nog een docentcontrole gaande weg tijdens het project plaats om de kwaliteit te bewaken, deze kan worden gedaan door de hoofddocent of door een van zijn assistenten.

## A.5. PROJECTORGANISATIE

### Organisatie

De projectgroep EPO-3 A5 bestaat uit acht leden. Om het overzicht te behouden is er een voorzitter benoemd. De voorzitter is Erné Bronkhorst. Verder bestaat de projectgroep uit:

- Marc Zwalua (Notulist)
- Niels de Winter
- Wietse Bouwmeester
- Bauke Meekes
- Job Tijhuis
- Hans Okkerman
- Jordy van der Horst

### Onderlinge afspraken:

- Om de voortgang te bewaken zal er wekelijks een evaluatie plaatsvinden waarin dit wordt bekeken. Hierin wordt zo nodig de planning voor de komende weken bijgesteld.
- Als groepsleden horen wij altijd op tijd en verplicht aanwezig te zijn bij elke geplande bijeenkomst. Als dit onverwachts niet mogelijk is, zal hier zo snel mogelijk melding van gemaakt moeten worden.
- Als er onderlinge geschillen zijn, dan worden deze besproken met de hoofddocent. Dit zorgt ervoor dat de voortgang niet in gevaar komt.
- Alle groepsleden moeten garant staan voor een gelijke bijdrage aan het project.

### Verantwoording

Er zal constant tijdens de project middagen mondeling gerapporteerd worden over de voortgang van het project. Schriftelijk wordt er halverwege de periode een mid-term rapport geschreven, hierin wordt de voortgang opgenomen. De verantwoording zal bij Dr. R. Ishihara worden gedaan.

### A.5.1. PLANNING

Een duidelijk schema van onze planning is toegevoegd als bijlage.

### A.5.2. RISICOANALYSE

Hier volgt een overzicht van de eventuele risico's die we tijdens EPO-3 tegen zouden kunnen komen. Door middel van deze analyse worden deze risico's zoveel mogelijk vermeden. **Externe risico's:**

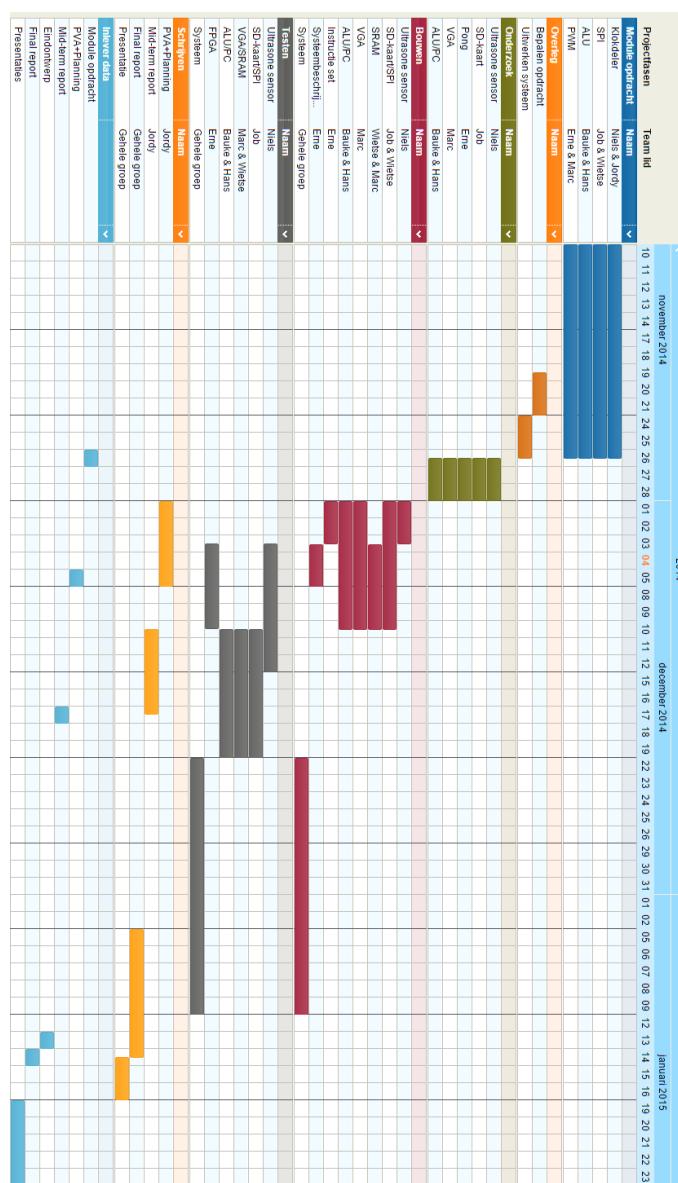
- Onduidelijke informatie voorziening.
- Onvoldoende feedback vanuit de docenten

**Interne risico's:**

- Verliezen van meetresultaten.
- Botsing tussen de groepsleden.
- Het eventueel ontbreken van kennis of inzet onder de groepsleden.
- Het project niet af kunnen ronden door een tekort aan tijd.

Een aantal interne risico's kunnen worden vermeden door het aanstellen van de voorzitter, deze zal de voortgang bewaken en alle resultaten netjes beheren. Voor de externe risico's zal er vooral zelf moeten worden gezocht naar oplossingen in combinatie met de docenten.

## A.6. BIJLAGE



Figuur A.1: Planning A5

# B

## BIJLAGE - VHDL BESTANDEN

### B.1. BLACKBOX

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

entity blackbox is
  port(sclk_m_0:in    std_logic;
       mosi_m_0:in    std_logic;
       miso_m_0:out   std_logic;
       ss_m_0  :in    std_logic;

       sclk_m_1:in    std_logic;
       mosi_m_1:in    std_logic;
       miso_m_1:out   std_logic;
       ss_m_1  :in    std_logic;

       sclk_s_0:out   std_logic;
       mosi_s_0:out   std_logic;
       miso_s_0:in    std_logic;
       ss_s_0  :out   std_logic;

       sclk_s_1:out   std_logic;
       mosi_s_1:out   std_logic;
       miso_s_1:in    std_logic;
       ss_s_1  :out   std_logic;

       s:         in    std_logic);
end blackbox;

library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of blackbox is

component stream is
  port(IN_0 :in    std_logic;
       IN_1 :in    std_logic;
       OUT_0:out   std_logic;
       OUT_1:out   std_logic;
       E     :in    std_logic);
```

