A remote programmable FPGA-LED cube

With the “European Digital Virtual Design Lab”

Bachelor in de Elektronica-ICT
Afstudeerrichting Elektronica

Thys Sam

Academiejaar 2013-2014

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

VOORWOORD

The second semester of the final year of my education “professional bachlor Elektronics-ICT “comprises of an internship and making a thesis.

The Electronics lab of Thomas More Kempen, campus Geel offerd me an internship, with the goal to create a "remote programmable embedded system", which makes it possible to remotely program, operate and monitor an FPGA.

I hereby would like to thank the following people for their help and for ensuring the proper conduct of the internship.

Ing. Friant Luc and Ing. Dielens Patrick their supervision during the internship and Ing. Mertes Gert for providing the informaton and help on eDiViDe.

And also thanks to the other teachers who have helped me during the internship.

SAMENVATTING

The teachers of the department of Electronics wanted a system where it is possible for students to practice on programming FPGAs with VHDL code at home without the need for students to purchase development boards. For educational purposes they want to connect a LED cube to the FPGA to practice on.

This project is made by using an existing system, “eDiViDe”, developed In the KHLim in Limburg. This system makes it possible to remotely program and operate an FPGA over internet.

First we take a look at how "eDiViDe" works, and what the requirements are. Then we investigated whether the available material are suitable for use with this system.

Since we need to drive an existing LED cube with the FPGA, we need to figure out if and how they can be linked. Hereafter code is developed to drive the LED cube.

Then a web interface is designed to control and monitor the FPGA.

Finally, we create a linux server to connect the FGPA to. This server is responsible for the programming and the control of the development board.

INHOUDSTAFEL

INLEIDING

In the electronics lab at Thomas More kempen college, students learn a variety of technical subjects. The teachers of this department would like a system that would make pacticing at home a lot more accessible for students. One of the cources the students get is “programmable systems”, where students learn to program FPGA’s with VHDL. This thesis forcues on making a system that allows sutdents to remotely program an FPGA that stays at the school while the students are at home. To make it more exciting for the students, an FPGA-LED cube will be connected to the FGPA so students can practice at driving this LED cube in VHDL code.

Since there already exists a system whereby it’s prossible to remotely program an FPGA, we’ll make use of this system, called “eDiViDe”.

1 DE PROBLEEMSTELLING

1.1 Achtergrond

Thomas More Kempen is een hogeschool met een opleidingsafdeling Elektronica-ICT. De studenten die deze opleiding volgen, krijgen een vak genaamd "programmeerbare systemen", waarbij ze VHDL leren programmeren. Voor sommige studenten is de voorziene tijd tijdens de les niet genoeg om het programmeren in VHDL onder de knie te krijgen. Zodanig zou het een goede hulp zijn als deze studenten de mogelijkheid hebben om thuis te oefenen. Maar het materiaal dat nodig is om te oefenen is duur en het simuleren van de geschreven code biedt geen goede visualisatie en representatie van het te programmeren systeem. Zodus is er nood aan een oplossing waarbij er echt materiaal gebruikt kan worden en waarbij studenten zelf geen extra materiaal en/of software moeten aankopen. Hiervoor bestaat er reeds een platform dat ontwikkeld is door de Katholieke hogeschool Limburg, waaraan opstellingen met ontwikkelborden kunnen gekoppeld worden. Er zijn reeds enkele onderwijsinstellingen verdeeld in Europa die hier reeds een opstelling voor hebben gecreëerd en het labo elektronica van Thomas More Kempen wil hier graag ook een opstelling voor maken.

1.2 Probleemanalyse

Studenten moeten over de mogelijkheid beschikken om thuis te oefenen op het programmeren van VHDL. De bedoeling van de stageopdracht is een programmeerbare opstelling maken met behulp van een FPGA-ontwikkelbord en een 3D LED-kubus die lokaal op school blijft staan en die door leerlingen van op afstand over internet via een webinterface kan geprogrammeerd en bediend worden en waarbij feedback wordt voorzien via een webcam. Het programmeren van deze opstelling dient te gebeuren met behulp van "Edivide", waarbij studenten in een tekstverwerker het programma in VHDL schrijven en deze vervolgens uploaden via de webinterface. De oefeningen die de studenten dienen op te lossen en de oplossingen hiervoor moeten tijdens deze stageopdracht ook gemaakt worden.

1.3 Doelstellingen

Het einddoel van de stageopdracht is het creëren van een volledig programmeerbare en controleerbare FPGA-opstelling met het"Digilent spartan 3"-ontwikkelbord en een 3D LED-kubus met bijhorende oefeningen en oplossingen. Maar de eerste doelstelling tijdens de stageopdracht is het opstellen van een programmeeromgeving en het kunnen programmeren van het ontwikkelbord. Tijdens het volgende doel dient de LED-kubus aan dit ontwikkelbord gekoppeld te worden en dienen hiervoor oefeningen met bijhorende oplossingen geschreven te worden, hiervoor dient wel eerst een analyse gemaakt te worden van de werking van de LED-kubus. Vervolgens dient er een lokale server gemaakt en ingesteld te worden. Hierna wordt de FPGA-opstelling gekoppeld met een lokale server en lokaal geprogrammeerd en bediend. Wanneer dit doel geslaagd is kan ten slotte de lokale server samen met de opstelling geïntegreerd worden in "Edivide".

2 REALISATIE

2.1 Edivide opstelling

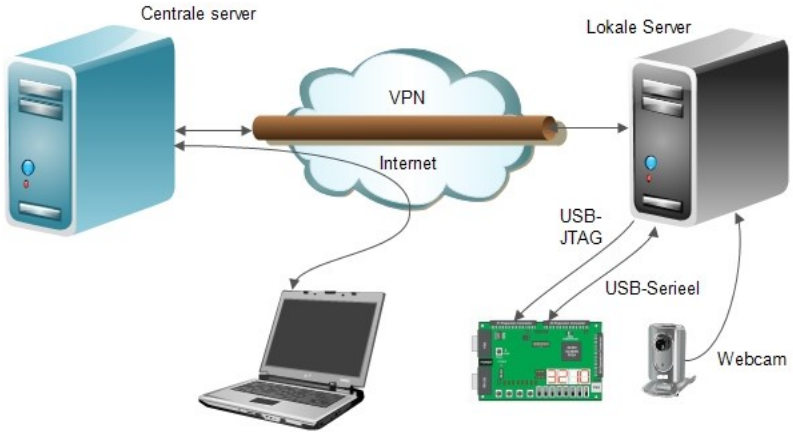
2.1.1 Wat is Edivide

"eDiViDe" stands for "European Digital Virtual Design Lab", a project developed in a collaboration between different college's in Europe within the Erasmus "Lifelong Learing Programme". The goal of the project was to develop a platform whereby each of the participating schools makes two easy and two hard FPGA setups available over internet. These FPGA setups are then available to program trough a webinterface and monitorable with a webcam.

2.1.2 Hoe werkt Edivide

2.1.2.1 Configuratie

The "eDiViDe" platorm exists of a Xilinx FPGA development board that is connected to a local server with USB-JTAG programmer to program the FPGA and a USB-Serial cable to drive and monitor inputs and outputs. The local server receives data packets from a webinterface with virtual inputs and outputs, these packets are then transmitted over a serial interface to the FPGA, where they are extracted and used as inputs to the user code. There's also a webcam connected to the local server to visually monitor the development board and the connected setup. The local server is connected with a VPN to a central server located at KHlim in Limburg, Belgium. Over this connection the local server receives information on what to do, like program the FPGA and sends back the video feed of the webcam and the received information from the serial connection with the development board. The central server hosts a webinterface where the user can connect to. This webinterface gives the user acces to reprogram, control and monitor the FPGA. The central server also contains a database with acces times to different FPGA setups, since these setups are real fysical setups, only one user at a time can use them.

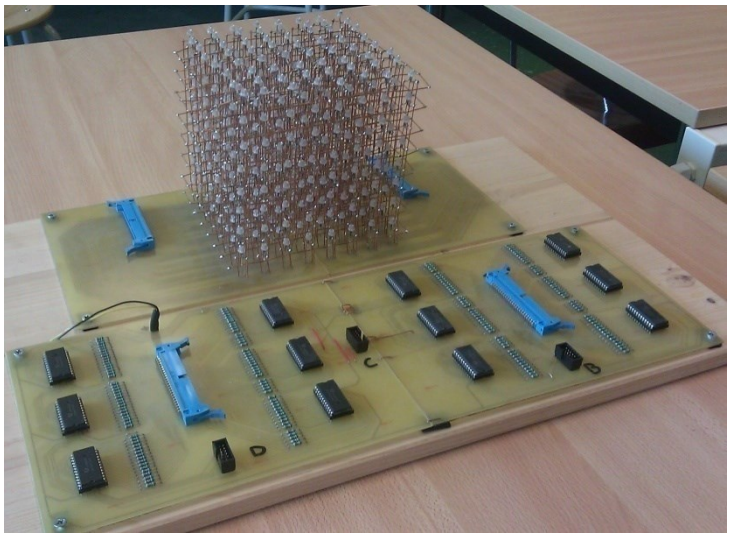


Figuur 1: eDiViDe structuur

2.2 Koppelen FPGA en LED-kubus

2.2.1 3D LED-Kubus

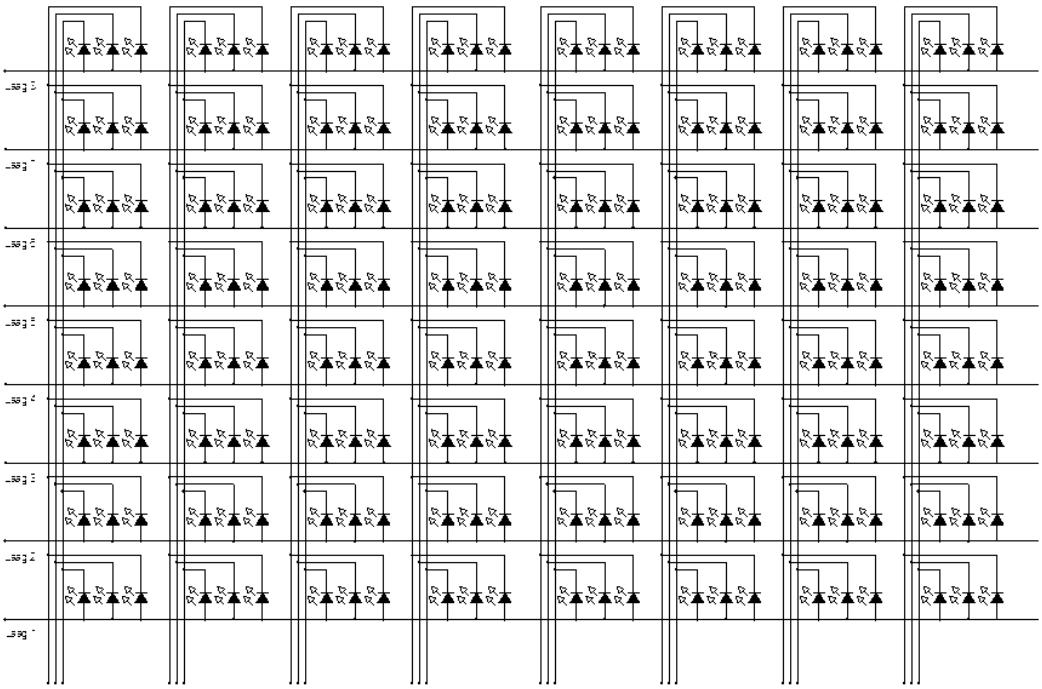
De The LED cube is an 8 x 8 x 8 matrix of 512 RGB LED's made by students in 2010 as an graduation project in their final year. The LED's can be driven by making use of multiplexing, each LED is driven for a fraction of a second to create the illusion of an image.



Figuur 2: Led-kubus

The original controller of the led cube is based on a microcontroller together with 12 decoders. To make sure the cube is still fully functional, the cube is tested with the original controller. After the working of the LED cube is confirmed, it's time to find out how the control the LED cube.

The LED's in the cube are ones with a common anode connection and 3 separate cathodes, one for each color. The cathodes of each separate color are vertically connected in a column and the anodes of each LED are horizontally connected with eachother.

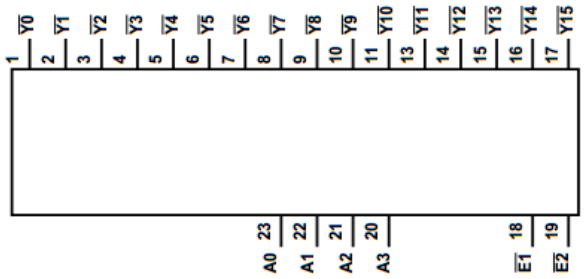


Figuur 3: Verbindingen van de led-kubus

The original controller of the led cube is based on a microcontroller together with 12 decoders. To make sure the cube is still fully functional, the cube is tested with the original controller. After the working of the LED cube is confirmed, it's time to find out how the control the LED cube.

The LED's in the cube are ones with a common anode connection and 3 separate cathodes, one for each color. The cathodes of each separate color are vertically connected in a column and the anodes of each LED are horizontally connected with each other.

This results in 200 inputs, 8 to select the horizontal level en 196 to vertically select a led and its color. Because 200 drive lines are too much connection for a microcontroller, the students who made de LED cube used 12 decoders to bring down the 196 drive lines back to 24 inputs, together with the 8 horizontal selection lines this makes 32 drive lines.



Figuur 4: Decoder IC - CD74HCT154E

They divided the 24 vertical selection inputs into three ports. In the image beneath ech square represents a LED in a horizontal plane and the background color represents the port, the letters and numbers represent the hexadecimal number that needs to be send to the port to get the color on the led in which the information is written.

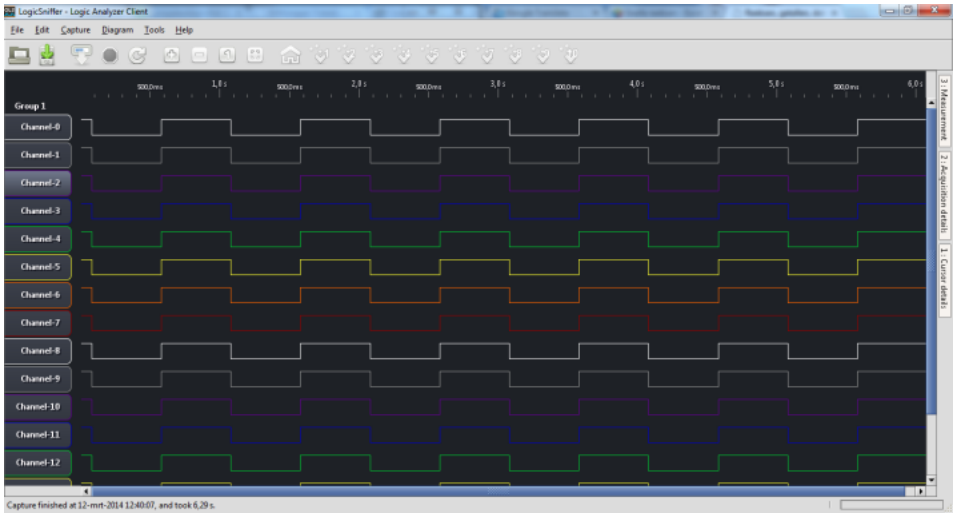
B0	B1	B3	B4	B6	B7	B9	BB	B4	B5	B7	B8	BA	BB	BD	BE
	B2		B5		B8		BA		B6		B9		BC		BF
BC	BD	BF	70	72	73	75	76	D8	D9	DB	DC	DE	DF	B1	B2
	BE		71		74		77		DA		DD		B0		B3
78	79	7B	7C	7E	7F	E1	E2	EC	ED	EF	D0	D2	D3	D5	D6
	7A		7D		E0		E3		EE		D1		D4		D7
E4	E5	E7	E8	EA	EB	ED	EE	E0	E1	E3	E4	E6	E7	E9	EA
	E6		E9		EC		EF		E2		E5		E8		EB
D0	D1	D3	D4	D6	D7	D9	DA	D4	D5	D7	D8	DA	DB	D0	DE
	D2		D5		D8		DB		D6		D9		DC		DF
DC	DD	DF	B0	B2	B3	B5	B6	E8	E9	EB	EC	EE	EF	D1	D2
	DE		B1		B4		B7		EA		ED		D0		D3
B8	B9	BB	BC	BE	BF	71	72	7C	7D	7F	E0	E2	E3	E5	E6
	BA		BD		70		73		7E		E1		E4		E7
74	75	77	78	7A	7B	7D	7E	70	71	73	74	76	77	79	7A
	76		79		7C		7F		72		75		78		7B

poort B
poort C
poort D

Figuur 5: Data tabel voor aansturing van de led's

2.2.2 FPGA Ontwikkelbor

The FPGA development board used is a Digilent Sparten 3 with a Xilinx FPGA, because ‘eDiViDe’ only supports Xilinx FPGA’s.

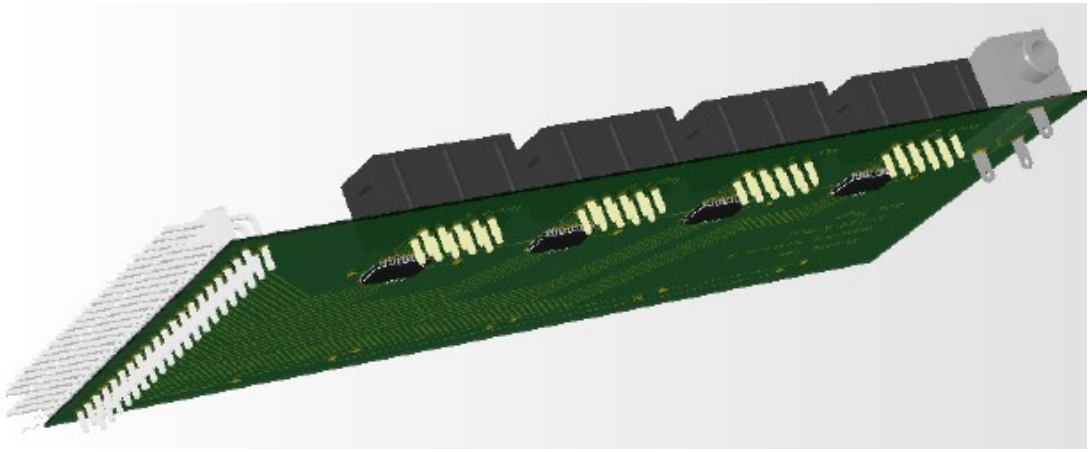


Figuur 6: Logic Analyzer - schermafbeelding

This development board has 3 external IO connectors with each 40 pins. Some of these pins are shared with onboard components. We programmed the FPGA with some signal generation and used a logic analyzer to check whether these components interfere. Next we choose 32 connections to connect with the LED cube.