```

end component;

signal scm0, mom0, mim0, ssm0, scml, moml, miml, ssml: std_logic;
signal scs0, mos0, mis0, sss0, scs1, mos1, mis1, sss1: std_logic;
signal ss: std_logic;

begin
stream0: stream port map(scm0, scml, scs0, scs1, ss);
stream1: stream port map(mom0, moml, mos0, mos1, ss);
stream2: stream port map(ssm0, ssml, sss0, sss1, ss);
stream3: stream port map(mis0, mis1, mim0, miml, ss);

ss <= s;

scm0 <= sclk_m_0;
mom0 <= mosi_m_0;
miso_m_0 <= mim0;
ssm0 <= ss_m_0;

scml <= sclk_m_1;
moml <= mosi_m_1;
miso_m_1 <= miml;
ssml <= ss_m_1;

sclk_s_0 <= scs0;
mosi_s_0 <= mos0;
mis0 <= miso_s_0;
ss_s_0 <= sss0;

sclk_s_1 <= scs1;
mosi_s_1 <= mos1;
mis1 <= miso_s_1;
ss_s_1 <= sss1;

end behaviour;

library IEEE;
use IEEE.std_logic_1164.ALL;

entity stream is
  port(IN_0 :in std_logic;
        IN_1 :in std_logic;
        OUT_0:out std_logic;
        OUT_1:out std_logic;
        E      :in std_logic);
end stream;

library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of stream is

begin
process(IN_0, IN_1, E)
begin
  if E = '0' then
    OUT_0 <= IN_0;

```

```
        OUT_1 <= IN_1;  
    else  
        OUT_0 <= IN_1;  
        OUT_1 <= IN_0;  
    end if;  
end process;  
end behaviour;
```

## B.2. ROM

```

library IEEE;
use IEEE.std_logic_1164.all;

entity rom is
port
(
    rom_a:    in std_logic_vector (7 DOWNTO 0);      -- address
    rom_d:    out std_logic_vector (11 DOWNTO 0)      -- instruction
);
end rom;

architecture behavioural OF rom IS
begin
    with rom_a select
        rom_d <=
    "111100000000" when "00000000",
    "011000000001" when "00000001",
    "101000101111" when "00000010",
    "100000000001" when "00000011",
    "010000001110" when "00000100",
    "011000000001" when "00000101",
    "101000011111" when "00000110",
    "100000000001" when "00000111",
    "010000110010" when "00001000",
    "011000000001" when "00001001",
    "101000001111" when "00001010",
    "100000000001" when "00001011",
    "010001011001" when "00001100",
    "000100000000" when "00001101",
    "111100000000" when "00001110",
    "011000000001" when "00001111",
    "101000101111" when "00010000",
    "100000000001" when "00010001",
    "010000001110" when "00010010",
    "111100000000" when "00010011",
    "011000000001" when "00010100",
    "110000001111" when "00010101",
    "011100000010" when "00010110",
    "011000000001" when "00010111",
    "110000010000" when "00011000",
    "001100011100" when "00011001",
    "010100000001" when "00011010",
    "011100000011" when "00011011",
    "011000000001" when "00011100",
    "110000100000" when "00011101",
    "001100100010" when "00011110",
    "011000000011" when "00011111",
    "100000000010" when "00100000",
    "011100000011" when "00100001",
    "011000000001" when "00100010",
    "110001000000" when "00100011",
    "001100101000" when "00100100",
    "011000000011" when "00100101",
    "100000000010" when "00100110",
    "001100100100" when "00100111"
);

```

```
"011100000011" when "00100111",
"011000000001" when "00101000",
"110010000000" when "00101001",
"001100101110" when "00101010",
"011000000011" when "00101011",
"100000001000" when "00101100",
"011100000011" when "00101101",
"011000000011" when "00101110",
"100100000010" when "00101111",
"011100000001" when "00110000",
"000100000000" when "00110001",
"111100000000" when "00110010",
"011000000001" when "00110011",
"101000011111" when "00110100",
"100000000001" when "00110101",
"010000110010" when "00110110",
"111100000000" when "00110111",
"011000000001" when "00111000",
"110000001111" when "00111001",
"011100000010" when "00111010",
"011000000001" when "00111011",
"110000010000" when "00111100",
"001101000000" when "00111101",
"010100000001" when "00111110",
"011100000011" when "00111111",
"011000000001" when "01000000",
"110000100000" when "01000001",
"001101000110" when "01000010",
"011000000011" when "01000011",
"100000000010" when "01000100",
"011100000011" when "01000101",
"011000000001" when "01000110",
"110010000000" when "01000111",
"001101001100" when "01001000",
"011000000011" when "01001001",
"100000000010" when "01001010",
"011100000011" when "01001011",
"011000000001" when "01001100",
"110010000000" when "01001101",
"001101010010" when "01001110",
"011000000011" when "01001111",
"100000001000" when "01010000",
"011100000011" when "01010001",
"011000000011" when "01010010",
"101011111111" when "01010011",
"1000000000001" when "01010100",
"100100000010" when "01010101",
"011100000001" when "01010110",
"111100000000" when "01010111",
"000100000000" when "01011000",
"111100000000" when "01011001",
"011000000001" when "01011010",
"101000001111" when "01011011",
"100000000001" when "01011100",
"010001011001" when "01011101",
"011000000001" when "01011110",
```

```
"110000001111" when "01011111",
"011100000010" when "01100000",
"011000000001" when "01100001",
"110000010000" when "01100010",
"001101100110" when "01100011",
"010100000001" when "01100100",
"011100000011" when "01100101",
"011000000001" when "01100110",
"110000100000" when "01100111",
"001101101100" when "01101000",
"011000000011" when "01101001",
"100000000010" when "01101010",
"011100000011" when "01101011",
"011000000001" when "01101100",
"110001000000" when "01101101",
"001101110010" when "01101110",
"011000000011" when "01101111",
"100000000100" when "01110000",
"011100000011" when "01110001",
"011000000001" when "01110000",
"011000000001" when "01110010",
"110010000000" when "01110011",
"001101111000" when "01110100",
"011000000011" when "01110101",
"100000000100" when "01110110",
"011100000011" when "01110111",
"011000000001" when "01111000",
"011000000001" when "01111001",
"110000000001" when "01111002",
"001110100010" when "01111010",
"011000000011" when "01111011",
"0111000000100" when "0111100",
"0110000000010" when "0111110",
"001110100101" when "01111111",
"0110000000010" when "10000000",
"100100000011" when "10000001",
"011100000101" when "10000010",
"011000000010" when "10000011",
"1100000000100" when "10000100",
"001110101000" when "10000101",
"011000000011" when "10000110",
"0110000000011" when "10000111",
"100100000011" when "10010000",
"0111000001000" when "10010001",
"0110000001000" when "10010010",
"1001000001000" when "10010011",
"0111000001000" when "10010100",
"0110000001000" when "10010101",
"1001000001000" when "10010110",
```

```
"011100000111" when "10010111",
"011000000100" when "10011000",
"100100000101" when "10011001",
"011100001000" when "10011010",
"011000000110" when "10011011",
"100100000111" when "10011100",
"011100001001" when "10011101",
"011000001000" when "10011110",
"100100001001" when "10011111",
"011100000001" when "10100000",
"000100000000" when "10100001",
"010100000000" when "10100010",
"011100000100" when "10100011",
"000101111101" when "10100100",
"010100000000" when "10100101",
"011100000101" when "10100110",
"000110000011" when "10100111",
"010100000000" when "10101000",
"011100000110" when "10101001",
"000110001100" when "10101010",
"010100000000" when "10101011",
"011100000111" when "10101100",
"000110011000" when "10101101",
"000000000000" when others;
end behavioural;
```

## B.3. CPU

```

library ieee;
use ieee.std_logic_1164.all;

entity cpu is
port(  cpu_RST :      in      std_logic;
        cpu_en   :      in      std_logic;
        cpu_in   :      in      std_logic_vector(7 downto 0);
        cpu_instr:      in      std_logic_vector(11 downto 0);
        cpu_pc   :      out     std_logic_vector(7 downto 0);
        cpu_out  :      out     std_logic_vector(7 downto 0));
end entity cpu;

architecture behaviour of cpu is

-- components --

component instr_buf
port( buf_in  : in  std_logic_vector(11 downto 0);
      buf_oe   : in  std_logic;
      buf_out  : out std_logic_vector(11 downto 0));
end component;

component decoder
port( decoder_in  : in  std_logic_vector(11 downto 0);
      decoder_c   : in  std_logic;
      decoder_z   : in  std_logic;
      decoder_pc_inc : out std_logic;
      decoder_pc_ld  : out std_logic;
      decoder_ibufoe : out std_logic;
      decoder_aregld : out std_logic;
      decoder_abufoe : out std_logic;
      decoder_bregld : out std_logic_vector(4 downto 0);
      decoder_bbbufoe : out std_logic_vector(4 downto 0);
      decoder_alu   : out std_logic_vector(2 downto 0);
      decoder_argout : out std_logic_vector(7 downto 0));
end component;

component alu
port ( alu_A   : in  std_logic_vector(7 downto 0);      -- input A
        alu_B   : in  std_logic_vector(7 downto 0);      -- input B
        opcode : in  std_logic_vector(2 downto 0);      -- opcode
        alu_clk: in  std_logic;                          -- clk
        alu_c   : out std_logic;                         -- flag C
        alu_z   : out std_logic;                         -- flag Z
        alu_y   : out std_logic_vector(7 downto 0));    -- Output result y
end component;

component pc_counter
port ( pc_in   : in  std_logic_vector(7 downto 0);
        pc_inc  : in  std_logic;
        pc_ld   : in  std_logic;
        pc_RST  : in  std_logic;
        pc_en   : in  std_logic;
        pc_out  : out std_logic_vector(7 downto 0));

```

```

end component;

component buf_arg
port( buf_in : in std_logic_vector(7 downto 0);
        buf_oe : in std_logic;
        buf_out : out std_logic_vector(7 downto 0));
end component;

component buf_A
port( buf_in : in std_logic_vector(7 downto 0);
        buf_oe : in std_logic;
        buf_out : out std_logic_vector(7 downto 0));
end component;

component reg_A
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld : in std_logic;
        reg_out : out std_logic_vector(7 downto 0));
end component;

component reg_cluster
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_select : in std_logic_vector(4 downto 0);
        buf_select : in std_logic_vector(4 downto 0);
        buf_in : in std_logic_vector(7 downto 0);
        buf_out : out std_logic_vector(7 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end component;

-- Signals --
signal cpu_dec_instr : std_logic_vector(11 downto 0); -- instruction from SPI
signal cpu_dec_o : std_logic_vector(7 downto 0); -- operand to buf_i
signal cpu_dec_ibufoe : std_logic; -- ibuf_oe
signal cpu_dec_pc_inc : std_logic; -- program counter increment
signal cpu_dec_pc_ld : std_logic; -- program counter load
signal cpu_dec_abufoe : std_logic; -- abuf_oe
signal cpu_dec_aregld : std_logic; -- areg_ld
signal cpu_dec_bbufoe : std_logic_vector(4 downto 0); -- bbuf_oe
signal cpu_dec_bregld : std_logic_vector(4 downto 0); -- breg_ld
signal cpu_bus : std_logic_vector(7 downto 0); -- operand in bus
signal cpu_alu_c : std_logic; -- alu carry
signal cpu_alu_z : std_logic; -- alu z-flag
signal cpu_alu_out : std_logic_vector(7 downto 0); -- alu output
signal cpu_alu_op : std_logic_vector(2 downto 0); -- alu opcode
signal cpu_areg_out : std_logic_vector(7 downto 0); -- output reg_A
signal cpu_reg_out : std_logic_vector(7 downto 0); -- output reg_out

begin

lbl_instrbuf: instr_buf port map(
    buf_in => cpu_instr,

```

```

buf_oe => cpu_en,
buf_out => cpu_dec_instr);

lbl_decoder: decoder      port map
  decoder_in => cpu_dec_instr,
  decoder_c => cpu_alu_c,
  decoder_z => cpu_alu_z,
  decoder_pc_inc => cpu_dec_pc_inc,
  decoder_pc_ld => cpu_dec_pc_ld,
  decoder_ibufoe => cpu_dec_ibufoe,
  decoder_aregld => cpu_dec_aregld,
  decoder_abufoe => cpu_dec_abufoe,
  decoder_bregld => cpu_dec_bregld,
  decoder_bbufoe => cpu_dec_bbufoe,
  decoder_alu => cpu_alu_op,
  decoder_argout => cpu_dec_o);

lbl_cpu_ibuf: buf_arg      port map
  buf_in => cpu_dec_o,
  buf_oe => cpu_dec_ibufoe,
  buf_out => cpu_bus);

lbl_cpu_pc: pc_counter     port map
  pc_in => cpu_bus,
  pc_inc => cpu_dec_pc_inc,
  pc_ld => cpu_dec_pc_ld,
  pc_rst => cpu_RST,
  pc_en => cpu_en,
  pc_out => cpu_pc);

lbl_cpu_alu: alu           port map
  alu_A => cpu_bus,
  alu_B => cpu_areg_out,
  opcode => cpu_alu_op,
  alu_clk => cpu_en,
  alu_c => cpu_alu_c,
  alu_z => cpu_alu_z,
  alu_y => cpu_alu_out);

lbl_cpu_reg_A: reg_A       port map
  reg_in => cpu_alu_out,
  reg_clk => cpu_en,
  reg_rst => cpu_RST,
  reg_ld => cpu_dec_aregld,
  reg_out => cpu_areg_out);

lbl_cpu_buf_A: buf_A       port map
  buf_in => cpu_areg_out,
  buf_oe => cpu_dec_abufoe,
  buf_out => cpu_bus);

lbl_bregs: reg_cluster port map
  reg_in      => cpu_bus,
  reg_clk     => cpu_en,
  reg_rst     => cpu_RST,
  reg_select  => cpu_dec_bregld,
```

```
    buf_select  => cpu_dec_bbufoe,
    buf_in      => cpu_in,
    buf_out     => cpu_bus,
    reg_out     => cpu_out);
end behaviour;
```

### B.3.1. DECODER

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

```

entity decoder is
port( decoder_in : in std_logic_vector(11 downto 0);
      decoder_c   : in std_logic;
      decoder_z   : in std_logic;
      decoder_pc_inc    : out std_logic;
      decoder_pc_ld     : out std_logic;
      decoder_ibufloe  : out std_logic;
      decoder_aregld   : out std_logic;
      decoder_abufloe  : out std_logic;
      decoder_bregld   : out std_logic_vector(4 downto 0);
      decoder_bbbufloe : out std_logic_vector(4 downto 0);
      decoder_alu      : out std_logic_vector(2 downto 0);
      decoder_argout   : out std_logic_vector(7 downto 0));
end decoder;

```

**architecture behaviour of decoder is**

## -- Signals --