2.2.3 FPGA & LED kubus interface

Because the LED cube works with 5 V and the FPGA's IO's work with 3.3V we need to make a small PCB with some voltage translators.



Figuur 7: Koppelings-PCB

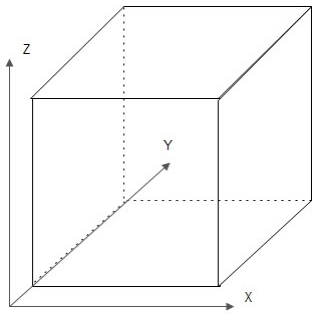
2.3 Programmatie LED-kubus

2.3.1 VHDL code

First we made exercises to drive the LED cube, separate from the "eDiViDe" platform. After which these exercises will be integrated in "eDiViDe".

2.3.1.1 Conversietabel

Driving the LED cube happens with 32 connections, divided into 4 ports of each 8bit wide. Because the original controlling scheme isn't very logical and educational. We made some VHDL code whereby the user can send coordinates in XYZ format and the related color, and this gets converted to the original control system.



Figuur 8: Led-kubus assenstelsel

```
type colorContainer is array ( integer range 0 to
3) of std_logic_vector ( 7 downto 0);

type xContainer is array ( integer range 0 to 7)
of colorContainer;
type yContainer is  array ( integer range 7 downto
0)
of xContainer;
```

```
constant DataTable : yContainer :=
(  (  ( x"B0",x"B1",x"B2",x"FF"),( x"B3",x"B4",x"B5",x"FF"),
      ( x"B6",x"B7",x"B8",x"FF"),( x"B9",x"BA",x"BB",x"FF"),
      ( x"B4",x"B5",x"B6",x"FF"),( x"B7",x"B8",x"B9",x"FF"),
      ( x"BA",x"BB",x"BC",x"FF"),( x"BD",x"BE",x"BF",x"FF")),
```

```
Type Portcontainer is array ( integer range 0 to 3) of
std_logic_vector ( 1 downto 0 );

xBuf <= DataTable(to_integer(unsigned(Y)));
colorBuf <= xBuf(to_integer(unsigned(X)));
dataBuf <= colorBuf(to_integer(unsigned(color)));
```

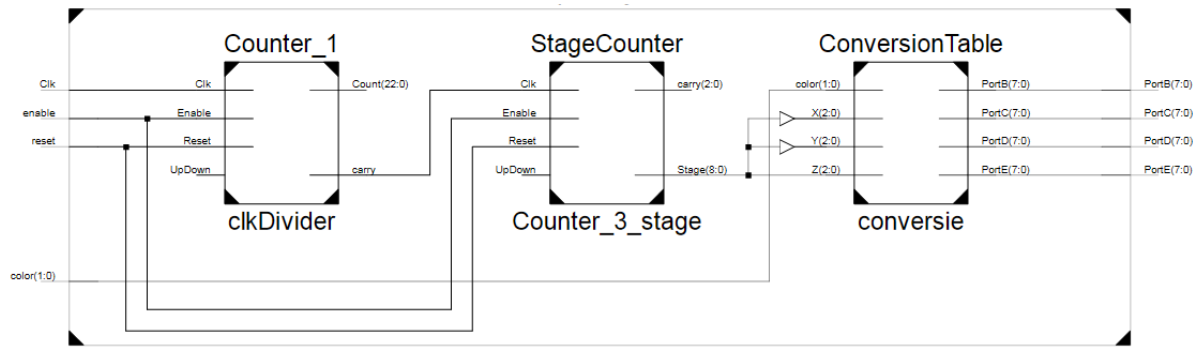
```
PortxBuffer <= PortTable(to_integer(unsigned(Y)));
PortcolorBuf <= PortxBuffer(to_integer(unsigned(X)));
PortdataBuf <= PortcolorBuf(to_integer(unsigned(color)));
```

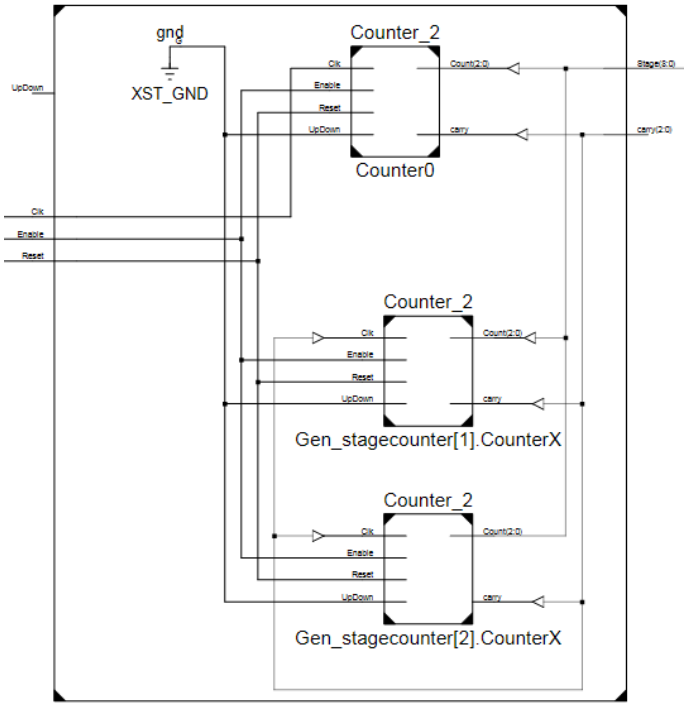
```
CASE Z IS
   WHEN  "000"  =>
         PortE <= x"01";
   WHEN  "001"  =>
         PortE <= x"02";
  WHEN OTHERS =>
         PortE <= x"00";
END CASE;
```

```
CASE PortdataBuf IS
    WHEN  "00"  =>
            PortB <= dataBuf;
            PortC <= x"FF";
            PortD <= x"FF";
    WHEN OTHERS =>
            PortB <= x"FF";
            PortC <= x"FF";
            PortD <= x"FF";
END CASE;
```

All the following exercises will use this conversion to drive the LED cube.

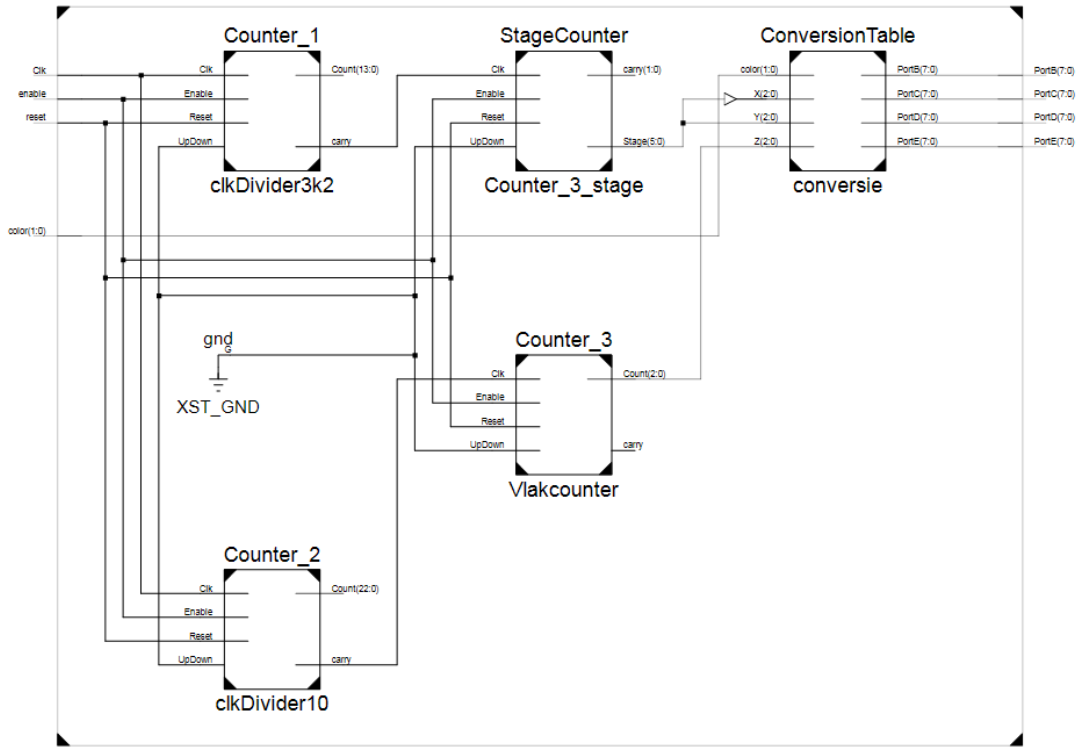
2.3.1.2 Looplicht





2.3.1.3 Lopend vlak

Het volgende wat gemaakt is op de-led kubus is een bewegend vlak, waarbij een vlak van 8x8 leds in één enkele richting beweegt met een frequentie van 10 Hz. De kleur kan gekozen worden met behulp van twee schakelaars.Het geheel heeft een "reset" en "enable" ingang die worden gebruikt voor het resetten en het aan-of uit schakelen van het geheel.



Figuur 9: Lopend vlak – Logisch schema

Allereerst dient een gans horizontaal vlak op te lichten. Dit kan worden gerealiseerd door alle leds in een horizontaal vlak snel op te laten lichten, waarbij het vlak een minimum vernieuwingsfrequentie moet hebben van 50Hz om de illusie op te wekken dat het ganse vlak oplicht. Één vlak bevat 64 leds, dus dient de frequentie waarmee elke led in moet vlak op te lichten, ingesteld te worden op 3,2 KHz.

$$F_{uit} = 8\ x\ 8\ x\ 50 = 3{,}2\ KHz$$

Deze frequentie kan men bekomen door de klok te delen met behulp van een tellern waarbij de teller limiet ingesteld dient te worden op 15625.

$$Maximum = \frac{50\,000\,000}{3200} = 15625$$

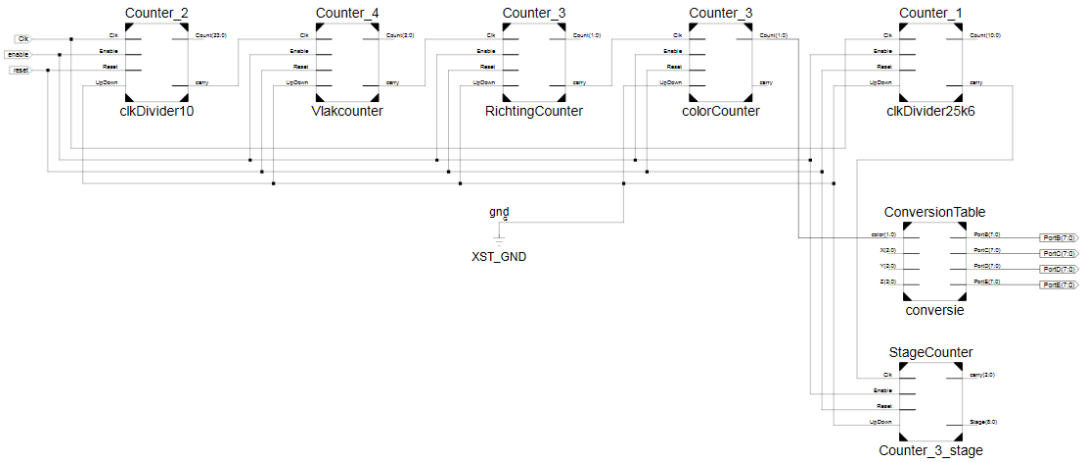
De "carry" van deze teller dient vervolgens als "enable"-ingang voor een "stagecounter", deze bevat inwendig twee tellers waarvan de "carry" van de eerste teller verboden is met de "enable"-ingang van de tweede teller. Met de eerste teller wordt de x-coördinaat geïncrementeerd en met de tweede de y-coördinaat.

Vervolgens dient een frequentie van 10 Hz gemaakt te worden. Dit gebeurt opnieuw met een teller waarvan de "carry"-uitgang gebruikt wordt als gedeelde

frequentie. Met dit 10 Hz-kloksignaal wordt er vervolgens een teller geïncrementeerd die de z-as voorstelt. Tot slot worden alle coördinaten door de conversietabel omgevormd tot de correcte data aan de uitgang.

2.3.1.4 Lopend vlak met automatische kleurverandering

In deze oefening wordt de VHDL-structuur beschreven om in de led-kubus een bewegend vlak te creëren, dat zich afwisselend over een andere as verplaatst. Nadat de verplaatsing over de drie assen is gebeurd, verandert het vlak van kleur.



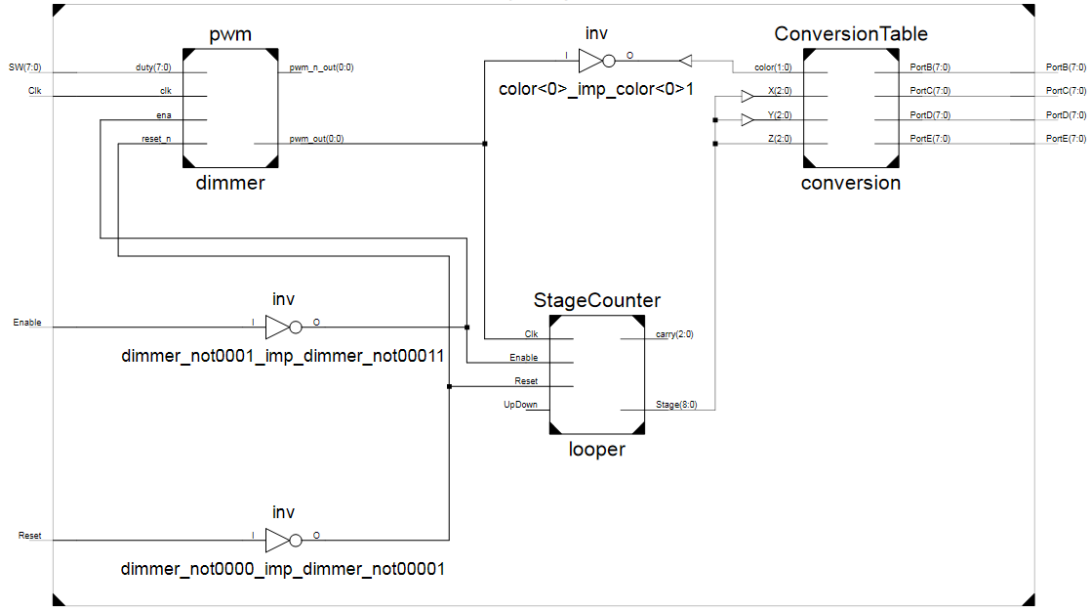
Figuur 10: Lopend vlak met kleurverandering – Logisch schema

Eerst wordt er met een counter een klokdeler gerealiseerd zoals in de vorige oefening. Ditmaal met een uitgangsfrequentie van 25,6KHz. Op elke stijgende flank van dit signaal wordt een "stagecounter" geïncrementeerd die inwendig bestaat uit 3 tellers, voor elke as één. Vervolgens wordt op dezelfde wijze een kloksignaal van 10Hz gecreëerd. Dit signaal wordt gebruikt als ingang van een teller, die eveneens één as incrementeerd. In een later stadium in deze structuur zal dan één van de assen van de "stagecounter" vervangen worden door de as afkomstig van deze teller. Het carry uitgang van deze enkele teller wordt gebruikt als klokingang op een volgende teller. Deze "carry"-uitgang zal telkens een puls creëren wanneer het vlak de rand van de kubus heeft bereikt. Deze volgende teller zal vervolgens incrementeren en op basis van de tellerstand zal er in een proces met behulp van een "case"-structuur geselecteerd worden welke van de assen uit de "stagecounter" verder wordt doorgegeven naar de conversietabel, om de richting van de beweging van het vlak te veranderen. De "carry" van deze teller wordt tot slot nog verbonden met een volgende teller, die de kleur van de leds voortstelt. Telkens wanneer het vlak de beweging in drie richtingen heeft voltooid zal deze teller incrementeren en wordt de volgende kleur doorgegeven naar de conversietabel.

2.3.1.5 PWM gestuurde led kubus

Hier dient de led-kubus volledig opgelicht te worden, waarbij de intensiteit van de kubus geregeld kan worden met behulp van acht schakelaars. Dit zorgt voor een resolutie van 8 bits, waarbij een waarde van 0 tot 255 de intensiteit voortstelt. Aan de ingang bevinden zich 2 drukknoppen die gebruikt moeten worden als "reset" en "enable"-ingang van het geheel.

Voor het regelen van de intensiteit van de kubus is puls breedte modulatie een zeer goede oplossing, waarvan hieronder ook dus gebruikt wordt gemaakt.