```
signal  opcode : std_logic_vector(3 downto 0);
signal  argument : std_logic_vector(7 downto 0);
signal  o0, o1, o2, o3: std_logic;
```

begin

```
opcode <= decoder_in(11 downto 8);  
argument <= decoder_in(7 downto 0);  
o0 <= opcode(0);  
o1 <= opcode(1);  
o2 <= opcode(2);  
o3 <= opcode(3);
```

```

decoder_pc_ld <=
    ((NOT o3) AND (NOT o2) AND (NOT o1) AND o0) OR
    ((NOT o3) AND (NOT o2) AND o1 AND (NOT o0)) OR
    ((NOT o3) AND (NOT o2) AND o1 AND o0 AND decoder_z) OR
    ((NOT o3) AND o2 AND (NOT o1) AND (NOT o0) AND decoder_c);

```

-- *Jp* #  
-- *Jp* *Ri*  
-- *Bz*  
-- *Bc*

**decoder\_pc\_inc** <= NOT( ((NOT o3) AND (NOT o2) AND (NOT o1) AND o0) OR --- Not Jp #

```

((NOT o3) AND (NOT o2) AND o1 AND (NOT o0)) OR          --- Not Jp Ri
((NOT o3) AND (NOT o2) AND o1 AND o0 AND decoder_z) OR    --- Not Bz
((NOT o3) AND o2 AND (NOT o1) AND (NOT o0) AND decoder_c)); --- Not Bc

decoder_ibufoe <=
  ((NOT o3) AND (NOT o2) AND (NOT o1) AND o0) OR          --- Jp #
  ((NOT o3) AND (NOT o2) AND o1 AND o0) OR                 --- Bz
  ((NOT o3) AND o2 AND (NOT o1) AND (NOT o0)) OR          --- Bc
  ((NOT o3) AND o2 AND (NOT o1) AND o0) OR                 --- Ld #
  (o3 AND (NOT o2) AND (NOT o1) AND (NOT o0)) OR          --- ADD #
  (o3 AND (NOT o2) AND o1 AND (NOT o0)) OR                 --- XOR #
  (o3 AND o2 AND (NOT o1) AND (NOT o0));                   --- AND #

decoder_aregld <=
  ((NOT o3) AND o2 AND (NOT o1) AND o0) OR          --- Ld #
  ((NOT o3) AND o2 AND o1 AND (NOT o0)) OR          --- Ld Ri
  (o3 AND (NOT o2) AND (NOT o1) AND (NOT o0)) OR      --- ADD #
  (o3 AND (NOT o2) AND (NOT o1) AND o0) OR          --- ADD Ri
  (o3 AND (NOT o2) AND o1 AND (NOT o0)) OR          --- XOR #
  (o3 AND (NOT o2) AND o1 AND o0) OR                 --- XOR Ri
  (o3 AND o2 AND (NOT o1) AND (NOT o0)) OR          --- AND #
  (o3 AND o2 AND (NOT o1) AND o0);                   --- AND Ri

decoder_abufoe <=
  ((NOT o3) AND o2 AND o1 AND o0);                   --- st Ri

decoder_bregld(4) <=
  ((NOT o3) AND o2 AND o1 AND o0);                   --- st Ri

decoder_bregld(3) <= argument(3);
decoder_bregld(2) <= argument(2);
decoder_bregld(1) <= argument(1);
decoder_bregld(0) <= argument(0);

decoder_bbbufoe(4) <=
  (NOT o3) AND (NOT o2) AND o1 AND (NOT o0)) OR          --- jp Ri
  ((NOT o3) AND o2 AND o1 AND (NOT o0)) OR          --- ld Ri
  (o3 AND (NOT o2) AND (NOT o1) AND o0) OR          --- ADD Ri
  (o3 AND (NOT o2) AND o1 AND o0) OR                 --- XOR Ri
  (o3 AND o2 AND (NOT o1) AND o0);                   --- AND Ri

decoder_bbbufoe(3) <= argument(3);
decoder_bbbufoe(2) <= argument(2);
decoder_bbbufoe(1) <= argument(1);
decoder_bbbufoe(0) <= argument(0);

decoder_alu(2) <=
  (o3 AND o2 AND o1 AND o0) OR          --- Clr c
  (o3 AND (NOT o2) AND (NOT o1) AND (NOT o0)) OR      --- ADD #
  (o3 AND (NOT o2) AND (NOT o1) AND o0);               --- ADD Ri

decoder_alu(1) <=
  (o3 AND o2 AND (NOT o1) AND (NOT o0)) OR          --- AND #
  (o3 AND o2 AND (NOT o1) AND o0) OR                 --- AND Ri
  (o3 AND o2 AND o1 AND (NOT o0));                   --- Set c

decoder_alu(0) <=
  (o3 AND (NOT o2) AND (NOT o1) AND (NOT o0)) OR      --- ADD #
  (o3 AND (NOT o2) AND (NOT o1) AND o0) OR          --- ADD Ri
  (o3 AND (NOT o2) AND o1 AND (NOT o0)) OR          --- XOR #
  (o3 AND (NOT o2) AND o1 AND o0) OR                 --- XOR Ri
  (o3 AND o2 AND o1 AND (NOT o0));                   --- Set c

decoder_argout <= argument;

end architecture;

```

### B.3.2. ALU

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity alu is
  port ( alu_A : in std_logic_vector(7 downto 0);      -- input A
         alu_B : in std_logic_vector(7 downto 0);      -- input B
         opcode : in std_logic_vector(2 downto 0);      -- opcode
         alu_clk: in std_logic;                         -- clk
         alu_c  : out std_logic;                        -- flag C
         alu_z  : out std_logic;                        -- flag Z
         alu_y  : out std_logic_vector(7 downto 0));    -- Output result y
end entity alu;

architecture behaviour of alu is
signal op0, op1, op2, c: std_logic;
signal alu_add: std_logic_vector(8 downto 0);

begin
  with opcode select
    alu_y <= alu_A
      when "000",      -- pass input_A
      alu_A xor alu_B
      when "001",      -- xor
      alu_A and alu_B
      when "010",      -- and
      "00000000"
      when "011",      -- set C
      "00000000"
      when "100",      -- clr C
      alu_A + alu_B
      when "101",      -- add
      "00000000"
      when others;

  op0 <= opcode(0);
  op1 <= opcode(1);
  op2 <= opcode(2);

  alu_add <= ((0" & (alu_A)) + (alu_B));

  process(alu_clk)
  begin
    if (alu_clk'event and alu_clk='1') then
      if ((op2='1') and (op1='0') and (op0='0')) or
        ((op2='1') and (op1='0') and (op0='1') and
        ((alu_add) <= "11111111")) then
        alu_c <= '0';
      elsif ((op2='0') and (op1='1') and (op0='1')) or
        ((op2='1') and (op1='0') and (op0='1') and
        ((alu_add) > "11111111")) then
        alu_c <= '1';
      end if;
    end if;
  end process;

  process(alu_clk)
  begin
    if (alu_clk'event and alu_clk='1') then
      if (((alu_A and alu_B) = "00000000") and (op0='0') and (op1='1') and (op2='0')) then
        alu_z <= '1';
      else

```

```

        alu_z <= '0';
    end if;
end if;
end process;

end architecture;