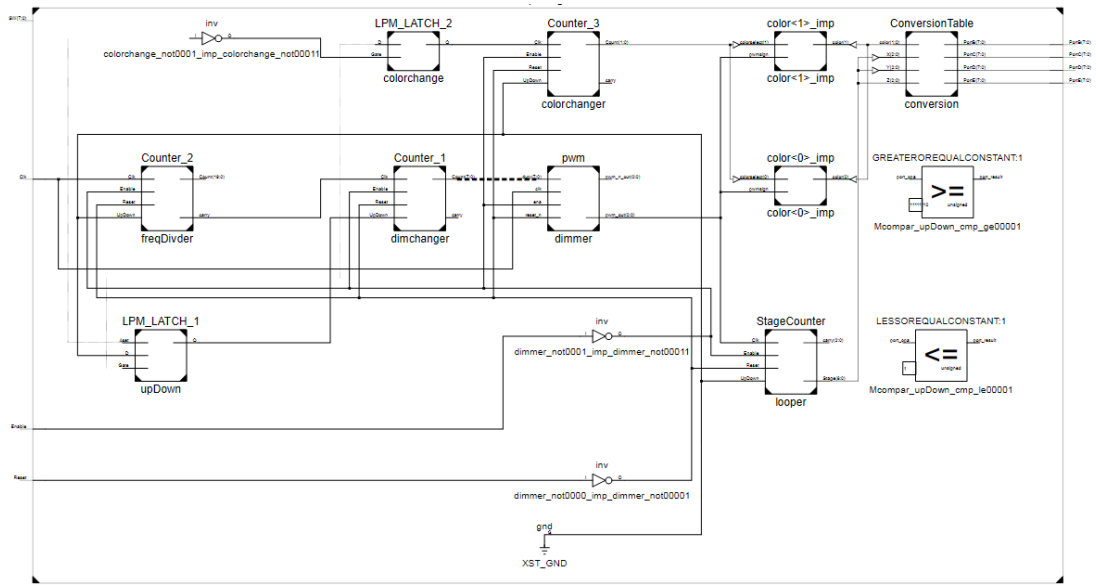


Figuur 11: PWM gestuurde kubus – Logisch schema

De eerste module in deze oefening is de "PWM"-module. De frequentie en resolutie van deze module kan ingesteld worden met de generieke parameters. De frequentie dient hier 25,6 KHz te zijn en de resolutie 8 bits. Ook dient de frequentie van het kloksignaal aan de ingang gedefinieerd te worden. Deze is hier 50 MHz. Verder wordt de systeemklok aan de klokingang van deze module verbonden en worden er ook 8 schakelaars verbonden met de "duty-cycle"-ingang. "Reset" en "enable" zijn in dit geval beide drukknoppen, die actief hoog zijn. Vermits de "reset" van alle onderdelen actief laag is, moet deze geïnverteerd te worden. Ook de enable moet geïnverteerd worden, waardoor deze actief is, wanneer de knop niet is ingedrukt. De PWM-uitgang van de "PWM"-module wordt aangesloten op de klok van de "stagecounter", hierdoor incrementeerd deze en zal telkens, wanneer het PWM-signaal opnieuw begint, de volgende led geactiveerd worden. Het PWM-signaal wordt gebruikt in een "case"-structuur. Hierin wordt beschreven dat, wanneer het signaal hoog is de leds blauw dienen op te lichten door de waarde "00" door te geven, en waneer dit signaal laag is de waarde "11" als kleur wordt meegegeven, wat er voor zorgt dat de led niet oplicht.Tot slot worden weer alle coördinaten omgezet met behulp van de conversietabel.

2.3.1.6 Lichtmodulatie

Deze oefening is een uitbreiding op de voorstaande, maar in plaats van het lichtniveau handmatig in te stellen, dient er een lichtmodulatie plaats te vinden. Hierbij hoort de kubus stelselmatig feller te gaan branden. Nadat het maximum is bereikt, moet deze terug stelselmatig doven. Vervolgens dient dit patroon herhaalt te worden, maar in de volgende kleur, namelijk groen, daarna ook in het rood. Hierna begint het patroon helemaal opnieuw in het blauw en blijft dit zich herhalen.

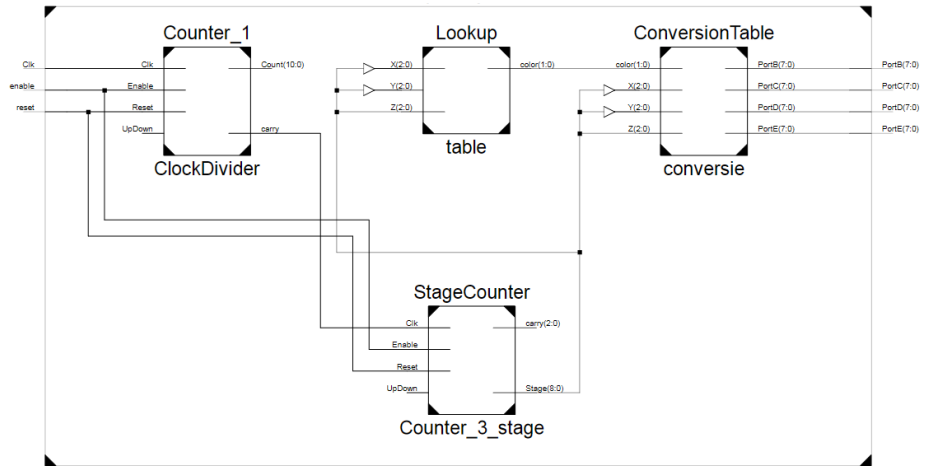


Figuur 12: Lichtgemoduleerde kubus – logisch schema

Zoals in de voorgaande oefening wordt eerst de PWM-module generiek ingesteld op 25,6 KHz. Deze PWM-uitgang wordt ook weer doorgevoerd naar de "stagecounter", die zorgt voor het één voor één aflopen van alle coördinaten. Vervolgens wordt er met een counter een frequentiedeler gerealiseerd, die een uitgangsfrequentie van 100 Hz bevat. Deze frequentie wordt aangesloten op een volgende teller, die van 0 tot 255 telt. De uitgang van deze teller dient als "duty-cycle"-ingang van de PWM-module. Hierdoor gaat de duty-cycle van het PWM-signaal stelselmatig wijzigen op een frequentie van 100Hz. In een apart proces wordt gecontroleerd wanneer deze teller op 255 komt, van hieruit wordt een signaal met een hoog niveau naar deze teller gestuurd, waardoor deze terug zal aftellen tot 0. Wanneer deze nul bereik zal dit in ditzelfde proces gedetecteerd worden en zal dit signaal weer laag gemaakt worden. Ook wordt er in dit zelfde proces een puls gegenereerd dat naar een andere teller gaat, die zorgt voor de regeling van de kleur. Deze teller zal op de stijgende flank van deze puls geïncrementeerd worden, en de kleur van de kubus veranderen.

2.3.1.7 Statisch patroon

In deze opgave is het de bedoeling om een statisch vooraf gedefinieerd patroon in de led-kubus te verkrijgen. Dit statisch patroon is een tweedimensionaal patroon van 8x8, dat van de voorkant zichtbaar is en zich herhaalt in de diepte.



Figuur 13: statisch patroon – Logisch schema

Om dit te verwezenlijken kan gebruik gemaakt worden van een klokdeler die een frequentie van 25,6KHz. Ook een "stagecounter"-module, die zorgt voor het aflopen van de coördinaten. Elke coördinaat gaat vervolgens naar een tabel, waarin wordt bekeken welke kleur de bepaalde led op die coördinaat dient te hebben. In dit geval wordt er in de tabel enkel rekening gehouden met de X- en Z-coördinaten omdat deze een tweedimensionale tabel is. Voor de Y-as vindt dan telkens een herhaling van het zelfde patroon plaats. Deze tabel wordt verwezenlijkt met behulp van een tweedimensionale array, waarin de kleur van elke led van het tweedimensionale vlak staat. Vervolgens wordt deze kleur doorgegeven naar de conversietabel en wordt de data doorgegeven naar de led-kubus. De tweedimensionale array wordt gemaakt door eerst een array te definieren met een "integer range" van 0 tot 7 van "std_logic_vector" met een breedte van 2 bits en vervolgens een tweede array met een "integer range" van 7 tot 0 van de vorige array zoals hier onder weergegeven. De tweede array krijgt een "integer range" van 7 tot 0 ,zodanig dat de array kan gemaakt worden, zoals er naar de kubus wordt gekeken. Indien er van "0 tot 7" zou staan zou de array verticaal gespiegeld zijn, wat het moeilijk maakt om een patroon te definieren.

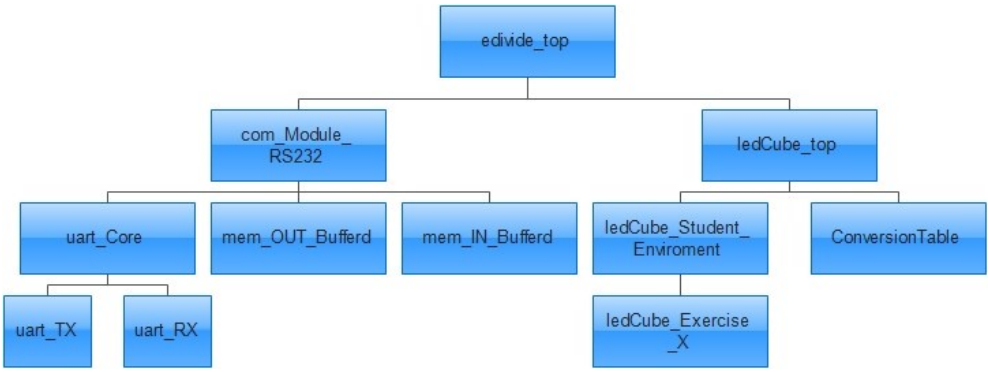
```
type xContainer is array ( integer range 0 to 7) of std_logic_vector
( 1 downto 0);
type zContainer is  array ( integer range 7 downto 0) of xContainer;
```

De eigenlijke array wordt dan vervolgens aangemaakt door een constante te definieren van het type van de tweede array, in dit geval “zContainer”. Hierin wordt vervolgens de data voor de kleur van de leds te geschreven. In dit onderstaande voorbeeld is er gekozen voor een blauw kruis, hierbij is de data voor de kleur “00” en de overige led’s dienen gedooft te zijn, zodus staat hier de data “11”.

```
constant STable : zContainer
    :=(("00","11","11","11","11","11","11","00"),
       ("11","00","11","11","11","11","00","11"),
       ("11","11","00","11","11","00","11","11"),
       ("11","11","11","00","00","11","11","11"),
       ("11","11","11","00","00","11","11","11"),
       ("11","11","00","11","11","00","11","11"),
       ("11","00","11","11","11","11","00","11"),
       ("00","11","11","11","11","11","11","00"));
```

2.3.2 Omzetting naar Edvide strucuur

Voordat de VDHL-bestanden op de server kunnen geplaatst worden, moeten deze omgevormd worden naar een structuur waar de software van "Edivide" mee kan werken. Zo moeten er van de oefeningen allereerst lege sjablonen gemaakt worden. Hiervoor moet de "conversietabel"-module hieruit verwijderd worden en moeten de poortuitgangen vervangen worden door de X-,Y-,Z- en kleuruitgangen. Vervolgens ook nog een "wrapper" waarin de oefening die de student gemaakt heeft, terecht komt. Deze noemt "student enviroment". Hierin kunnen eventuele oefeningsafhankelijke componenten worden gedeclareerd, waarvan de student gebruik kan maken. Ook worden in dit bestand alle ingangen en uitgangen die niet naar de oefening gaan, aan dummy-signalen gekopped. Zodanig ontstaan geen fouten tijdens compilatie. Dit bestand komt vervolgens terecht in de algemene module voor de setup, deze noemt "Setup Top", deze kan nog extra opstellingsafhankelijke modules bevatten. In dit geval zal deze de conversietabel bevatten. Ten slotte wordt dit bestand verwerkt in de algemene "top"-module van "Edivide" waarin ook de seriële communicatie zit verwerkt.



Figuur 14: eDiViDe VHDL structuur

2.3.2.1 LedCube Exercise

Voor elke oefening moet er een sjabloon gemaakt worden. Dit sjabloon kom op de webinterface van "Edivide", en wordt door de student gebruikt om de code in te schrijven. Het sjabloon bestaat uit de "entity" die al reeds ingevuld is en een lege "architecture", waarin de code komt.

2.3.2.2 Student enviroment

Allereerst is voor elke oefening een module gemaakt waarin de oefening terecht komt. In deze modules worden alle ingangen en uitgangen die in de oefening worden gebruikt, doorverbonden naar de oefeningmodule. De ingangen en uitgangen die niet worden gebuikt, worden verbonden met het dummy-signaal om zodanig fouten en waarschuwingen te vermijden die niets te maken hebben met de oefening.

2.3.2.3 LEDCube Top

In deze module wordt de "Student enviroment"-module verwerkt. Alle ingangen en uitgangen die mogelijk voorkomen in een van de oefeningen, worden hierin doorverbonden naar de "Student enviroment"-module. Ook zit in deze module de conversietabel verwerkt die de coördinaten en kleur waarden omzet naar de juiste data op de juiste poorten.

2.3.2.4 Edivide Top

Deze module is de hoogste in de hierachische structuur, hierin wordt de "LEDCube Top"-module verwerkt. Ook bevat deze de module die zorgt voor de seriële communicatie. Alle in- en uitgangen van de "LEDCube Top"-module worden aan fysieke in- en uitgangen gekoppeld of aan de data buffers, status en control, van de seriële communicatie. De twee buffers van de seriële communicatie bevatten elk 16 bytes. Van de "status"-buffer worden er 5 bytes gebruikt en van de "control"-buffer worden er 4 bytes gebruikt. De verbinding met de in- en uitgangen van de "LEDCube Top"- module is als volgt.

```
status(0)<=FreeOut;
status(1)<= "00000" & x;
status(2)<= "00000" & y;
status(3)<="00000" & z;
status(4)<="00000" & colorOut;
```

Elke in- en uitgang wordt toegewezen aan een apparte byte en indien de in – of uitgang minder dan 8 bits bevat worden er nullen aan toegevoegt. Dit maakt het eenvoudiger om later in de website deze data te gebruiken. Enkel de "reset"-ingang en de "enable"-ingang worden samen aan één byte verbonden.

```
FreeIn<=control(0);
LightLevel<=control(1);
colorSel<=control(2) ( 1 downto 0);
enable<= control(3)(0);
reset<= control(3)(1);
```

2.3.2.5 Bestandsstructuur

Alle bestanden moeten georganiseerd worden in een mappenstructuur, die opgegeven is door "Edivide". Deze structuur kan later gekopiëerd worden naar de lokale server. In de map "exercise_solutions" komen alle ingevulde oplossingen te staand, samen met de onderliggende modules. De map "exercise_templates" bevat alle lege sjablonen voor elke oefening. Hierna is er voor elke oefening een map voorzien genaamd "wrapper_exercise_xx", waarbij "xx" vervangen moet worden door de nummer van de oefening. In deze mappen komt voor elke oefening de "LedCube_Student_Enviroment". Vervolgens is er een map voorzien, genaamd "wrapper_general". Hierin komt de "edivide_top"-module te staan, samen met de onderliggende modules, die zorgen voor de seriële communicatie. Tot slot is er een map "wrapper _setup", waarin de "LedCube_top"-module staat. In deze map staat ook de "ConversionTable".

```
exercise_solutions
    Counter.vhd
    LedCube_Exercise_01_Solution.vhd
    LedCube_Exercise_02_Solution.vhd
    LedCube_Exercise_03_Solution.vhd
    LedCube_Exercise_04_Solution.vhd
    LedCube_Exercise_05_Solution.vhd
    LedCube_Exercise_06_Solution.vhd
    Lookup.vhd
    pwm.vhd
    StageCounter.vhd
exercise_templates
    LedCube_Exercise_01_Empty.vhd
    LedCube_Exercise_02_Empty.vhd
    LedCube_Exercise_03_Empty.vhd
    LedCube_Exercise_04_Empty.vhd
    LedCube_Exercise_05_Empty.vhd
    LedCube_Exercise_06_Empty.vhd
wrapper_exercise_01
    LedCube_Student_Environment.vhd
wrapper_exercise_02
    LedCube_Student_Environment.vhd
wrapper_exercise_03
    LedCube_Student_Environment.vhd
wrapper_exercise_04
    LedCube_Student_Environment.vhd
wrapper_exercise_05
    LedCube_Student_Environment.vhd
wrapper_exercise_06
    LedCube_Student_Environment.vhd
wrapper_general
    comm_Module_RS232.vhd
    edivide_TOP.ucf
    edivide_TOP.vhd
    mem_IN_Buffered.vhd
    mem_OUT_Buffered.vhd
    package_TypeDeclarations.vhd
    uart_Core.vhd
    uart_RX.vhd
    uart_TX.vhd
wrapper_setup
    ConversionTable.vhd
    LedCube_TOP.vhd
```

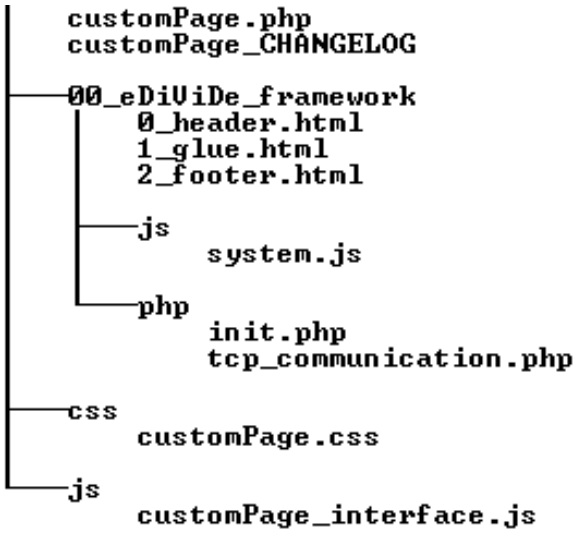
Figuur 15: VHDL bestandsstructuur

2.4 Ontwikkeling Webinterface

Het bedienen en controleren van de led-kubus gebeurt via een interface op een webpagina. De webpagina moet de video van de webcam kunnen weergeven, knoppen en schakelaars bevatten voor het bedienen van de led-kubus en eventueel ook een weergave van de status. De pagina is gebaseerd op een sjabloon dat onwikkeld is door "Edivide".

2.4.1 Bestanden en structuur

De webinterface die de gebruiker te zien zal krijgen is de "customPage.php", hierin worden alle interface elementen in gemaakt. In de map "js" staat het bestand "customPage_interface.js", hierin worden alle functies geschreven die uitgevoerd moeten worden wanneer de gebruiker één van de interface elementen bediend.



Figuur 16: Webinterface bestandsstructuur

Deze functies roepen vervolgens functies aan in het "system.js" bestand dat samen met de PHP code in "tcp_communication.php" zorgt voor het verzenden en ontvangen van data naar de led-kubus. De HTML bestanden in de map "00_eDiviDe_framework" zijn de hoofding en voet van de webinterface. Dan is er nog een bestand "init.php" deze bevat de code voor de initalisatie van de opstelling. Tot slot de map "css" bevat alle bestanden die gebruikt worden voor de layout van de pagina.

Alle bestanden in de map "00_eDiviDe_framework" moeten ongewijzigd blijve, omdat deze het protocol voor de communicatie bevatten en later eventueel geüpdate zullen worden door de makers van "Edivide.

2.4.2 Werking en ontwikkeling van de webinterface

Wanneer de gebruikter toegang krijgt tot de opstelling, komt deze terecht op de pagina "customPage.php". Als de gebruiker één van de bedieningselementen bedient, zoals een schakelaar, wordt er een javascript-functie uitgevoerd die staan geschreven in "customPage_interface.js".

```
<input type="checkbox" class="Vswitch" name="EnableSW" id="EnableSW"
onClick="EnableUpdate()"/>
```

In deze functie wordt de status van het bedieningselement gecontrolleerd, in het geval van een schakelaar zal de controle bestaan uit het controleren of de schakelaar aan of uit is.

```
function EnableUpdate() {
            if($('#EnableSW').prop('checked'))
                  {setupControl.enable -= 1; }
            else { setupControl.enable += 1; }
            compose_data();
}
```

De variabele "setupControl.enable" stelt één byte voor en de laagste bit hiervan moet de stand van de schakelaar bevatten. Als de schakelaar aan is wordt bij deze byte 1 opgeteld. Wanneer deze terug uit wordt gezet, wordt er 1 van de byte afgetrokken. Als de hoogste bit de status van de schakelaar zou moeten bevatten zou i.p.v. één, er 128 bij opgeteld of afgetrokken moeten worden.

In het geval van een drukknop wordt er bij het indrukken een functie aangeroepen om één bit in een data byte hoog te maken en bij het loslaten van de drukknop een functie om de data in deze byte terug op nul te zetten.

```
<button id="ResetBtn" class="basic_button" onmouseDown="ResetOn()"
onmouseUp="ResetOff()">RESET</button>
```

```
function ResetOff() {
    setupControl.reset =0 ;
    compose_data();
}
function ResetOn() {
    setupControl.reset =1 ;
    compose_data();
}
```

Nadat de data in de variabele is geplaatst wordt de functie "compose_data" aangeroepen. Deze functie gaat de data variabelen van alle bedieningselementen achter elkaar in een array plaatsen, waarna de functie "postData" wordt uitgevoerd.

```
function compose data() {
  data.byte6 = setupControl.freeIn;
  data.byte1 = setupControl.freeIn2;
  data.byte2 = setupControl.lightLevel;
  data.byte3 = setupControl.color;
  data.byte4 = setupControl.Enable;
  data.byte5 = setupControl.reset;

  postData(SETUPIDGIVENBYJOCHEN,data);
}
```

De functie “postData” bevindt zicht in het bestand “system.js” en ziet er als volgt uit.

```
function postData(setup,data) {

    $.post('00_eDiViDe_framework/php/tcp_communication.php',
      {command:"send",setupID:setup,data:data});
}
```

Deze functie gaat de PHP code in het bestand “tcp_communication.php” uitvoeren. Hierbij wordt het commando “send” meegegeven om te melden aan de PHP code dat de data verzonden dient te worden. In deze code wordt elke byte van de variable “data” in een variabele “tempbyte” gestoken, waarna er wordt gecontroleerd wat de waarde is van deze byte. Als de waarde van de byte 2,3 of 16 is wordt er voor deze byte nog een extra byte ingevoerd met de waarde 16.

```
if($tempbyte0 == 2 || $tempbyte0 == 3 || $tempbyte0 == 16)
      {$byte0 = chr(16) . chr($tempbyte0);}
else  {$byte0 = chr($tempbyte0);}
```

Dit omdat de seriële communicatie gebruik maakt van softwarematige flow controle, waarbij de start van een frame wordt aangeduid met de waarde 2 en het einde van een frame met de waarde 3. Als er data met dezelfde waarden voorkomt wordt hierbij dan een extra byte voorzien.Zo verwart de ontvanger geen data bytes met start- of stop bytes.

```
  $fileHandle = fopen("/home/edivide-admin/temp/input-
setup".$setupID.".txt", 'w') or die("Error opening file");
    fwrite($fileHandle,$data."\n");
    fclose($fileHandle);
```

Tot slot wordt de data weggeschreven naar een tekst bestand, waarna het door de “ComPortServer” verstuurd wordt naar de FPGA.

Op het moment dat de pagina geladen wordt, wordt de functie “initsetup()“ aangesproken. Deze functie zorgt voor het laden van de layout van de bedieningselementen en voor het instellen van de eerste waarden die naar de opstelling verzonden moeten worden. Ook wordt er direct de status opgevraagt van de led-kubus door de functie “getStatus(SETUPIDGIVENBYJOCHEN)”. Deze functie wordt vanaf dan elke 500ms uitgevoerd. De functie “getStatus(SETUPIDGIVENBYJOCHEN)” staat bescheven in het bestand “system.js”

```
function getStatus(setup) {

$.post('00_eDiViDe_framework/php/tcp_communication.php',{command:"po
ll",setupID:setup}, read_data, 'json');
}
```

Hierbij wordt naar het bestand “tcp_communication.php” het commando “poll” verstuurd. Hierbij gaat de PHP code een bestand inlezen dat gemaakt is door de “ComPortServer”, deze bevat de data die verzonden is door de FPGA.
Hierna geeft de PHP code deze data door aan de javascript, en wordt de functie “read_data()” uitgevoerd.

```
function read_data(response) {
  setupStatus.freeOut = response['byte0'];
  setupStatus.freeOut2 = response['byte1'];
  setupStatus.x = response['byte2'];
  setupStatus.y = response['byte3'];
  setupStatus.z = response['byte4'];
  setupStatus.colorOut = response['byte5'];

window.setTimeout(refresh,500)
}
```

Deze functie gaat elke byte inlezen in de bijhorende “setupStatus”-variabele. Hierna wordt een timeout van 500ms ingesteld, als deze tijd voorbij is wordt de functie “refresh()” aangesproken. Deze gaat de weergave-elementen op de webinterface updaten, zodat de status van de led-kubus wordt weergegeven. En hierna wordt opnieuw de status van de led-kubus opgevraagt en begint het process opnieuw.

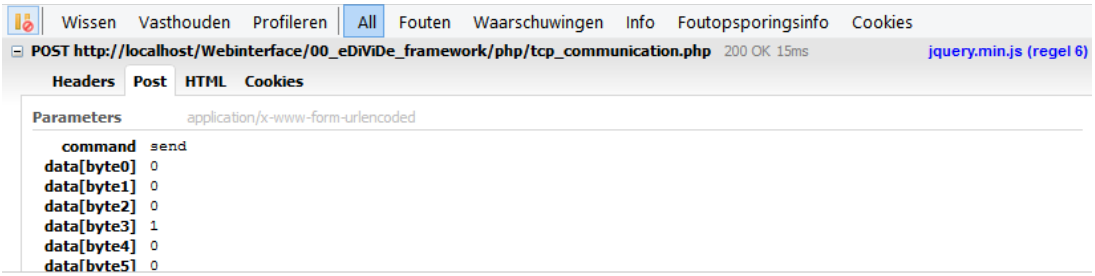
```
function refresh() {
    // update the LEDS
    var value = parseInt(setupStatus.freeOut , 16);
    var value2 = parseInt(setupStatus.freeOut2 , 16);
    if(value & 1)   { $("#LED0").addClass("LED_ON"); }  else {
$("#LED0").removeClass("LED_ON"); }
    if(value & 2)   { $("#LED1").addClass("LED_ON"); }  else {
$("#LED1").removeClass("LED ON"); }
    if(value & 4)   { $("#LED2").addClass("LED_ON"); }  else {
$("#LED2").removeClass("LED ON"); }
    if(value & 8)   { $("#LED3").addClass("LED_ON"); }  else {
$("#LED3").removeClass("LED ON"); }
    if(value & 16)  { $("#LED4").addClass("LED_ON"); }  else {
$("#LED4").removeClass("LED ON"); }
    if(value & 32)  { $("#LED5").addClass("LED_ON"); }  else {
$("#LED5").removeClass("LED_ON"); }
    if(value & 64)  { $("#LED6").addClass("LED_ON"); }  else {
$("#LED6").removeClass("LED ON"); }
    if(value & 128) { $("#LED7").addClass("LED_ON"); }  else {
$("#LED7").removeClass("LED ON"); }
    if(value2 & 1)  { $("#LED8").addClass("LED_ON"); }  else {
$("#LED8").removeClass("LED_ON"); }

getStatus(SETUPIDGIVENBYJOCHEN);
}
```

2.4.3 Testen van de webinterface

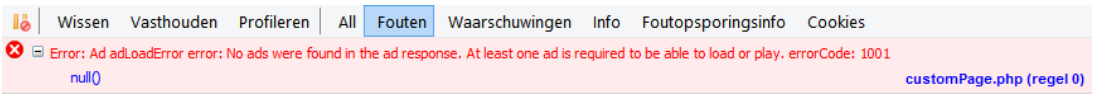
De webinterface wordt ontwikkeld en getest zonder dat deze al geïntegreerd is in het "Edivide"-systeem. Hiervoor is er op een windows-PC een lokale webserver opgezet met behulp van het programma "WampServer". Dit programma is gratis te verkrijgen en heeft alle functies die nodig zijn om de webinterface weer te geven. Zo kan deze webserver ook PHP code uitvoeren.

Om de fouten die ontstaan in de javascript code te kunnen bekijken wordt er gebruik gemaakt van de webbrowser "Mozilla Firefox" met een plugin, genaamd "Firebug". Hiermee is het ook mogelijk om de data te bekijken die wordt verstuurd van en naar de PHP code.



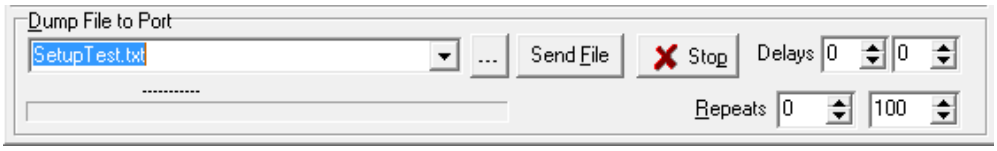
Figuur 17: Firebug – Schermafbeelding van verzonden data

Zodanig kan er worden gecontroleerd of de de data correct is, en kan er in geval van een fout gemakkelijke worden gecontroleerd waar de fout ontstaat. Hieronder een voorbeeld van de weergave van een fout en waar de fout zich bevindt.



Figuur 18: Firebug – Schermafbeelding van een Javascript fout

Tot slot wordt er ook gebruik gemaakt van het programma "Realterm", dit is een programma voor communicatie via een seriële poort. Hiermee kan een bestand geselecteerd worden dat herhaaldelijk seriëel verstuurd moet worden. Hierbij wordt ingesteld dat dit bestand elke 100ms verzonden moet worden.



Figuur 19: Realterm - schemafbeelding

Voor het controleren van de status van de led-kubus is dit iets ingewikkelder. Realterm beschikt niet over de mogelijkheid om data tussen een bepaalde start en stop sequentie weg te schrijven. Zodus wordt er handmatig gecontroleerd wat de ontvangen data is, en deze data wordt dan manueel in een bestand gescheven om zo te controleren of de webinterface dit correct weergeeft.

2.5 Opzetten van lokale server

Nadat de VHDL-code is geschreven voor de oefeningen, dient de lokale server geïnstalleerd te worden. Dit is een Linux-server met software voor het compileren van de VHDL-code en het programmeren van de FPGA. Ook moet deze server serieel kunnen communiceren met de FPGA. Op de server is ook een webcam aangesloten waarmee de FPGA-opstelling kan bekeken worden. Verder dient er nog een reeks software en scripts geïnstalleerd worden voor de communicatie met de hoofdserver van "Edivide".

2.5.1 Hardware

De lokale server wordt gerealiseerd met behulp van een DELL TowerEdge 110II, deze is te koop vi ode website van DELL en kost € 746. De specificaties van de server zijn de volgende:

Basis:	PowerEdge T110 II Tower Chassis, Up to 4x 3.5" Cabled HDDs
Processor:	Intel® Xeon® Processor E3-1220v2, 4C/4T,3.10GHz, 8M Cache, 69W, Turbo
Geheugen:	4GB Memory (2x2GB) 1600Mhz Single RankedLow Volt UDIMM(speed is CPU dependent)
RAID-connectiviteit:	C3 - RAID1 w ith On-board SATA Controller, Requires Exactly 2 SATA HDDs
Opslagmedia:	2x 500GB, SATA, 3.5-in, 7.2K RPM Hard Drive(Cabled)
Netwerkkaarten:	Intel® PRO/1000PT GbE Single Port ServerAdapter, Cu, PCIe-1
Optische apparatuur:	16X DVD-ROM Drive with SATA Cable

Verder is er ook nog een webcam nodig. Hiervoor wordt gebruik gemaakt van een Logitech-webcam van het type "C310". Deze werkt zeer zeker met de scripts van Edivide. Deze webcam kan gekocht worden via de website van "RS Components". De webcam kost € 46.30.

Tenslotte wordt er gebruik gemaakt van een USB-naar-serieel omvormer van het merk "FTDI". Er kan mogelijks ook geopteerd worden om kabels van het mer "Prolific" of "Exar". Maar FTDI-kabels hebben een ingebouwd serienummer, wat ten goede komt wanneer er eventueel een tweede opstelling aan de server zou gekoppeld worden. Zo kan de server d.m.v. het serienummer de juiste-kabel aanspreken. Deze USB-naar serieel omvormer kan ook gekocht worden via de website "van RS Components".

2.5.2 Installeren van Linux

Het eerste wat er dient te gebeuren is het installeren van Linux. De Linux-distributie die hiervoor wordt gebruikt is "Debian 6.0.5". Er bestaan reeds nieuwere versies, maar omdat deze distributie op alle andere servers draait en werkt is hier voor dezelfde versie gekozen om zo eventuele problemen met comptabiliteit te vermijden. Verder zijn er dan nog verschillende versies van deze distributie beschikbaar. Er zijn versies voor 32-bit en voor 64-bit processoren alsook zijn er versies beschikbaar voor ARM-procesoren. Vermits hier gebruikt gemaakt wordt van een 64-bit PC kan er gekozen worden voor zowel de 32-bit als de-64 bit versie. Maar om optimaal gebruik te maken van de capaciteiten van de PC is er gekozen voor 64-bit. Dit wordt aangeduid in de distributies als "amd64". De aanduiding "amd64" wilt niet zeggen dat deze versie enkel bedoeld is voor AMD-procesoren. Deze kan zowel voor Intel- als AMD-procesoren gebruikt worden. Dat de aanduiding "amd" bevat komt doordat AMD de eerste was die een 64-bit processor hadden ontwikkeld en hiervoor de naam "amd64" hadden gebruikt. Deze benaming is vervolgens verder blijven bestaan maar wordt nu gebruikt voor een algemene aanduiding op 64-bit procesoren, ongeacht het merk.

Verder kunnen de versies nog opgedeeld worden met verschil in bureaublad-omgeving, zo bestaat er "GNOME", "KDE", "LXDE" en "XFCE". Deze verschillen zijn in deze toepassing niet echt belangerijk, vermits de pc als server zal gebruikt worden. Daarom is er hier gekozen voor de "GNOME" bureaublad omgeving.

De distributie kan vervolgens ook op verschillende wijzen geïnstalleerd worden. Allereerst met behulp van installatie DVD's die de volledige Debian distributie bevatten. Ook is het mogelijk om linux te installeren met behulp van een "USB live image", maar deze bevat niet de volledige Debian-distributie. Zodus wordt er in dit geval gekozen om gebruik te maken van de installatie-DVD's. De volledige installatie van de distributie bestaat uit 7 DVD's, maar enkel de éérste is nodig voor het installeren van de distributie. De overige DVD's bevatten enkel softwarepaketten. De installatie-DVD is verkrijgbaar op de website van Debian in de vorm van een ISO-bestand. Dit bestand kan vervolgens op een DVD gebrand worden, waarna deze kaar is om gebruikt te worden voor de installatie.

Vervolgens dient de opstartvolgorde van de PC veranderd te worden, zodanig dat het eerste medium waarop gezocht wordt voor een opstartsyseem, de DVD is. Dit kan vast ingesteld worden in de bios van de pc of het medium waarvan opgestart moet worden. Er kan ook eenmalig worden gekozen voor opstarten vanaf de DVD, door F12 in te drukken bij het opstarten en dan te kiezen voor "DVD-ROM".