```

### B.3.3. PROGRAM COUNTER

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity pc_counter is
port ( pc_in : in std_logic_vector(7 downto 0);
       pc_inc : in std_logic;
       pc_ld : in std_logic;
       pc_rst : in std_logic;
       pc_en : in std_logic;
       pc_out : out std_logic_vector(7 downto 0));
end entity pc_counter;

architecture behaviour of pc_counter is
signal count : std_logic_vector(7 downto 0);
begin
begin
process(pc_rst, pc_en)
begin
if (pc_rst='1') then
count <= "00000000";
else
if (pc_en'event AND pc_en='1') then
if (pc_inc='1') then
count <= unsigned(count) + 1;
elsif (pc_ld='1') then
count <= pc_in;
end if;
end if;
end if;
end process;

pc_out <= count;
end architecture;

```

### B.3.4. TESTBENCH

```

library IEEE;
use IEEE.std_logic_1164.ALL;

entity cpu_tb is
end cpu_tb;

architecture behaviour of cpu_tb is
component cpu
port(cpu_RST:  in      std_logic;
      cpu_EN:   in      std_logic;
      cpu_IN:   in      std_logic_vector(7 downto 0);
      cpu_INSTR: in      std_logic_vector(11 downto 0);

```

```
cpu_pc: out      std_logic_vector(7 downto 0);
cpu_out:   out      std_logic_vector(7 downto 0));
end component;

signal cpu_RST, cpu_en: std_logic;
signal cpu_in, cpu_pc, cpu_out :      std_logic_vector(7 downto 0);
signal cpu_instr :      std_logic_vector(11 downto 0);

begin
lbl1:  cpu port map (cpu_RST, cpu_en, cpu_in, cpu_instr, cpu_pc, cpu_out);
cpu_RST <= '1' after 0 ns,
          '0' after 50 ns;
cpu_en <= '1' after 0 ns,
          '0' after 10 ns when cpu_en/='0' else '1' after 40 ns;
cpu_in <= "00000000" after 0 ns,
          "00000010" after 950 ns;
cpu_instr <= "000000000000" after 0 ns,
          "010100001100" after 50 ns,
          "011100000001" after 100 ns,
          "010100001100" after 150 ns,
          "011100000010" after 200 ns,
          "010100000110" after 250 ns,
          "011100000011" after 300 ns,
          "010100001100" after 350 ns,
          "0111000000100" after 400 ns,
          "0101000010000" after 450 ns,
          "0111000000101" after 500 ns,
          "010100000001" after 550 ns,
          "0111000000110" after 600 ns,
          "0101000000001" after 650 ns,
          "0111000000111" after 700 ns,
          "010111111111" after 750 ns,
          "0111000001010" after 800 ns,
          "000100010001" after 850 ns,
          "1111000000000" after 900 ns,
          "0110000000001" after 950 ns,
          "101000001000" after 1000 ns,
          "100011111000" after 1050 ns,
          "0111000000001" after 1100 ns,
          "010000010111" after 1150 ns,
          "000100010001" after 1200 ns;

end architecture;
```

## B.4. BUFFERS

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_2 is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic_vector(4 downto 0);
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_2;

architecture behaviour of buf_2 is
begin
process (buf_in, buf_oe)
begin
  if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=2)) then
    buf_out <= buf_in;
  else
    buf_out <= "ZZZZZZZZ";
  end if;
end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_3 is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic_vector(4 downto 0);
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_3;

architecture behaviour of buf_3 is
begin
process (buf_in, buf_oe)
begin
  if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=3)) then
    buf_out <= buf_in;
  else
    buf_out <= "ZZZZZZZZ";
  end if;
end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_4 is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic_vector(4 downto 0);
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_4;

architecture behaviour of buf_4 is

```

```

begin
process (buf_in , buf_oe)
begin
    if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=4)) then
        buf_out <= buf_in ;
    else
        buf_out <= "ZZZZZZZZ";
    end if;
end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_5 is
port( buf_in : in std_logic_vector(7 downto 0);
       buf_oe : in std_logic_vector(4 downto 0);
       buf_out : out std_logic_vector(7 downto 0));
end entity buf_5;

architecture behaviour of buf_5 is
begin
    process (buf_in , buf_oe)
    begin
        if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=5)) then
            buf_out <= buf_in ;
        else
            buf_out <= "ZZZZZZZZ";
        end if;
    end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_6 is
port( buf_in : in std_logic_vector(7 downto 0);
       buf_oe : in std_logic_vector(4 downto 0);
       buf_out : out std_logic_vector(7 downto 0));
end entity buf_6;

architecture behaviour of buf_6 is
begin
    process (buf_in , buf_oe)
    begin
        if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=6)) then
            buf_out <= buf_in ;
        else
            buf_out <= "ZZZZZZZZ";
        end if;
    end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;

```

```

use IEEE.numeric_std.all;

entity buf_7 is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic_vector(4 downto 0);
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_7;

architecture behaviour of buf_7 is
begin
  process (buf_in, buf_oe)
  begin
    if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=7)) then
      buf_out <= buf_in;
    else
      buf_out <= "ZZZZZZZZ";
    end if;
  end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_8 is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic_vector(4 downto 0);
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_8;

architecture behaviour of buf_8 is
begin
  process (buf_in, buf_oe)
  begin
    if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=8)) then
      buf_out <= buf_in;
    else
      buf_out <= "ZZZZZZZZ";
    end if;
  end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_9 is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic_vector(4 downto 0);
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_9;

architecture behaviour of buf_9 is
begin
  process (buf_in, buf_oe)
  begin
    if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=9)) then

```

```

        buf_out <= buf_in;
    else
        buf_out <= "ZZZZZZZZ";
    end if;
end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_10 is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic_vector(4 downto 0);
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_10;

architecture behaviour of buf_10 is
begin
process (buf_in, buf_oe)
begin
    if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=10)) then
        buf_out <= buf_in;
    else
        buf_out <= "ZZZZZZZZ";
    end if;
end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;

entity buf_a is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic;
      buf_out : out std_logic_vector(7 downto 0));
end entity buf_a;

architecture behaviour of buf_a is
begin
process (buf_in, buf_oe)
begin
    if (buf_oe = '1') then
        buf_out <= buf_in;
    else
        buf_out <= "ZZZZZZZZ";
    end if;
end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;

entity buf_arg is
port( buf_in : in std_logic_vector(7 downto 0);
      buf_oe : in std_logic;
      buf_out : out std_logic_vector(7 downto 0));

```

```

end entity buf_arg;

architecture behaviour of buf_arg is
begin
  process (buf_in, buf_oe)
  begin
    if (buf_oe = '1') then
      buf_out <= buf_in;
    else
      buf_out <= "ZZZZZZZZ";
    end if;
  end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity buf_i is
port( buf_in : in std_logic_vector(7 downto 0);
       buf_oe : in std_logic_vector(4 downto 0);
       buf_out : out std_logic_vector(7 downto 0));
end entity buf_i;

architecture behaviour of buf_i is
begin
  process (buf_in, buf_oe)
  begin
    if (buf_oe(4) = '1' AND (unsigned(buf_oe(3 downto 0))=1)) then
      buf_out <= buf_in;
    else
      buf_out <= "ZZZZZZZZ";
    end if;
  end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity instr_buf is
port( buf_in : in std_logic_vector(11 downto 0);
       buf_oe : in std_logic;
       buf_out : out std_logic_vector(11 downto 0));
end entity instr_buf;

architecture behaviour of instr_buf is
begin
  process (buf_in, buf_oe)
  begin
    if (buf_oe = '1') then
      buf_out <= buf_in;
    else
      buf_out <= "ZZZZZZZZZZZZ";
    end if;
  end process;
end architecture;