Wanneer de DVD opgestart is kan er gekozen worden voor "GUI install". Hiermee wordt de grafische installatie omgeving opgestart, die als een wizard doorlopen kan worden. Allereerst kan de taal worden gekozen, dit is Engels. Vervolgens kan het land worden ingesteld door allereerst te kiezen voor "others", daarna voor "Europe" en tot slot "Belgium". In de volgende stap dient gekozen te worden voor "UK" en "GB.UTF-8" dit zijn de locals waarmee het systeem gaat werken. Hierna dient het toetsenbord gekozen te worden dewelke in dit geval een azertytoetsenbord is, dus wordt er gekozen voor "Belgian". In de volgende stap gaat het systeem een verbinding met internet proberen te leggen. Dit kan door te kiezen voor "automatic configure DHCP". De hostnaam voor de pc die vervolgens moet worden ingevuld is "ed-lsHK02". Vervolgens dient "edivide.eu" als hostnaam ingevuld te worden.

Hierna kan er een eerste gebruiker worden aangemaakt. Voor de gebruikersnaam wordt gekozen voor "edivide" met een bijbehorend paswoord.

In de volgende stap moeten de harde schijven geconfigureerd worden voor RAID. Eerst wordt er gekozen voor "Guided Partitioning" en één partitie voor de gehele schijf, de swap partitie wordt automatisch gecreëerd. Dit moet dan ook voor de tweede schijf gebeuren. Hierna moet de root-paritie '/' van de elke schijf geselecteerd worden en dient de partitie "EXT3" veranderd te worden naar "Physical volume for RAID". Vervolgens "Configure software RAID" waar dan de optie "Create MD device", "RAID 1", "2", en dan "0" gekozen moet worden. Dit dient toegepast te worden op de hoofdpartitie van beide schijven, waarna het bevestigd moet worden en de wijziginen zullen vervolgens uitgevoerd worden.

Nadat de harde schijven zijn ingesteld kan de package manager worden geselecteerd, hiervoor wordt er gekozen voor de belgische FTP-link, be.debian.org. Hierna wordt er gevraagd voor software, en wordt er gekozen voor GDE, SSH en Standard system utilities. Tot slot moet in de volgende stap GRUB nog geïnstalleerd worden. Wanneer dit voltooid is word de computer opnieuw opgestard en kan de DVD verwijderd worden en de BIOS instellingen terug naar opstarten van harde schijf worden gezet.

2.5.3 Systeem configuratie

Nadat de installatie voltooid is en linux voor de eerste maal opstart, dient er eerst een extra gebruiker aangemaakt te worden. Dit kan door in de terminal volgende stappen in te voeren

su

Hierna wordt gevraagd voor het root paswoord. Nadat dit is ingevuld en correct is, kan de nieuwe gebruiker toegevoegd worden.

adduser edivide−khlim

Als paswoord dient hier "KHLim$1" ingevuld te worden. Vervolgens dient deze gebruiker ook administratorrechten te verkrijgen. Hiervoor dient het "sudoers"-bestand aangepast te worden met het volgende commando:

su −c visudo

Hierin dienen de volgende regels toegevoegd te worden:

edivide ALL=(ALL) ALL
edivide-khlim ALL=(ALL) ALL

Vervolgens moeten de gebruikers in elkaars groep toegevoegd worden zodanig dat ze beide schijfrechten bezitten voor de "eDiViDe"-map.

"sudo usermod −a −G edivide−khlim edivide"
"sudo usermod −a −G edivide edivide−khlim"

Hierna dient de onderstaande UMASK-instelling toegevoegd te worden in het sshd en het login bestand.

"sudo vi /etc/pam.d/sshd"

"sudo vi /etc/pam.d/login"

Optional UMASK setting - created by edivide-admin
session optional pam umask.so umask=0002.

Vervolgens dient nog het loggen tijdens het opstarten geactiveerd te worden, dit kan door volgende commando's:

cd /etc/rc3.d
"ls | grep gdm | xargs sudo rm"
"ls | grep cups | xargs sudo rm"
"ls | grep bluetooth | xargs sudo rm"
"ls | grep saned | xargs sudo rm"

Vervolgens dient ook not de opstartmodus gewijzigd te worden door het bestand inittab te wijzigen:

"sudo vi /etc/inittab"

Hierin dient volgende lijn aangepast te worden zodanig dat er een 3 staat i.p.v. een 5.

"id:**3**:initdefault"

Hierna kan de server herstart worden zodanig dat men in deze opstartmodus terecht komt met behulp van het onderstaande commando:

"sudo shutdown −r now"

Vanaf nu start de server in een terminal zonder grafische interface.

Hierna kan er nog wat overbodige software verwijderd worden, deze is overbodig voor de server en heeft heel wat updates en schijfruimte nodig.

"sudo apt−get remove −−purge openoffice.org"
"sudo apt−get remove −−purge openoffice.org−base"
"sudo apt−get remove −−purge openoffice.org−core"
"sudo apt−get autoremove"

Tot slot kunnen er nog systeem updates worden geïnstalleerd via het apt-get comamndo.

"sudo apt−get update"
"sudo apt−get upgrade"

2.5.4 Installatie van software en scripts

Nadat bovestaande configuratie is voltooid, kunnen de nodige software en scripts geïnstalleerd worden.

2.5.4.1 Xilinx Tools

Het eerste programma dat wordt geïnstalleerd, is Xilinx 14.2. Deze software kan gedownload worden op de webpagina van Xilinx. Vervolgens wordt het op een USB-opslagapparaat geplaatst en overgezet naar de server, in de map "/home/edivide-khlim/software/xilinx". Wanneer de USB aangesloten wordt, moet deze gemount worden.

Men kan het apperaat vinden met onderstaande regel, waarna men een lijst krijgt van alle opslagapparaten.

"Sudo fdisk –l"

Vervolgens dient hieruit het correcte apparaat gekozen te worden, in dit geval is dat "sdc1". Hierna moet dit gemount worden, als locatie hiervoor is gekozen voor /media/USBstick". En tenslotte kan dan het bestand gekopiëerd worden.

"Sudo mount /dev/sdc1 /media/USBstick"

"Cp /media/USBstick/Xilinx_ISE_DS_Lin_14.2_p.28xd.3.0tar /home/edivide-khlim/software/Xilinx/"

Hierna moet het bestand uitgepakt worden op dezelfde locatie.

"cd /home/edivide−khlim/software/Xilinx"
"tar −xf Xilinx_ISE_DS_Lin_14.2_P.28xd.3.0.tar"

Om de software te kunnen installeren moet er gebruik gemaakt worden van een grafische omgeving, omdat het installatiepakket hier gebruik van maakt. De grafische omgeving kan worden opgestart met het commando "startx". Hierna ken het installatiepakket worden gestart met "sudo ./xsetup"

Tijdens de installatie moet eerst twee maal de liscentievoorwaarden geaccepteerd worden. Hierna kiest men voor de optie "ISE Design Suite Embedded Edition+ Vivaldo Design Edition". In de volgende stap moet de optie "install cable drivers" uitgezet worden. En vervolgens wordt "/opt/Xilinx" als installatiepad gekozen.

Hierna dient het liscentiebestand in de map "/home/edivide/.Xilinx/" geplaatst te worden. De "edivide"-administrator zal dit bestand kopiëren naar de map "/home/edivide-khlim/.Xilinx".

Vervolgens moet er voor worden gezorgt dat de instellingen van Xilinx automatisch geladen worden tijdens het opstarten. Hiervoor geven we allereerst uitvoerrechten aan alle gebruikers voor het bestand dat zorgt voor deze instellingen.

sudo chmod a+x /opt/Xilinx/14.2/ISE DS/settings64.sh

Vervolgens maken we een script aan "xilinx64.sh" in de map "/etc/profile.d/" waarin een verwijzing wordt geplaatst naar script "settings64.sh", zo worden bij het opstarten de instellingen geladen. De verwijzingen naar dit script is de volgende :

source /opt/Xilinx/14.2/ISE DS/settings64.sh

Hierna volgt het installeren van de drivers voor de kabels, dit moet manueel gebeuren. De drivers die bij Xilinx ISE worden bijgeleverd geven problemen in Debian. Allereerst moeten "fxload" en "libusb-dev" geinstalleerd worden.

```
sudo apt−get install −y fxload
sudo apt−get install −y libusb−dev
```

Hierna dient de usb-driver gedownload en geïnstalleerd te worden.

```
cd /home/edivide−khlim/software/xilinx
sudo git clone git://git.zerfleddert.de/usb−driver
cd usb−driver
sudo make
export LD PRELOAD=/opt/software/usb−driver/libusb−driver.so
```

2.5.4.2 Digilent Tools

Vervolgens moeten er enkele software paketten van "Digilent" worden geïnstalleerd. Deze pakketen vormen samen de software "Adept" waarmee Digilent ontwikkelborden geprogrammeerd kunnen worden. Vervolgens wordt er ook nog een plugin geïnstalleerd waarmee het mogelijk is "Impact" te gebruiken om het bord te programmeren.

Allereerst wordt er een map voorzien om deze in te plaatsten. Deze map genaamd "digilent" wordt aangemaakt in de locatie "/home/edivide−khlim/software/". Vervolgens worden de softwarepakketten gedownload naar deze map. Door eerst naar deze map te gaan en vervolgens het onderstaande in te voeren.

```
wget http://www.digilentinc.com/Data/Products/ADEPT2/\
digilent.adept.runtime 2.10.2−x86_64.tar.gz

wget http://www.digilentinc.com/Data/Products/ADEPT2/\
digilent.adept.utilities 2.1.1−x86_64.tar.gz

wget http://www.digilentinc.com/Data/Products/DIGILENT−PLUGIN/
libCseDigilent 2.4.3−x86_64.tar.gz
```

Hierna worden alle paketten uitgepakt door gebruik te maken van het "tar" commando met de optie "-xzf".

```
tar −xzf digilent.adept.runtime 2.10.2−x86_64.tar.gz

tar −xzf digilent.adept.utilities 2.1.1−x86_64.tar.gz

tar -xfz libCseDigilent 2.4.3−x86_64.tar.gz
```

Als deze zijn uitgepakt kunnen de eerste twee pakketten worden geïnstalleerd door in de map te gaan met de uitgepakte bestanden en het installatie script uit te voeren.

sudo ./install.sh

Van het derde pakket dienen enkele bestanden gekopiëerd te worden naar de installatie-locatie van Xilinx. Op de locatie "/opt/Xilinx/14.2/ISE_DS/ISE/lib/lin64/plugins/" moet eerst de map "Digilent/LibCseDigilent/" gemaakt worden.

Hierna kunnen de bestanden uit de locatie "/home/edivide−khlim/software/digilent/libCseDigilent_2.4.3−x86_64/ISE14x/plugin/libCseDigilent" naar deze map gekopiëerd worden.

2.5.4.3 Configuratie

Vervolgens moet er een bestand worden aangemaakt waarin een regel wordt gedefinieerd zodanig dat "Impact" kan werken met de USB-driver van "libusb". Met onderstaand commando wordt dit bestand aangemaakt en geopend in de "VI" tekstverwerker.

sudo vi /etc/udev/rules.d/51−libusb−driver.rules

Hierin moet volgende regel worden geschreven:

ACTION=="add", SUBSYSTEMS=="usb", ATTRS{idVendor}=="03fd", MODE="666"

Op dezelfde locatie moet nog een bestand worden aangepast, zodanig dat Adept kan worden uitgevoerd zonder administrator-rechten.

sudo vi /etc/udev/rules.d/52−digilent−usb.rules

Hierin moeten onderstaande tekst vervangen worden.

ATTR{idVendor}=="1443", MODE="666"
ACTION=="add", ATTR{idVendor}=="0403", ATTR{manufacturer}=="Digilent", MODE="666", RUN+="/usr/local/sbin/dftdrvdtch %s{busnum} %s{devnum}"

De tekst die in de plaats moet komen is de volgende :

SUBSYSTEM=="usb", ATTRS{idVendor}=="1443", MODE:="0666"
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="0403", ATTRS{manufacturer}=="Digilent",
MODE:="0666", RUN+="/usr/local/sbin/dftdrvdtch %s{busnum} %s{devnum}"

Hierna dient er naar deze locatie nog een bestand met bepaalde regels gekopiëerd te worden. In dit bestand dienen enkele woorden aangepast te worden. Zo moet "TEMPNODE" vervangen worden door "tempnode", "SYSFS" door "ATTRS" en "BUS" door "SUBSYSTEMS". Ook moeten de ".hex" bestanden gekopieerd worden naar een andere map.

sudo cp /opt/Xilinx/14.2/ISE DS/ISE/bin/lin64/xusbdfwu.rules /etc/udev/rules.d/

sudo sed −i −e 's/TEMPNODE/tempnode/' −e 's/SYSFS/ATTRS/g' −e 's/BUS/ SUBSYSTEMS/' /etc/udev/rules.d/xusbdfwu.rules

sudo cp /opt/Xilinx/14.2/ISE DS/ISE/bin/lin64/xusb*.hex /usr/share/

Vervolgens dient de configuratie te gebeuren van de USB-naar-serieel omvormer. In deze opstelling wordt gebruik gemaakt van een FTDI- naar serieel omvormer. Dit is de beste keuze omdat deze een geïntegreerd serienummer bevat. Met dit serienummer is het mogelijk verschillende USB-naar-serieel kabels te onderscheiden en dienen deze niet op een vaste usb poort aangesloten te worden. Indien hier later een tweede opstelling wordt aangekoppeld, moet er een tweede usb-naar-serieel kabel van het merk FTDI worden gebruikt.

Voor de configuratie van deze kabel, moet eerst bekeken worden wat de usb-bus nummer en het apparaat nummer is. Dit kan met het commando "lsusb", waarna een lijst verschijnt met alle aangesloten usb apparaten, dat er als volgt uitziet:

Bus 002 Device 006: ID 0403:6001 Future Technology Devices International, Ltd FT232 USB-Serial (UART) IC

Met deze info kan vervolgens meer info over de kabel worden opgevraagt door het onderstaande in de terminal uit te voeren met de correcte usb-bus nummer en het apparaat nummer .

sudo udevadm info −a −p $(sudo udevadm info -q path -n /dev/bus/usb/002/006)

In deze lijst met informative is de eigenschap "ATTRSerial" terug te vinden. De waarde die hieraan is toegekend is het serienummer van de kabel, in dit geval "FTGVESTD".

Hiervoor moet opnieuw een regel worden opgesteld in de map"/etc/udev/rules.d/".

sudo vi /etc/udev/rules.d/01−persistent−USBSERIAL.rules

In dit bestand dien het volgende geschreven te worden:

#FTDI USB SERIAL #1
SUBSYSTEMS=="usb", ATTRS{manufacturer}=="FTDI", ATTRS{serial}=="FTGVESTD", NAME="USBSERIAL1", MODE="0666"

Vervolgens moeten al de regels geladen en geactiveerd worden.

sudo udevadm control −−reload−rules

sudo udevadm trigger −−action=change

Hierna wordt de bestandsstructuur aangemaakt, waar de overige configuratiebestanden in kunnen worden geplaatst. Deze bestandsstructuur dient in de "home" map van de gebruiker "edivide-khlim" te worden geplaatst. De locatie van deze map bevindt zich op"/home/edivide-khlim/". Deze structuur ziet er als volgt uit.

Figuur 20: eDiViDe – bestandsstructuur op de lokale server

Allereerst wordt het configuratiebestand voor de FPGA gemaakt. Dit bestand bevat de opstellingnummer, de naam van de opstelling, het type van ontwikkelbord en het serienummer van de programmeerkabel.

Het serienummer van de programmeerkabel kan worden achterhaald met de "Adept tools". Wanneer het onderstaande commando in de terminal wordt uitgevoerd, bekomt men de naam en het serienummer van de kabel.

dadutil enum

Dit geeft vervolgens onderstaande uitvoer:

Device: Nexys2-Setup01
Product Name: Onboard USB
User Name: Nexys2-Setup01
Serial Number: 10054D273705

Vervolgens wordt er een bestand "FPGA" in de map "config" aangemaakt waarin het volgende wordt geschreven:

01:LedCube: 50003C008141

Ook is het mogelijk met het commando "dadutil" om de naam van de kabel te veranderen. Deze naam wordt veranderd naar "Spartan3-Setup01".

dadutil setusr −d DCabUsb −−usr Spartan3-Setup01

Hierna wordt het configuratiebestand voor de webcam aangemaakt. Dit bestand heeft de naam "webcam" en wordt ook in de map "config" geplaatst. Voordat dit bestand gemaakt kan worden moet eerst het serienummer en de "vendor-id" van de webcam achterhaald worden. Wanneer de webcam aangesloten is, staat in de map "/dev/v4l/by-id/" een naam met daarin "video-index0". Deze naam dient gebruikt te worden in het configuratie bestand van de webcam. In dit geval is de naam "usb-046d 081b 001D84D0-video-index0".

In het configuratiebestand van de webcam wordt dan volgende informatie geschreven:

ln -s /dev/v4l/by-id/usb-046d_081b_09D79A50-video-index0 /dev/webcam1

Vervolgens moeten de rechten van dit bestand gewijzigd worden, zodat iedereen lees- en uitvoerrechten over het bestand bezit:

chmod 755 /home/edivide-khlim/eDiViDe/config/webcam

Vervolgens moet er voor worden gezorgt dat het apparaat gestart worden wanneer de server opgestart wordt. Dit kan door een link naar het bestand te plaatsen in de map "init.d".

sudo ln −s /home/edivide−khlim/eDiViDe/config/webcam /etc/init.d/webcam
sudo update−rc.d webcam start 20 3 . stop 20 0 1 6 .

Vervolgens kan het apparaat manueel gestart worden, zo moet de server niet opnieuw opgestart worden.

cd /etc/init.d
./webcam console

Dan moet er worden voorzien dat het start/stop commando van "daemons" beschikbaar is voor de accounts op de server.