```

## B.5. REGISTERS

```

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

entity reg_2 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_2;

architecture behaviour of reg_2 is
begin
  process(reg_clk, reg_rst)
  begin
    if (reg_rst = '1') then
      reg_out <= "00000000";
    elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=2)) then
      reg_out <= reg_in;
    end if;
  end process;
end architecture;

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

entity reg_3 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_3;

architecture behaviour of reg_3 is
begin
  process(reg_clk, reg_rst)
  begin
    if (reg_rst = '1') then
      reg_out <= "00000000";
    elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=3)) then
      reg_out <= reg_in;
    end if;
  end process;
end architecture;

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

entity reg_4 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;

```

```

        reg_rst : in  std_logic;
        reg_ld  : in  std_logic_vector(4 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end entity reg_4;

architecture behaviour of reg_4 is
begin
  process(reg_clk, reg_rst)
  begin
    if (reg_rst = '1') then
      reg_out <= "00000000";
    elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=4)) then
      reg_out <= reg_in;
    end if;
  end process;
end architecture;

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

entity reg_5 is
port ( reg_in  : in  std_logic_vector(7 downto 0);
       reg_clk : in  std_logic;
       reg_rst : in  std_logic;
       reg_ld  : in  std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_5;

architecture behaviour of reg_5 is
begin
  process(reg_clk, reg_rst)
  begin
    if (reg_rst = '1') then
      reg_out <= "00000000";
    elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=5)) then
      reg_out <= reg_in;
    end if;
  end process;
end architecture;

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

entity reg_5 is
port ( reg_in  : in  std_logic_vector(7 downto 0);
       reg_clk : in  std_logic;
       reg_rst : in  std_logic;
       reg_ld  : in  std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_5;

architecture behaviour of reg_5 is
begin
  process(reg_clk, reg_rst)
  begin
    if (reg_rst = '1') then
      reg_out <= "00000000";
    elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=6)) then
      reg_out <= reg_in;
    end if;
  end process;
end architecture;

```

```

if (reg_rst = '1') then
    reg_out <= "00000000";
elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=5)) then
    reg_out <= reg_in;
end if;
end process;
end architecture;

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

entity reg_6 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_6;

architecture behaviour of reg_6 is
begin
process(reg_clk, reg_rst)
begin
    if (reg_rst = '1') then
        reg_out <= "00000000";
    elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=6)) then
        reg_out <= reg_in;
    end if;
end process;
end architecture;

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

entity reg_7 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_7;

architecture behaviour of reg_7 is
begin
process(reg_clk, reg_rst)
begin
    if (reg_rst = '1') then
        reg_out <= "00000000";
    elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=7)) then
        reg_out <= reg_in;
    end if;
end process;
end architecture;

```

```

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

entity reg_8 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_8;

architecture behaviour of reg_8 is
begin
process(reg_clk, reg_rst)
begin
  if (reg_rst = '1') then
    reg_out <= "00000000";
  elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=8)) then
    reg_out <= reg_in;
  end if;
end process;
end architecture;

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

entity reg_9 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_9;

architecture behaviour of reg_9 is
begin
process(reg_clk, reg_rst)
begin
  if (reg_rst = '1') then
    reg_out <= "00000000";
  elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=9)) then
    reg_out <= reg_in;
  end if;
end process;
end architecture;

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

entity reg_10 is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);

```

```

    reg_out : out std_logic_vector(7 downto 0));
end entity reg_10;

architecture behaviour of reg_10 is
begin
process(reg_clk, reg_rst)
begin
  if (reg_rst = '1') then
    reg_out <= "00000000";
  elsif (reg_clk'event AND reg_clk='1' AND reg_ld(4)='1' AND (unsigned(reg_ld(3 downto 0))=10)) then
    reg_out <= reg_in;
  end if;
end process;
end architecture;

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

```

--Register A (RA)--

```

entity reg_a is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic;
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_a;

architecture behaviour of reg_a is
begin
process(reg_clk, reg_rst)
begin
  if (reg_rst = '1') then
    reg_out <= "00000000";
  elsif (reg_clk'event AND reg_clk='1' AND reg_ld='1') then
    reg_out <= reg_in;
  end if;
end process;
end architecture;

```

```

library IEEE;
use ieee.std_logic_1164.all;

entity reg_cluster is
port ( reg_in      : in std_logic_vector(7 downto 0);
       reg_clk     : in std_logic;
       reg_rst     : in std_logic;
       reg_select  : in std_logic_vector(4 downto 0);
       buf_select  : in std_logic_vector(4 downto 0);
       buf_in      : in std_logic_vector(7 downto 0);
       buf_out     : out std_logic_vector(7 downto 0);
       reg_out     : out std_logic_vector(7 downto 0));
end entity reg_cluster;

```

```

architecture behavioural of reg_cluster is

--components--

--registers--

component reg_0
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld : in std_logic_vector(4 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end component;

component reg_2
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld : in std_logic_vector(4 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end component;

component reg_3
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld : in std_logic_vector(4 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end component;

component reg_4
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld : in std_logic_vector(4 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end component;

component reg_5
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld : in std_logic_vector(4 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end component;

component reg_6
port ( reg_in : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld : in std_logic_vector(4 downto 0);
        reg_out : out std_logic_vector(7 downto 0));
end component;

component reg_7
port ( reg_in : in std_logic_vector(7 downto 0);

```

```

    reg_clk : in std_logic;
    reg_rst : in std_logic;
    reg_ld   : in std_logic_vector(4 downto 0);
    reg_out  : out std_logic_vector(7 downto 0));
end component;

component reg_8
port ( reg_in   : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld   : in std_logic_vector(4 downto 0);
        reg_out  : out std_logic_vector(7 downto 0));
end component;

component reg_9
port ( reg_in   : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld   : in std_logic_vector(4 downto 0);
        reg_out  : out std_logic_vector(7 downto 0));
end component;

component reg_10
port ( reg_in   : in std_logic_vector(7 downto 0);
        reg_clk : in std_logic;
        reg_rst : in std_logic;
        reg_ld   : in std_logic_vector(4 downto 0);
        reg_out  : out std_logic_vector(7 downto 0));
end component;

--buffers--

component buf_i
port ( buf_in   : in std_logic_vector(7 downto 0);
        buf_oe   : in std_logic_vector(4 downto 0);
        buf_out  : out std_logic_vector(7 downto 0));
end component;

component buf_2
port ( buf_in   : in std_logic_vector(7 downto 0);
        buf_oe   : in std_logic_vector(4 downto 0);
        buf_out  : out std_logic_vector(7 downto 0));
end component;

component buf_3
port ( buf_in   : in std_logic_vector(7 downto 0);
        buf_oe   : in std_logic_vector(4 downto 0);
        buf_out  : out std_logic_vector(7 downto 0));
end component;

component buf_4
port ( buf_in   : in std_logic_vector(7 downto 0);
        buf_oe   : in std_logic_vector(4 downto 0);
        buf_out  : out std_logic_vector(7 downto 0));
end component;