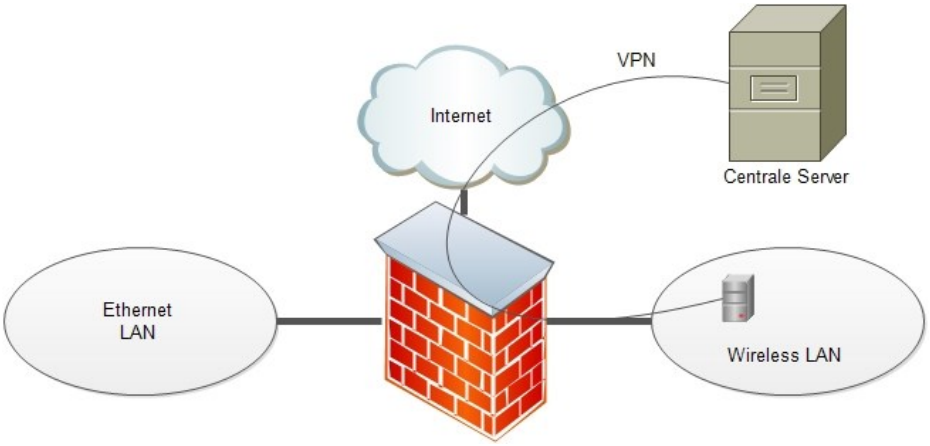
cd /bin
ln −s /sbin/start−stop−daemon start−stop−daemon

Hierna dient de mappenstructuur met de VHDL code's gekopiëerd te worden naar de server. Hiervoor wordt er een map, "setup01", gemaakt in de map "setups". Hierin dient de gehele structuur geplaatst te worden.

Tenslotte moeten in de map "tools" nog enkele programma's worden geplaatst. Deze programma's zijn, "ComPortServer" en het programma dat zorgt voor de video, "justinTV". Deze programma's worden gegeven door de systeembeheerder van "edivide".

2.5.4.4 VPN-verbinding instellen

Als de lokale server volledig geconfigureerd is moet deze verbinden worden met de centrale server d.m.v. een VPN. (Virtual Private Network).
Omwille van veiligheidsredenen mag de server niet aangesloten zijn op het lokale schoolnetwerk. Zo zou er een koppeling ontstaan tussen het netwerk van Thomas More en KHLim. Hiervoor is een aansluiting voorzien waarbij de server in het wireless netwerk komt. Dit netwerk staat open en is volledig gescheiden van het schoolnetwerk.



Figuur 21: VPN netwerk

Voor het opstellen van de VPN-verbinding wordt gebruik gemaakt van de software "OpenVPN". De software installeren gebeurt met het commando "apt-get".

sudo apt−get install −y openvpn

Hierna dient er een configuratiebestand gemaakt te worden, met de nodige gegeven voor de software.

sudo vi /etc/openvpn tun61.conf

Vervolgens de gegevens die in dit bestand moeten worden geschreven:

remote 193.190.58.17 21161
2
dev tun61
ifconfig 192.168.61.2 192.168.61.1
user nobody
group nogroup
resolv-retry infinite
persist-tun
persist-key
ping 10
ping-restart 30
secret /etc/openvpn/static.key

Hierna moet er aan de systeembeheerder een code worden gevraagd. Deze code wordt geplaatst in een bestand "static.key" en dit bestand moet op de locatie "/etc/openvpn/static.key" worden geplaatst. In de map "/etc/default/" staat een bestand "openvpn" met daarin een regel "#AUTOSTART='all'". Het "#" moet uit deze regel verwijdert, zodanig dat de vpn verbinding automatisch wordt opgezet wanneer de server herstart.

Vervolgens wordt er software geïnstalleerd die zorgt voor de synchronisatie tussen de servers. Deze software noemt "ntpdate"

Hiervoor dient ook een configuratiebestand gemaakt te worden, genaamd "ntpdate" in de map "/etc/cron.daily/". Hierin dient onderstaande tekst te komen.

```
#!/bin/sh
NTPDATE=/usr/sbin/ntpdate
NTPSERVER=ntp1.oma.be
$NTPDATE $NTPSERVER
```

Daarna dienen de rechten over dit bestand gewijzigd te worden.

```
sudo chmod a+x /etc/cron.daily/ntpdate
```

Vervolgens is het aan de systeembeheerder van "eDiViDe" om de VPN op de centrale server in te stellen.

2.6 Verdere planning

Het boekwerk over de stage en het eindwerk dient enkele tijd voor het einde van de stage ingeleverd te worden. Dit heeft als gevolg dat niet de volledige stage in dit boek besproken kan worden. Hieronder worden taken beschreven die nog gepland zijn voor de resterende tijd.

2.6.1 Uploaden van de webinterface

Wanneer de VPN-verbinding actief is en nadat de systeembeheerder de scripts op de lokale server heeft geplaatst, kan de webinterface geupload worden naar de centrale server. Hierop draait een webserver, met webruimte toegekent aan elke instelling die een opstelling maakt. Deze webruimte is voorzien om de gehele opstelling te testen, zonder dat de webinterface publiek beschikbaar wordt gesteld.

2.6.2 Testen van het geheel

Nadat de webinterface actief is op de centrale server, kan het geheel getest worden. Hierbij moet de code nog manueel in de FPGa geprogrammeerd worden, maar kan de bediening met de webinterface getest worden.

2.6.3 Online beschikbaar stellen van de opstelling

Vanaf dat de gehele opstelling correct werkt, kan deze publiekelijk beschikbaar gesteld worden. Hierbij zal de opstelling toegangkelijk worden via de website van "eDiViDe". Hiervoor is er een algemene engelse uitleg geschreven over de opstelling en uitleg over elke oefening. Deze tekst zal door de systeem-beheerder online geplaatst worden waarna de opstelling vrij toegankelijk is.

BESLUIT

In dit eindwerk wordt het mogelijk om een led-kubus te besturen met een FPGA-ontwikkelbord. Waarna er hiervoor enkele oefeningen met de oplossingen zijn opgesteld. Het FPGA-ontwikkelbord is vervolgens verbonden aan een server, die geïntegreerd is in "eDiViDe". Hiermee is het mogelijk om een led-kubus vanop afstand te programmeren, te bedienen en te controleren via een webinterface.

Later zou, met behulp van de uitleg in deze scriptie, het aantal opstellingen uitgebreid kunnen worden. Aan deze server kan een tweede opstelling gekoppeld kunnen worden en er kunnen op dezelfde methode meerdere servers worden aangemaakt.

Het zou evenwel mogelijk zijn om met deze basis, en verder onderzoek, een systeem te ontwikkelen waarmee het mogelijk is om microcontroller gebaseerde systemen vanop afstand te programmeren.

LITERATUURLIJST

Elen, D., Mannaerts, C., Van Braeckel, G., Vuegen, L. (2010). 8x8x8 RGB- LED kubus. Geel: Campina Media vzw.

Kenneth L. Short. (2008). VHDL for engineers. Pearson

Vandorpe Jochen. eDiViDe: Debian Installation Guidelines

Vandorpe Jochen. eDiViDe: Debian Configuration Guidelines

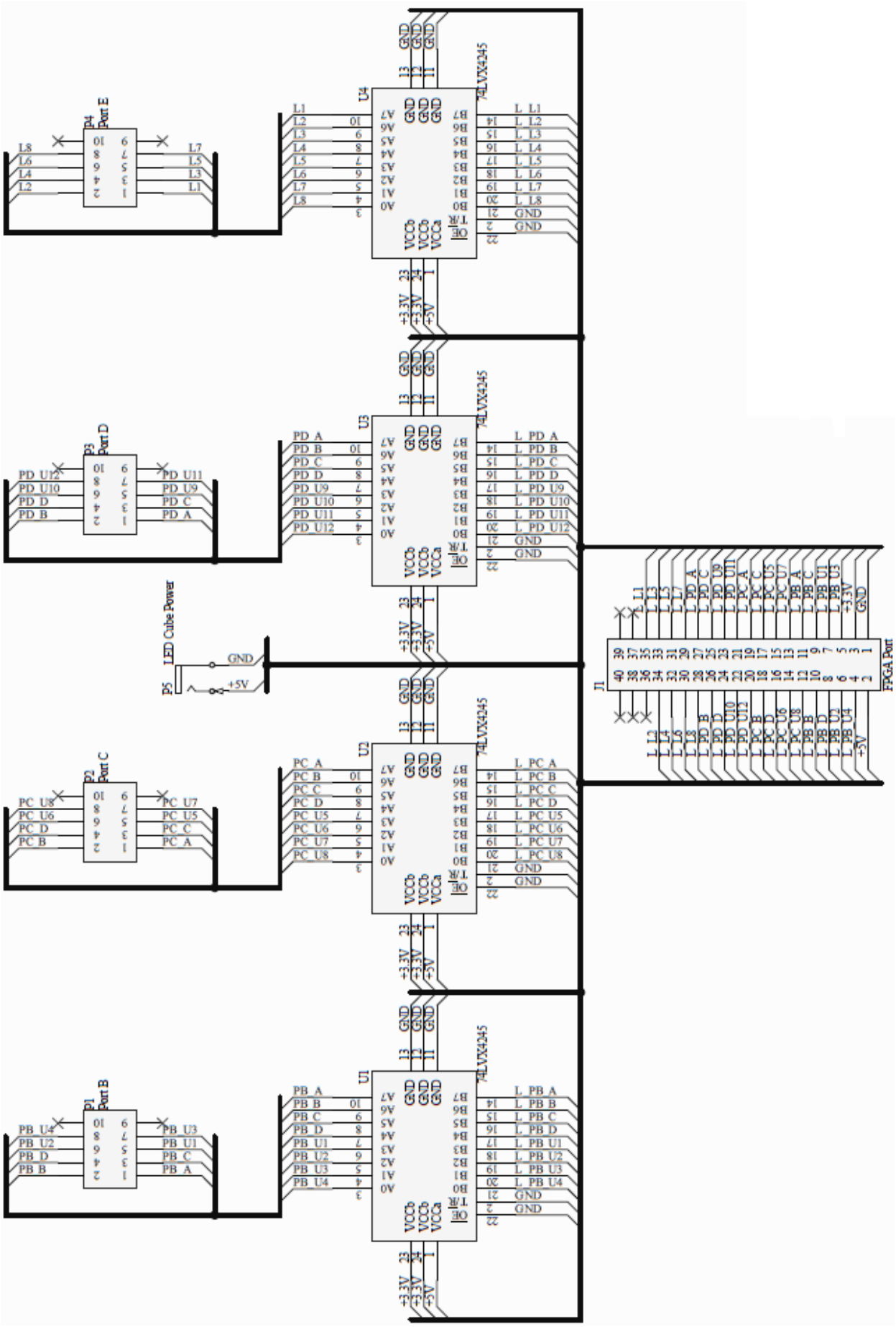
Vandorpe Jochen. eDiViDe: Local setup integration, Part I - System

Vandorpe Jochen. eDiViDe: Local setup integration, Part II - Web interface

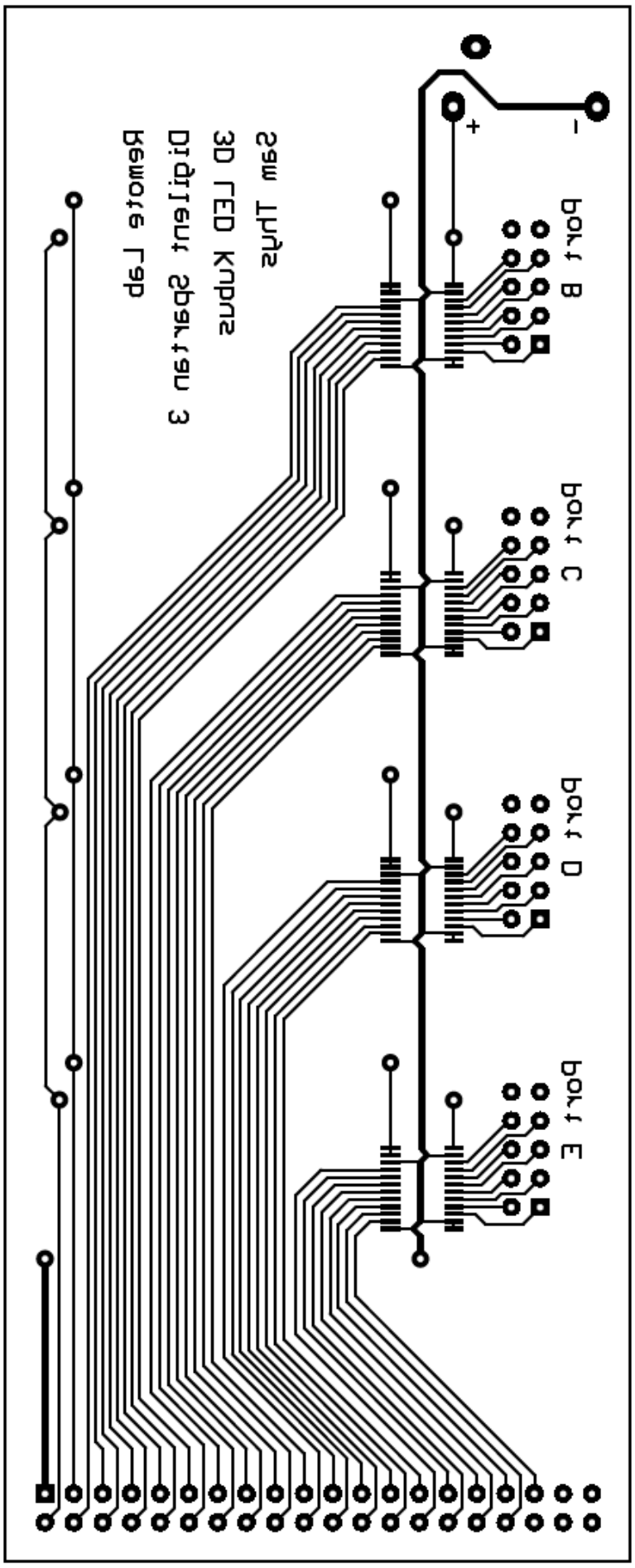
Vandorpe Jochen. eDiViDe: Debian Xilinx tools installation

BIJLAGEN

Bijlage 1 : Interface bord schema



Bijlage 2: Interface board layout



Bijlage 3: Conversion table module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ConversionTable is
    Port ( --CLK : in  STD_LOGIC;
              X : in  STD_LOGIC_VECTOR (2 downto 0);
              Y : in  STD_LOGIC_VECTOR (2 downto 0);
              Z : in std_logic_vector ( 2 downto 0);
              color : in STD_LOGIC_VECTOR ( 1 downto 0);
              PortB : out std_logic_vector ( 7 downto 0);
              PortC : out std_logic_vector ( 7 downto 0);
              PortD : out std_logic_vector ( 7 downto 0);
              PortE : out std_logic_vector ( 7 downto 0));
end ConversionTable;

architecture Behavioral of ConversionTable is
type colorContainer is array ( integer range 0 to 3)
      of std_logic_vector ( 7 downto 0);
type xContainer is array ( integer range 0 to 7) of colorContainer;
type yContainer is  array ( integer range 7 downto 0) of xContainer;
Type Portcontainer is array ( integer range 0 to 3)
      of std_logic_vector ( 1 downto 0 );
type xPortContainer is array ( integer range 0 to 7) of Portcontainer;
type yPortContainer is  array ( integer range 7 downto 0) of
xPortcontainer;

signal  xBuf : xContainer;
signal colorBuf : colorContainer;
signal dataBuf : std_logic_vector ( 7 downto 0 );

constant DataTable : yContainer :=
(((x"B0",x"B1",x"B2",x"FF"),(x"B3",x"B4",x"B5",x"FF"),
  (x"B6",x"B7",x"B8",x"FF"),(x"B9",x"BA",x"BB",x"FF"),
  (x"B4",x"B5",x"B6",x"FF"),(x"B7",x"B8",x"B9",x"FF"),
  (x"BA",x"BB",x"BC",x"FF"),(x"BD",x"BE",x"BF",x"FF")),
 ((x"BC",x"BD",x"BE",x"FF"),(x"BF",x"70",x"71",x"FF"),
  (x"72",x"73",x"74",x"FF"),(x"75",x"76",x"77",x"FF"),
  (x"D8",x"D9",x"DA",x"FF"),(x"DB",x"DC",x"DD",x"FF"),
  (x"DE",x"DF",x"B0",x"FF"),(x"B1",x"B2",x"B3",x"FF")),
 ((x"78",x"79",x"7A",x"FF"),(x"7B",x"7C",x"7D",x"FF"),
  (x"7E",x"7F",x"E0",x"FF"),(x"E1",x"E2",x"E3",x"FF"),
  (x"EC",x"ED",x"EE",x"FF"),(x"EF",x"D0",x"D1",x"FF"),
  (x"D2",x"D3",x"D4",x"FF"),(x"D5",x"D6",x"D7",x"FF")),
 ((x"E4",x"E5",x"E6",x"FF"),(x"E7",x"E8",x"E9",x"FF"),
  (x"EA",x"EB",x"EC",x"FF"),(x"ED",x"EE",x"EF",x"FF"),
  (x"E0",x"E1",x"E2",x"FF"),(x"E3",x"E4",x"E5",x"FF"),
  (x"E6",x"E7",x"E8",x"FF"),(x"E9",x"EA",x"EB",x"FF")),
 ((x"D0",x"D1",x"D2",x"FF"),(x"D3",x"D4",x"D5",x"FF"),
  (x"D6",x"D7",x"D8",x"FF"),(x"D9",x"DA",x"DB",x"FF"),
  (x"D4",x"D5",x"D6",x"FF"),(x"D7",x"D8",x"D9",x"FF"),
  (x"DA",x"DB",x"DC",x"FF"),(x"DD",x"DE",x"DF",x"FF")),
 ((x"DC",x"DD",x"DE",x"FF"),(x"DF",x"B0",x"B1",x"FF"),
  (x"B2",x"B3",x"B4",x"FF"),(x"B5",x"B6",x"B7",x"FF"),
  (x"E8",x"E9",x"EA",x"FF"),(x"EB",x"EC",x"ED",x"FF"),
  (x"EE",x"EF",x"D0",x"FF"),(x"D1",x"D2",x"D3",x"FF")),
 ((x"B8",x"B9",x"BA",x"FF"),(x"BB",x"BC",x"BD",x"FF"),
  (x"BE",x"BF",x"70",x"FF"),(x"71",x"72",x"73",x"FF"),
  (x"7C",x"7D",x"7E",x"FF"),(x"7F",x"E0",x"E1",x"FF"),
  (x"E2",x"E3",x"E4",x"FF"),(x"E5",x"E6",x"E7",x"FF")),
 ((x"74",x"75",x"76",x"FF"),(x"77",x"78",x"79",x"FF"),
  (x"7A",x"7B",x"7C",x"FF"),(x"7D",x"7E",x"7F",x"FF"),
  (x"70",x"71",x"72",x"FF"),(x"73",x"74",x"75",x"FF"),
  (x"76",x"77",x"78",x"FF"),(x"79",x"7A",x"7B",x"FF")));
```