```



```

        reg_ld => reg_select ,
        reg_out => reg_out);

r2:      reg_2  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,
        reg_out => reg_buf_2);

r3:      reg_3  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,
        reg_out => reg_buf_3);

r4:      reg_4  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,
        reg_out => reg_buf_4);

r5:      reg_5  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,
        reg_out => reg_buf_5);

r6:      reg_6  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,
        reg_out => reg_buf_6);

r7:      reg_7  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,
        reg_out => reg_buf_7);

r8:      reg_8  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,
        reg_out => reg_buf_8);

r9:      reg_9  port map (
        reg_in => reg_in ,
        reg_clk => reg_clk ,
        reg_rst => reg_rst ,
        reg_ld => reg_select ,

```

```

        reg_out => reg_buf_9);

r10:      reg_10 port map (
        reg_in => reg_in,
        reg_clk => reg_clk,
        reg_rst => reg_rst,
        reg_ld => reg_select,
        reg_out => reg_buf_10);

--buffers--

bin:       buf_i port map (
        buf_in => buf_in,
        buf_oe => buf_select,
        buf_out => buf_out);

b2:       buf_2 port map (
        buf_in => reg_buf_2,
        buf_oe => buf_select,
        buf_out => buf_out);

b3:       buf_3 port map (
        buf_in => reg_buf_3,
        buf_oe => buf_select,
        buf_out => buf_out);

b4:       buf_4 port map (
        buf_in => reg_buf_4,
        buf_oe => buf_select,
        buf_out => buf_out);

b5:       buf_5 port map (
        buf_in => reg_buf_5,
        buf_oe => buf_select,
        buf_out => buf_out);

b6:       buf_6 port map (
        buf_in => reg_buf_6,
        buf_oe => buf_select,
        buf_out => buf_out);

b7:       buf_7 port map (
        buf_in => reg_buf_7,
        buf_oe => buf_select,
        buf_out => buf_out);

b8:       buf_8 port map (
        buf_in => reg_buf_8,
        buf_oe => buf_select,
        buf_out => buf_out);

b9:       buf_9 port map (
        buf_in => reg_buf_9,
        buf_oe => buf_select,
        buf_out => buf_out);

```

```

b10:           buf_10  port map (
                           buf_in => reg_buf_10,
                           buf_oe => buf_select,
                           buf_out => buf_out);

end architecture;

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

entity reg_o is
port ( reg_in : in std_logic_vector(7 downto 0);
       reg_clk : in std_logic;
       reg_rst : in std_logic;
       reg_ld : in std_logic_vector(4 downto 0);
       reg_out : out std_logic_vector(7 downto 0));
end entity reg_o;

architecture behaviour of reg_o is
begin
process(reg_clk, reg_rst)
begin
  if (reg_rst = '1') then
    reg_out <= "00000000";
  elsif (reg_clk'event AND (reg_clk='1') AND (reg_ld(4)='1') AND (unsigned(reg_ld(3 downto 0))=1))
    reg_out <= reg_in;
  end if;
end process;
end architecture;

library IEEE;
use IEEE.std_logic_1164.ALL;

entity shift_reg is
port( clk:          in std_logic;
      reset:        in std_logic;
      shift_in:     in std_logic;
      output:       out std_logic_vector (7 downto 0)
      );
end shift_reg;

architecture behaviour of shift_reg is

signal reg_shift,shifted_reg: std_logic_vector (7 downto 0);

begin

shifted_reg(7 downto 1) <= reg_shift(6 downto 0);
shifted_reg(0) <= shift_in;

process(clk,reset)
begin
  if(reset = '1') then
    reg_shift <= (others => '0');
  end if;
  if (clk'event AND (clk='1')) then
    reg_shift <= shifted_reg;
  end if;
end process;

```

```
    else
        if(falling_edge(clk)) then
            reg_shift <= shifted_reg;
        end if;
    end if;
end process;
```

```
    output <= reg_shift;
end behaviour;
```

## B.6. GATE

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

```
entity gate is
port(  input : in std_logic;
          output : out std_logic;
          sw      : in std_logic);
end gate;
```

```
architecture Behavioral of gate is
begin
```

```
    output <= input when (sw = '1') else '0';
```

```
    end Behavioral;
```

## B.7. GPU

```

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

entity gpu is
port (
clk : in std_logic;
gpu_in : in std_logic_vector(7 downto 0);
reset : in std_logic;
ssclk : out std_logic;
mosi : out std_logic;
miso : in std_logic;
cs : out std_logic;
test_xbuf : out std_logic_vector(5 downto 0);
test_ybuf : out std_logic_vector(5 downto 0);
test_data : out std_logic_vector(7 downto 0)
);
end gpu;

architecture behavioural of gpu is

-- definition of state machine states
type statetype is ( state0 ,
state1 ,
state2 ,
state3 ,
state4 ,
state5 ,
state6 ,
state7 ,
state8 ,
state9 ,
state10 ,
state11 ,
state12 ,
state13 ,
state14
);

signal state , next_state : statetype;

-- definition of x and y coordinate buffers
signal x_buf, y_buf : std_logic_vector(5 downto 0);

-- definition of z vector
signal z : std_logic_vector(15 downto 0);

-- non used signal
signal sclk : std_logic;

signal instruction : std_logic_vector(23 downto 0);
signal instruction8 : std_logic_vector(7 downto 0);
signal send : std_logic_vector(31 downto 0);

```

```

signal data : std_logic_vector(7 downto 0);
signal bitplace : unsigned(3 downto 0);
signal sig : std_logic_vector(7 downto 0);

signal count, new_count : unsigned (10 downto 0);

begin
test_xbuf <= x_buf;
test_ybuf <= y_buf;
test_data <= sig;
ssclk <= sclk;

sig <= data;

-- Staterenew
process(clk, reset)
begin
if (reset = '1') then
state <= state0;
else
if (rising_edge(clk)) then
state <= next_state;
else
state <= state;
end if;
end if;
end process;
-- counter 1
process (clk)
begin
if (rising_edge (clk)) then
if (reset = '1') then
count <= (others => '0');
else
count <= new_count;
end if;
else
end if;
end process;

-- Finite state machine
process(gpu_in, clk)
begin
case state is
when state0 =>
new_count <= to_unsigned(0,11);
x_buf <= "ZZZZZZ";
y_buf <= "ZZZZZZ";
if(gpu_in(7 downto 6) = "11") then -- clr screen
next_state <= state13;
elsif (gpu_in(7 downto 6) = "01") then -- go
x_buf <= gpu_in(5 downto 0);
next_state <= state1;
else
next_state <= state0;
end if;