```
signal  PortxBuffer : xPortContainer;
signal PortcolorBuf : PortContainer;
signal PortdataBuf : std_logic_vector ( 1 downto 0 );
constant PortTable : yPortContainer :=
(((("01","01","01","11"),( "01","01","01","11"),
  ("01","01","01","11"),( "01","01","01","11"),
  ("00","00","00","11"),( "00","00","00","11"),
  ("00","00","00","11"),( "00","00","00","11")),
 (("01","01","01","11"),( "01","01","01","11"),
  ("01","01","01","11"),( "01","01","01","11"),
  ("00","00","00","11"),( "00","00","00","11"),
  ("00","00","00","11"),( "00","00","00","11")),
 (("01","01","01","11"),( "01","01","01","11"),
  ("01","01","10","11"),( "10","10","10","11"),
  ("00","00","00","11"),( "00","00","00","11"),
  ("00","00","00","11"),( "00","00","00","11")),
 (("10","10","10","11"),( "10","10","10","11"),
  ("10","10","10","11"),( "10","10","10","11"),
  ("00","00","00","11"),( "00","00","00","11"),
  ("00","00","00","11"),( "00","00","00","11")),
 (("10","10","10","11"),( "10","10","10","11"),
  ("10","10","10","11"),( "10","10","10","11"),
  ("01","01","01","11"),( "01","01","01","11"),
  ("01","01","01","11"),( "01","01","01","11")),
 (("10","10","10","11"),( "10","10","10","11"),
  ("10","10","10","11"),( "10","10","10","11"),
  ("01","01","01","11"),( "01","01","01","11"),
  ("01","01","01","11"),( "01","01","01","11")),
 (("10","10","10","11"),( "10","10","10","11"),
  ("10","10","10","11"),( "10","10","10","11"),
  ("00","00","00","11"),( "00","01","01","11"),
  ("01","01","01","11"),( "01","01","01","11")),
 (("10","10","10","11"),( "10","10","10","11"),
  ("10","10","10","11"),( "10","10","10","11"),
  ("00","00","00","11"),( "00","00","00","11"),
  ("00","00","00","11"),( "00","00","00","11")));

Begin

xBuf <= DataTable(to_integer(unsigned(Y)));
colorBuf <= xBuf(to_integer(unsigned(X)));
dataBuf <= colorBuf(to_integer(unsigned(color)));
PortxBuffer <= PortTable(to_integer(unsigned(Y)));
PortcolorBuf <= PortxBuffer(to_integer(unsigned(X)));
PortdataBuf <= PortcolorBuf(to_integer(unsigned(color)));

process (PortdataBuf,databuf,z)
begin
CASE Z IS
      WHEN  "000"  =>
             PortE <= x"01";
      WHEN  "001"  =>
             PortE <= x"02";
      WHEN  "010"  =>
             PortE <= x"04";
      WHEN  "011"  =>
             PortE <= x"08";
      WHEN  "100"  =>
            PortE <= x"10";
      WHEN  "101"  =>
             PortE <= x"20";
      WHEN  "110"  =>
             PortE <= x"40";
      WHEN  "111"  =>
            PortE <= x"80";
      WHEN OTHERS =>
            PortE <= x"00";
END CASE;
```

```
CASE PortdataBuf IS
       WHEN  "00"  =>
            PortB <= dataBuf;
            PortC <= x"FF";
            PortD <= x"FF";
      WHEN  "01"  =>
            PortB <= x"FF";
            PortC <= dataBuf;
            PortD <= x"FF";
      WHEN  "10"  =>
            PortB <= x"FF";
            PortC <= x"FF";
            PortD <= dataBuf;
      WHEN OTHERS =>
            PortB <= x"FF";
            PortC <= x"FF";
            PortD <= x"FF";
END CASE;
end process;
end Behavioral;
```

Bijlage 4: VHDL Counter module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.all;

entity Counter is
    Generic (CountLimit : INTEGER := 200;
             bitlength : integer :=8;
             increment : integer :=1);

    Port     (Clk : in  STD_LOGIC;
             Reset : in  STD_LOGIC; -- High active
             Enable : in  STD_LOGIC; -- High active
             UpDown : in  STD_LOGIC; -- High -> up
             Count : out  std_logic_vector ( bitlength-1 downto 0);
             carry : out std_logic);
end Counter;

architecture Behavioral of Counter is

begin
process ( Clk,Reset,Enable,UpDown)
variable counter : unsigned (bitlength-1 downto 0):= ( others =>'0');
begin
if ( reset ='0') then
    counter := ( others => '0');
    carry <='0';
elsif ( clk'event and clk ='1') then
    if ( enable = '1') then
        case UpDown is
        When '0' =>
            if ( counter = to_unsigned(countlimit,counter'length)) then
            counter := ( others => '0');
            carry <= '1';
            else
            counter := counter +1;
            carry <= '0';
            end if;
        When others =>
            if ( counter = to_unsigned(0,counter'length)) then
            counter := to_unsigned(countlimit,counter'length);
            carry <= '1';
            else
            counter := counter -1;
            carry <= '0';
            end if;
        end case;
    end if;
end if;
count <= std_logic_vector(counter);
end process;
end Behavioral;
```

Bijlage 5: VHDL StageCounter module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity StageCounter is
generic ( Countlimit : integer := 7;
          Stages : integer := 3;
          BitLength : integer := 3);
Port( Clk : in  STD_LOGIC;
      Reset : in  STD_LOGIC;
      Enable : in  STD_LOGIC;
      UpDown : in  STD_LOGIC;
      Stage : out  STD_LOGIC_VECTOR ((stages*bitlength-1) downto 0);
      carry : out STD_logic_VECTOR ( stages-1 downto 0));
end StageCounter;

architecture Behavioral of StageCounter is
signal carry_v : std_logic_vector ( stages-1 downto 0);
component Counter is
    Generic ( CountLimit : INTEGER := 200;
              bitlength : integer :=8;
              increment : integer :=1);
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC; -- High active
           Enable : in  STD_LOGIC; -- High active
           UpDown : in  STD_LOGIC; -- High -> up
           Count : out  std_logic_vector ( bitlength-1 downto 0);
           carry : out std_logic);
end component;
begin
Counter0 : Counter
generic map ( CountLimit,Bitlength,1)
port map (Clk,reset,enable,upDown,stage(bitlength -1 downto 0),carry_v(0));

Gen_stagecounter : for I in 1 to Stages-1 generate

CounterX : Counter
generic map ( CountLimit,Bitlength,1)
port map (carry_v(I-1),reset,enable,upDown,stage((bitlength*(I+1))-1 downto
bitlength*I),carry_v(I));
end generate;
carry <= carry_v;
end Behavioral;
```

Bijlage 6: VHDL PWM module

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY pwm IS
  GENERIC(
  sys_clk: INTEGER := 50_000_000; --system clock frequency in Hz
  pwm_freq: INTEGER := 100_000; --PWM switching frequency in Hz
  bits_resolution: INTEGER := 8;--bits of resolution setting the duty cycle
      phases: INTEGER := 1); --number of output pwms and phases
  PORT(
      clk: IN  STD_LOGIC; --system clock
      reset_n: IN  STD_LOGIC; --asynchronous reset
      ena: IN  STD_LOGIC; --latches in new duty cycle
      duty: IN  STD_LOGIC_VECTOR(bits_resolution-1 DOWNTO 0); --duty cycle
      pwm_out: OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0); --pwm outputs
      pwm_n_ou: OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0)); --pwm inverse
END pwm;

ARCHITECTURE logic OF pwm IS
  CONSTANT period  : INTEGER := sys_clk/pwm_freq;
--number of clocks in one pwm period
  TYPE counters IS ARRAY (0 TO phases-1) OF INTEGER RANGE 0 TO period - 1;
--data type for array of period counters
  SIGNAL count     : counters := (OTHERS => 0);
--array of period counters
  SIGNAL half_duty : INTEGER RANGE 0 TO period/2 := 0;
--number of clocks in 1/2 duty cycle
BEGIN
  PROCESS(clk, reset_n)
  BEGIN
    IF(reset_n = '0') THEN --asynchronous reset
      count <= (OTHERS => 0); --clear counter
      pwm_out <= (OTHERS => '0'); --clear pwm outputs
      pwm_n_out <= (OTHERS => '0'); --clear pwm inverse outputs
    ELSIF(clk'EVENT AND clk = '1') THEN  --rising system clock edge
      IF(ena = '1') THEN    --latch in new duty cycle
        half_duty <= conv_integer(duty)*period/(2**bits_resolution)/2;
--determine clocks in 1/2 duty cycle
      END IF;
      FOR i IN 0 to phases-1 LOOP   --create a counter for each phase
        IF(count(0) = period - 1 - i*period/phases) THEN  --end of period
          count(i) <= 0; --reset counter
        ELSE   --end of period not reached
          count(i) <= count(i) + 1; --increment counter
        END IF;
      END LOOP;
      FOR i IN 0 to phases-1 LOOP --control outputs for each phase
        IF(count(i) = half_duty) TH    --phase's falling edge reached
          pwm_out(i) <= '0'; --deassert the pwm output
          pwm_n_out(i) <= '1'; --assert the pwm inverse output
        ELSIF(count(i) = period - half_duty) THEN --phase's rising edge
          pwm_out(i) <= '1'; --assert the pwm output
          pwm_n_out(i) <= '0'; --deassert the pwm inverse output
        END IF;
      END LOOP;
    END IF;
  END PROCESS;
END logic;
```

Bijlage 7: VHDL LedCube_Top

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;
use work.TypeDeclarations_Constants.all;

entity edivide_TOP is
port (
    -- Input
    CLK         : in    std_logic;      -- Main clock
    rst         : in    std_logic;      -- Main clock
    -- Communications Interface
    RX          : in    std_logic;      -- RS232 received serial data
    TX          : out   std_logic;      -- RS232 transmitted serial data
    -- Dummy input /output signal to handle unconnected signals
    DUMMY_OUT   : out std_logic;
    -- Outputs to LED Cube
    PortB : out STD_LOGIC_VECTOR ( 7 downto 0);
    PortC : out STD_LOGIC_VECTOR ( 7 downto 0);
    PortD : out STD_LOGIC_VECTOR ( 7 downto 0);
    PortE : out STD_LOGIC_VECTOR ( 7 downto 0)
);
end edivide_TOP;

architecture Behavioral of edivide_TOP is

signal control, status: mem_array;
signal x,y,z : std_logic_vector ( 2 downto 0);
signal colorSel,colorOut : std_logic_vector ( 1 downto 0);
signal enable,reset : std_logic;
signal FreeIn,FreeOut : std_logic_vector ( 8 downto 0);
signal LightLevel : std_logic_vector ( 7 downto 0);
signal ControlBuf : std_logic_vector ( 127 downto 0);

component comm_Module_RS232 is
  generic (
    CLK_FREQ            : natural := 50;--system clock frequency
    BAUD_RATE           : natural := 115_200;  -- desired baud rate
    DATA_BITS           : natural := 8;     -- # data bits
    STATUS_UPDATE_RATE  : natural := 10); --Update rate to send status [Hz]
  port (
  -- Input
    clk     : in  std_logic;    -- Main clock
    rst     : in  std_logic;    -- Main reset
  -- External Interface
    rx      : in  std_logic;    -- RS232 received serial data
    tx      : out std_logic;    -- RS232 transmitted serial data
  -- Internal Interface
    control : out mem_array;
    status  : in mem_array);
end component;
```

```
component LedCube_Top is
    port(     Clk : in  STD_LOGIC;
              enable : in  STD_LOGIC;
              reset : in  STD_LOGIC;
              colorSel : in STD_LOGIC_VECTOR (1 downto 0);
              LightLevel : in STD_LOGIC_VECTOR (7 downto 0);
              FreeIn : in STD_LOGIC_VECTOR ( 8 downto 0);
              colorOut : out STD_LOGIC_VECTOR ( 1 downto 0);
              x : out STD_LOGIC_VECTOR ( 2 downto 0);
              y : out STD_LOGIC_VECTOR ( 2 downto 0);
              z : out STD_LOGIC_VECTOR ( 2 downto 0);
              PortB : out STD_LOGIC_VECTOR ( 7 downto 0);
              PortC : out STD_LOGIC_VECTOR ( 7 downto 0);
              PortD : out STD_LOGIC_VECTOR ( 7 downto 0);
              PortE : out STD_LOGIC_VECTOR ( 7 downto 0);
              FreeOut : out STD_LOGIC_VECTOR ( 8 downto 0);
              DummyOut: out std_logic);
end component;

begin

    inst_comm_Module_RS232: comm_Module_RS232
        port map (clk,rst,rx,tx,control,status);


inst_LedCube_Top : LedCube_Top
       port map (clk,enable,reset,colorSel,LightLevel,FreeIn,colorOut,
                    x,y,z,PortB,PortC,PortD,PortE,FreeOut,Dummy_Out);

-- COMM Channel mapping
enable <= control(4)(0);
reset <=control(5)(0);
colorsel <= control(3)(1 downto 0);
LightLevel <=control(2);
FreeIn( 7 downto 0) <=control(0);
FreeIn(8) <=control(1)(0);

end  Behavioral;
```

Bijlage 8:VHDL Edivide_Top

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;
use work.TypeDeclarations_Constants.all;

entity edivide_TOP is
port (
    -- Input
    CLK         : in    std_logic; -- Main clock
    rst         : in    std_logic; -- Main clock
    -- Communications Interface
    RX          : in    std_logic; -- RS232 received serial data
    TX          : out   std_logic; -- RS232 transmitted serial data
    -- Dummy input /output signal
    DUMMY_OUT   : out std_logic;
    -- Outputs to LED Cube
    PortB : out STD_LOGIC_VECTOR ( 7 downto 0);
    PortC : out STD_LOGIC_VECTOR ( 7 downto 0);
    PortD : out STD_LOGIC_VECTOR ( 7 downto 0);
    PortE : out STD_LOGIC_VECTOR ( 7 downto 0));
end edivide_TOP;

architecture Behavioral of edivide_TOP is

signal control, status: mem_array;
signal x,y,z : std_logic_vector ( 2 downto 0);
signal colorSel,colorOut : std_logic_vector ( 1 downto 0);
signal enable,reset : std_logic;
signal FreeIn,FreeOut,LightLevel : std_logic_vector ( 7 downto 0);

component comm_Module_RS232 is
    generic (   CLK_FREQ  : natural := 50; -- system clock frequency in MHz
                BAUD_RATE : natural := 115200; -- desired baud rate
                DATA_BITS : natural := 8;  -- # data bits
                MAX_VALUE   : integer := 5000000 -- Max value for counter
            );
    port(
  clk         : in    std_logic;      -- Main clock
  rst         : in    std_logic;      -- Main reset
  -- External Interface
  rx          : in    std_logic;      -- RS232 received data
  tx          : out   std_logic;      -- RS232 transmitted data
-- Internal Interface
 control : out mem_array;
 status  : in mem_array);
end component;
```

```
component LedCube_Top is
    port( Clk : in  STD_LOGIC; -- 50 MHz system Clock
          enable : in  STD_LOGIC; -- Button Input
reset : in  STD_LOGIC;  -- Button Input
colorSel : in STD_LOGIC_VECTOR (1 downto 0);  -- Switch vector Input
LightLevel : in STD_LOGIC_VECTOR (7 downto 0);  -- Switch vector
FreeIn : in STD_LOGIC_VECTOR ( 7 downto 0); -- Free to use Input
colorOut : out STD_LOGIC_VECTOR ( 1 downto 0); -- Color selection: "00" =
Blue / "01" = Red / "10" = Green / "11" = OFF
x : out STD_LOGIC_VECTOR ( 2 downto 0); -- X Coordinate
y : out STD_LOGIC_VECTOR ( 2 downto 0); -- Y Coordinate
z : out STD_LOGIC_VECTOR ( 2 downto 0); -- Z Coordinate
PortB : out STD_LOGIC_VECTOR ( 7 downto 0);
PortC : out STD_LOGIC_VECTOR ( 7 downto 0);
PortD : out STD_LOGIC_VECTOR ( 7 downto 0);
PortE : out STD_LOGIC_VECTOR ( 7 downto 0);
FreeOut : out STD_LOGIC_VECTOR ( 7 downto 0); -- Free to use Output
DummyOut: out std_logic);
end component;

begin
    inst_comm_Module_RS232: comm_Module_RS232
        port map (clk,rst,rx,tx,control,status);


    inst_LedCube_Top : LedCube_Top
        port map
(clk,enable,reset,colorSel,LightLevel,FreeIn,colorOut,x,y,z,PortB,PortC,Por
tD,PortE,FreeOut,Dummy_Out);

-- COMM Channel mapping

    status(0)<=FreeOut;
    status(1)<= "00000" & x;
    status(2)<= "00000" & y;
    status(3)<="00000" & z;
    status(4)<="000000" & colorOut;

    FreeIn<=control(0);
    LightLevel<=control(1);
    colorSel<=control(2)(1 downto 0);
    enable<=control(3)(0);
    reset<=control(3)(1);
end  Behavioral;
```