```

```

-- Output define
sclk <= '0';
mosi <= '0';
--data <= "XXXXXXXX";
cs <= '1';

when state1 =>
    if(gpu_in(7 downto 6) = "10") then
        y_buf <= gpu_in(5 downto 0);
        next_state <= state2;
    else
        next_state <= state1;
    end if;
-- Output define
sclk <= '0';
mosi <= '0';
-- reset counters
new_count <= to_unsigned(0,11);
cs <= '1';

when state2 =>
instruction(23 downto 16) <= "00000011"; -- read instruction
instruction(15 downto 0) <= std_logic_vector(resize(unsigned(x_buf) / 8 + 5 * unsigned(y_buf), 16));
next_state <= state3;
new_count <= to_unsigned(0,11);

when state3 =>
cs <= '0';
sclk <= '0';
if (count < 23) then
mosi <= instruction(to_integer(23 - count));
new_count <= count;
next_state <= state4;
--data <= std_logic_vector(resize(count, 8));
elsif (count = 23) then
mosi <= instruction(to_integer(23 - count));
new_count <= count;
next_state <= state4;
else
new_count <= to_unsigned(0,11);
next_state <= state5;
end if;

when state4 =>
sclk <= '1';
new_count <= count + 1;
next_state <= state3;

when state5 =>
sclk <= '0';
mosi <= '0';
if (count < 8) then
data(to_integer(7 - count)) <= miso;
new_count <= count;
next_state <= state6;

```

```

else
next_state <= state7;
new_count <= to_unsigned(0,11);
end if;

when state6 =>
sclk <= '1';
new_count <= count + 1;
next_state <= state5;

when state7 =>
cs <= '1';
--bitplace <= unsigned(unsigned(x_buf) - ((unsigned(x_buf) / 8) * 8));
new_count <= to_unsigned(0,11);
next_state <= state8;

when state8 =>
data(to_integer(7 - unsigned(unsigned(x_buf) - ((unsigned(x_buf) / 8) * 8)))) <= '1';
new_count <= to_unsigned(0,11);
next_state <= state9;

when state9 =>
send(31 downto 24) <= "00000010";
send(23 downto 8) <= std_logic_vector(resize(unsigned(x_buf) / 8 + 5 * unsigned(y_buf), 16));
send(7 downto 0) <= data;
new_count <= to_unsigned(0,11);
next_state <= state10;

when state10 =>
cs <= '0';
sclk <= '0';
if (count < 31 and count /= 31) then
mosi <= send(to_integer(31 - count));
new_count <= count;
next_state <= state11;
elsif (count = 31) then
mosi <= send(to_integer(31 - count));
new_count <= count;
next_state <= state11;
else
new_count <= to_unsigned(0,11);
next_state <= state12;
end if;

when state11 =>
sclk <= '1';
new_count <= count + 1;
next_state <= state10;

when state12 =>
cs <= '1';
new_count <= to_unsigned(0,11);
next_state <= state0;

when state13 =>
cs <= '0';

```

```
sclk <= '0';
instruction8 <= "00000010";
if (count < 8) then
mosi <= instruction8(to_integer(7 - count));
new_count <= count;
next_state <= state14;
elsif (count >= 7 and count < 1280) then
new_count <= count;
mosi <= '0';
next_state <= state14;
else
mosi <= '0';
next_state <= state0;
end if;

when state14 =>
sclk <= '1';
new_count <= count + 1;
next_state <= state13;

when others =>
new_count <= to_unsigned(0,11);
next_state <= state0;

end case;
end process;
end architecture;
```

## B.8. CALCULATOR

```

library IEEE;
use IEEE.std_logic_1164.ALL;

entity calculator is
    port(XI           :      in      std_logic;
          XO           : inout std_logic;
          top_reset:in   std_logic;
          top_sclk :in   std_logic;
          top_mosi :in   std_logic;
          top_ss   :in   std_logic;
          top_out  :out  std_logic_vector(7 downto 0));
end calculator;

architecture behaviour of calculator is

--components--
component osc10
    PORT(E: IN STD_LOGIC;
          F: OUT STD_LOGIC;
          XI: IN STD_LOGIC;
          XO: INOUT STD_LOGIC := 'Z');
END component;

component cpu
    port(  cpu_RST :      in      std_logic;
           cpu_en   :      in      std_logic;
           cpu_in    :      in      std_logic_vector(7 downto 0);
           cpu_instr:      in      std_logic_vector(11 downto 0);
           cpu_pc   :      out     std_logic_vector(7 downto 0);
           cpu_out  :      out     std_logic_vector(7 downto 0));
end component;

component gate
    port(      input : in std_logic;
              output : out std_logic;
              sw     : in std_logic
            );
end component;

component rom
    port(
        rom_a:      in std_logic_vector (7 DOWNTO 0);      -- address
        rom_d:      out std_logic_vector (11 DOWNTO 0)      -- instruction
    );
end component;

component shift_reg
    port(  clk:          in std_logic;
           reset:        in std_logic;
           shift_in:      in std_logic;
           output:       out std_logic_vector (7 downto 0)
    );
end component;

```

```

);
end component;

--Signals--

signal cpu_enable : std_logic;
signal cpu_input      : std_logic_vector(7 downto 0);
signal cpu_instr      : std_logic_vector(11 downto 0);
signal cpu_addr       : std_logic_vector(7 downto 0);
signal osc_clk        : std_logic;

--Portmaps

begin

lbl_osc:          osc10 port map(
    E            =>      '1',
    F            =>      osc_clk,
    XI           =>      XI,
    XO           =>      XO);

lbl_cpu:          cpu port map(
    cpu_RST      =>      top_reset,
    cpu_EN       =>      cpu_enable,
    cpu_IN       =>      cpu_input,
    cpu_INSTR    =>      cpu_instr,
    cpu_PC       =>      cpu_addr,
    cpu_OUT      =>      top_out);

lbl_gate:          gate port map(
    input        =>      osc_clk,
    output       =>      cpu_enable,
    sw           =>      top_ss);

lbl_rom:          rom port map(
    rom_A        =>      cpu_addr,
    rom_D        =>      cpu_instr);

lbl_shift:         shift_reg port map(
    clk          =>      osc_clk,
    reset        =>      top_reset,
    shift_in     =>      top_mosi,
    output       =>      cpu_input);

end behaviour;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb is
end tb;

architecture behavior of tb is

```

```
component calculator is
port(
    XI          :      in  std_logic;
    XO          :      inout std_logic;
    top_reset   :      in  std_logic;
    top_sclk    :      in  std_logic;
    top_mosi    :      in  std_logic;
    top_ss      :      in  std_logic;
    top_out     :      out std_logic_vector(7 downto 0)
);
end component;

signal XI, top_reset, top_sclk, top_mosi, top_ss : std_logic;
signal top_out : std_logic_vector(7 downto 0);

begin
lib01: calculator port map( XI, open, top_reset, top_sclk, top_mosi, top_ss, top_out);

XI <=  '0' after 0 ns,
      '1' after 170 ns when XI /= '1' else '0' after 170 ns;

top_mosi <=  '0' after 0 ns;
top_sclk <=  '0' after 0 ns;
top_reset <=  '1' after 0 ns,
              '0' after 200 ns;
top_ss <=  '0' after 0 ns;

end behavior;
```

## B.9. ASSAMBL