Bijlage 9: VHDL looplicht

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;

entity LedCube_Exercise_01 is
    Port (
    Clk : in  STD_LOGIC; -- 50 MHz system Clock
    enable : in  STD_LOGIC; -- Switch Input
    reset : in  STD_LOGIC;  -- Button Input, High Active '1'
    colorSel : in STD_LOGIC_VECTOR (1 downto 0);--Switch vector Input
    colorOut : out STD_LOGIC_VECTOR ( 1 downto 0); -- Color selection
         of leds: "00" = Blue / "01" = Red / "10" = Green / "11" = OFF
    x : out STD_LOGIC_VECTOR ( 2 downto 0); -- X Coordinate, value 0 - 7
    y : out STD_LOGIC_VECTOR ( 2 downto 0); -- Y Coordinate, value 0 - 7
    z : out STD_LOGIC_VECTOR ( 2 downto 0); -- Z Coordinate, value 0 - 7
    FreeIn : in STD_LOGIC_VECTOR ( 8 downto 0); -- Free to use Input
    FreeOut : out STD_LOGIC_VECTOR ( 8 downto 0)); -- Free to use Output

end LedCube_Exercise_01;

architecture Structural of LedCube_Exercise_01 is

signal Freq100 : std_logic;
signal xyz : std_logic_vector ( 8 downto 0);

component StageCounter is
    generic (    Countlimit : integer := 7;
                     Stages : integer := 3;
                     BitLength : integer := 3);
    Port    (      Clk : in  STD_LOGIC;
                   Reset : in  STD_LOGIC;
                   Enable : in  STD_LOGIC;
                   UpDown : in  STD_LOGIC;
                   Stage : out  STD_LOGIC_VECTOR ((stages*bitlength-1)
downto 0);
                   carry : out STD_logic_VECTOR ( stages-1 downto 0));
end component;
component Counter is
    Generic (   CountLimit : INTEGER := 200;
                   bitlength : integer :=8;
                   increment : integer :=1);
        Port(      Clk : in  STD_LOGIC;
                   Reset : in  STD_LOGIC; -- High active
                   Enable : in  STD_LOGIC; -- High active
                   UpDown : in  STD_LOGIC; -- High -> up
                   Count : out  std_logic_vector ( bitlength-1 downto 0);
                   carry : out std_logic);
end component;

begin
Counter_3_stage: StageCounter   generic map (7,3,3)
                  port map (clk,reset,freq100,'0',xyz,open);
clkDivider : counter            generic map (5000000,23,1)
                  port map ( clk,reset,enable,'0',open,freq100);
FreeOut<=FreeIn;
x<=xyz( 2 downto 0);
y<=xyz( 5 downto 3);
z<=xyz( 8 downto 6);
colorOut<=colorSel;
end Structural;
```

Bijlage 10: VHDL lopend vlak Topmodule

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;

entity LedCube_Exercise_02 is
    Port (
    Clk : in  STD_LOGIC; -- 50 MHz system Clock
    enable : in  STD_LOGIC; -- Switch Input
    reset : in  STD_LOGIC;  -- Button Input, High Active '1'
    colorSel : in STD_LOGIC_VECTOR (1 downto 0);--Switch vector Input
    colorOut : out STD_LOGIC_VECTOR ( 1 downto 0); -- Color selection
         of leds: "00" = Blue / "01" = Red / "10" = Green / "11" = OFF
    x : out STD_LOGIC_VECTOR ( 2 downto 0); -- X Coordinate, value 0 - 7
    y : out STD_LOGIC_VECTOR ( 2 downto 0); -- Y Coordinate, value 0 - 7
    z : out STD_LOGIC_VECTOR ( 2 downto 0); -- Z Coordinate, value 0 - 7
    FreeIn : in STD_LOGIC_VECTOR ( 8 downto 0); -- Free to use Input
    FreeOut : out STD_LOGIC_VECTOR ( 8 downto 0)); -- Free to use Output

end LedCube_Exercise_02;

architecture Structural of LedCube_Exercise_02 is

signal Freq3k2 : std_logic;
signal Freq10 : std_logic;
signal vlak : std_logic_vector ( 5 downto 0);
signal richting : std_logic_vector ( 2 downto 0);

component StageCounter is
    generic (   Countlimit : integer := 7;
                Stages : integer := 3;
                BitLength : integer := 3);
    Port    (   Clk : in  STD_LOGIC;
                Reset : in  STD_LOGIC;
                Enable : in  STD_LOGIC;
                UpDown : in  STD_LOGIC;
                Stage : out  STD_LOGIC_VECTOR ((stages*bitlength-1) downto
0);
                carry : out STD_logic_VECTOR ( stages-1 downto 0));
end component;

component Counter is
    Generic (   CountLimit : INTEGER := 200;
                bitlength : integer :=8;
                increment : integer :=1);
        Port(   Clk : in  STD_LOGIC;
                Reset : in  STD_LOGIC; -- High active
                Enable : in  STD_LOGIC; -- High active
                UpDown : in  STD_LOGIC; -- High -> up
                Count : out  std_logic_vector ( bitlength-1 downto 0);
                carry : out std_logic);
end component;

begin

clkDivider3k2 : counter generic map (15625,14,1)
                            port map ( clk,reset,enable,'0',open,freq3k2);
Counter_3_stage: StageCounter   generic map (7,2,3)
                                port map
(freq3k2,reset,enable,'0',vlak,open);
```

```
clkDivider10 : counter generic map (5000000,23,1)
  port map ( clk,not reset,enable,'0',open,freq10);

Vlakcounter : counter generic map (7,3,1)
  port map ( freq10,not reset,enable,'0',richting,open);

x<= vlak(2 downto 0);
y<=vlak(5 downto 3);
z<=richting;
colorOut<=colorSel;
FreeOut<=FreeIn;

end Structural;
```

Bijlage 11: VHDL lopend vlak met 3 kleuren module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;

entity LedCube_Exercise_03 is
    Port (
  Clk : in  STD_LOGIC; -- 50 MHz system Clock
  enable : in  STD_LOGIC; -- Switch Input
  reset : in  STD_LOGIC;  -- Button Input
  colorOut : out STD_LOGIC_VECTOR ( 1 downto 0); -- Color selection of
      leds: "00" = Blue / "01" = Red / "10" = Green / "11" = OFF
  x : out STD_LOGIC_VECTOR ( 2 downto 0); -- X Coordinate, value 0 - 7
  y : out STD_LOGIC_VECTOR ( 2 downto 0); -- Y Coordinate, value 0 - 7
  z : out STD_LOGIC_VECTOR ( 2 downto 0); -- Z Coordinate, value 0 - 7
  FreeIn : in STD_LOGIC_VECTOR ( 8 downto 0); -- Free to use Input
  FreeOut : out STD_LOGIC_VECTOR ( 8 downto 0)); -- Free to use Output

end LedCube_Exercise_03;

architecture Structural of LedCube_Exercise_03 is
signal Freq25k6 : std_logic;
signal Freq10 : std_logic;
signal xyz : std_logic_vector ( 8 downto 0);
signal richtingChange :std_logic;
signal richting : std_logic_vector(2 downto 0);
signal richtingData : std_logic_vector(1 downto 0);
signal colorChange :std_logic;
signal color : std_logic_vector ( 1 downto 0);

component StageCounter is
    generic (Countlimit : integer := 7;
             Stages : integer := 3;
             BitLength : integer := 3);
    Port    (Clk : in  STD_LOGIC;
             Reset : in  STD_LOGIC;
             Enable : in  STD_LOGIC;
             UpDown : in  STD_LOGIC;
             Stage : out  STD_LOGIC_VECTOR ((stages*bitlength-1) downto 0);
             carry : out STD_logic_VECTOR ( stages-1 downto 0));
end component;
component Counter is
    Generic (CountLimit : INTEGER := 200;
             bitlength : integer :=8;
             increment : integer :=1);
        Port(Clk : in  STD_LOGIC;
             Reset : in  STD_LOGIC; -- High active
             Enable : in  STD_LOGIC; -- High active
             UpDown : in  STD_LOGIC; -- High -> up
             Count : out  std_logic_vector ( bitlength-1 downto 0);
                carry : out std_logic);
end component;

begin
clkDivider25k6 : counter generic map (1900,11,1)
                         port map ( clk,reset,enable,'0',open,freq25k6);
Counter_3_stage: StageCounter   generic map (7,3,3)
      port map (freq25k6,reset,enable,'0',xyz,open);
clkDivider10 : counter generic map (5000000,24,1)
      port map ( clk,reset,enable,'0',open,freq10);
```

```
RichtingCounter : counter generic map (2,2,1)
 port map ( RichtingChange,not reset,enable,'0',RichtingData,colorChange);
colorCounter : counter generic map (2,2,1)
 port map ( colorChange,not reset,enable,'0',colorOut,open);
Vlakcounter : counter generic map (7,3,1)
 port map ( freq10,not reset,enable,'0',richting,richtingchange);
FreeOut<=FreeIn;

process ( richtingchange)
begin
case richtingData is
    WHEN  "00"  =>
            x <= xyz( 2 downto 0);
            y <= xyz( 5 downto 3);
            z <= richting;
    WHEN  "01"  =>
            x <= xyz( 2 downto 0);
            z <= xyz( 8 downto 6);
            y <= richting;
    WHEN  "10"  =>
            y <= xyz( 5 downto 3);
            z <= xyz( 8 downto 6);
            x <= richting;
  WHEN OTHERS =>
            x <= xyz( 2 downto 0);
            y <= xyz( 5 downto 3);
            z <= xyz( 8 downto 6);

  END CASE;

end process;
end Structural;
```

Bijlage 12: VHDL PWM manueel dimbare kubus

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;

entity LedCube_Exercise_04 is
    Port (
  Clk : in  STD_LOGIC; -- 50 MHz system Clock
  enable : in  STD_LOGIC; -- Switch Input
  reset : in  STD_LOGIC;  -- Button Input
  LightLevel : in STD_LOGIC_VECTOR (7 downto 0);  -- Switch vector Input
  colorOut : out STD_LOGIC_VECTOR ( 1 downto 0); -- Color selection of
      leds: "00" = Blue / "01" = Red / "10" = Green / "11" = OFF
  x : out STD_LOGIC_VECTOR ( 2 downto 0); -- X Coordinate, value 0 - 7
  y : out STD_LOGIC_VECTOR ( 2 downto 0); -- Y Coordinate, value 0 - 7
  z : out STD_LOGIC_VECTOR ( 2 downto 0); -- Z Coordinate, value 0 - 7
  FreeIn : in STD_LOGIC_VECTOR ( 8 downto 0); -- Free to use Input
  FreeOut : out STD_LOGIC_VECTOR ( 8 downto 0)); -- Free to use Output

end LedCube_Exercise_04;

architecture Structural of LedCube_Exercise_04 is

signal xyz : std_logic_vector ( 8 downto 0);
signal pwmsign : std_logic_vector ( 0 downto 0);
signal color : std_logic_vector ( 1 downto 0);

component StageCounter is
    generic (Countlimit : integer := 7;
             Stages : integer := 3;
             BitLength : integer := 3);
    Port    (Clk : in  STD_LOGIC;
             Reset : in  STD_LOGIC;
             Enable : in  STD_LOGIC;
             UpDown : in  STD_LOGIC;
             Stage : out  STD_LOGIC_VECTOR ((stages*bitlength-1) downto 0);
             carry : out STD_logic_VECTOR ( stages-1 downto 0));
end component;

component pwm IS
  GENERIC(
      sys_clk: INTEGER := 50_000_000; --system clock frequency in Hz
      pwm_freq: INTEGER := 100_000; --PWM switching frequency in Hz
      bits_resolution: INTEGER := 8;--bits of resolution setting the duty
cycle
      phases: INTEGER := 1); --number of output pwms and phases
  PORT(
      clk: IN  STD_LOGIC;    --system clock
      reset_n: IN  STD_LOGIC; --asynchronous reset
      ena: IN  STD_LOGIC; --latches in new duty cycle
      duty: IN  STD_LOGIC_VECTOR(bits_resolution-1 DOWNTO 0); --duty cycle
      pwm_out: OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0); --pwm outputs
      pwm_n_out: OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0)); --pwm inverse
END component;
```

```
begin

dimmer : pwm
generic map (50000000,25600,8,1)
port map (clk,not reset,enable,LightLevel(7 downto 0),pwmsign,open);
looper : stagecounter
generic map (7,3,3)
port map (pwmsign(0),not reset,not enable,'0',xyz,open) ;


x<=xyz(2 downto 0);
y<=xyz(5 downto 3);
z<=xyz(8 downto 6);
colorOut<=color;
FreeOut<=FreeIn;
process ( pwmsign)
begin
if pwmsign(0) ='1' then
    color <="00";
else
    color <="11";
end if;
end process;
end Structural;
```

Bijlage 13: VHDL PWM automatisch dimbare kubus

```
 library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;

entity LedCube_Exercise_05 is
    Port (
  Clk : in  STD_LOGIC; -- 50 MHz system Clock
  enable : in  STD_LOGIC; -- Switch Input
  reset : in  STD_LOGIC;  -- Button Input
  colorOut : out STD_LOGIC_VECTOR ( 1 downto 0); -- Color selection of
      leds: "00" = Blue / "01" = Red / "10" = Green / "11" = OFF
  x : out STD_LOGIC_VECTOR ( 2 downto 0); -- X Coordinate, value 0 - 7
  y : out STD_LOGIC_VECTOR ( 2 downto 0); -- Y Coordinate, value 0 - 7
  z : out STD_LOGIC_VECTOR ( 2 downto 0); -- Z Coordinate, value 0 - 7
  FreeIn : in STD_LOGIC_VECTOR ( 8 downto 0); -- Free to use Input
  FreeOut : out STD_LOGIC_VECTOR ( 8 downto 0)); -- Free to use Output

end LedCube_Exercise_05;

architecture Structural of LedCube_Exercise_05 is

signal xyz : std_logic_vector ( 8 downto 0);
signal pwmsign : std_logic_vector ( 0 downto 0);
signal color : std_logic_vector ( 1 downto 0);
signal dimlevel : std_logic_vector ( 7 downto 0);
signal clk100 : std_logic;
signal upDown : std_logic :='0';
signal colorchange : std_logic :='0';
signal colorselect :std_logic_vector ( 1 downto 0);

component StageCounter is
    generic (Countlimit : integer := 7;
             Stages : integer := 3;
             BitLength : integer := 3);
    Port    (Clk : in  STD_LOGIC;
             Reset : in  STD_LOGIC;
             Enable : in  STD_LOGIC;
             UpDown : in  STD_LOGIC;
             Stage : out  STD_LOGIC_VECTOR ((stages*bitlength-1) downto 0);
             carry : out STD_logic_VECTOR ( stages-1 downto 0));
end component;

component pwm IS
  GENERIC(
      sys_clk: INTEGER := 50_000_000; --system clock frequency in Hz
      pwm_freq: INTEGER := 100_000;    --PWM switching frequency in Hz
      bits_resolution : INTEGER := 8; --bits of resolution setting the duty
cycle
      phases: INTEGER := 1);         --number of output pwms and phases
  PORT(
      clk: IN  STD_LOGIC; --system clock
      reset_n: IN  STD_LOGIC;  --asynchronous reset
      ena: IN  STD_LOGIC;  --latches in new duty cycle
      duty: IN  STD_LOGIC_VECTOR(bits_resolution-1 DOWNTO 0); --duty cycle
      pwm_out   : OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0); --pwm outputs
      pwm_n_out : OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0)); --pwm inverse
END component;
```

```
component Counter is
    Generic (   CountLimit : INTEGER := 200;
                bitlength : integer :=8;
                increment : integer :=1);
        Port(   Clk : in  STD_LOGIC;
                Reset : in  STD_LOGIC; -- High active
                Enable : in  STD_LOGIC; -- High active
                UpDown : in  STD_LOGIC; -- High -> up
                Count : out  std_logic_vector ( bitlength-1 downto 0);
                carry : out std_logic);
end component;

begin

dimmer : pwm
generic map (50000000,50000,8,1)
port map (clk,not reset, not enable,dimlevel,pwmsign,open);
looper : stagecounter
generic map (7,3,3)
port map (pwmsign(0),not reset,not enable,'0',xyz,open) ;
dimchanger : counter
generic map (255,8,1)
port map (clk100,not reset, not enable,UpDown,dimlevel,open);
freqDivder : counter
generic map (2000000,20,1)
port map (clk,not reset, not enable, '0',open,clk100);
colorchanger : counter
generic map (2,2,1)
port map (colorchange,not reset, not enable, '0',colorselect,open);

FreeOut<=FreeIn;
colorOut<=color;
x<=xyz(2 downto 0);
y<=xyz(5 downto 3);
z<=xyz(8 downto 6);

process ( pwmsign)
begin
if pwmsign(0) ='1' then
    color <=colorselect;
else
    color <="11";
end if;
end process;

process ( dimlevel,clk)
begin
if dimlevel >=x"fE" then
    upDown <= '1';
elsif dimlevel <=x"01" then
    UpDown <='0';
    colorchange <='1';
else
    colorchange <='0';
end if;
end process;

end Structural;
```

Bijlage 14: VHDL statisch patroon

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.all;

entity LedCube_Exercise_06 is
    Port (
  Clk : in  STD_LOGIC; -- 50 MHz system Clock
  enable : in  STD_LOGIC; -- Switch Input
  reset : in  STD_LOGIC;  -- Button Input
  colorOut : out STD_LOGIC_VECTOR ( 1 downto 0); -- Color selection of
      leds: "00" = Blue / "01" = Red / "10" = Green / "11" = OFF
  x : out STD_LOGIC_VECTOR ( 2 downto 0); -- X Coordinate, value 0 - 7
  y : out STD_LOGIC_VECTOR ( 2 downto 0); -- Y Coordinate, value 0 - 7
  z : out STD_LOGIC_VECTOR ( 2 downto 0); -- Z Coordinate, value 0 - 7
  FreeIn : in STD_LOGIC_VECTOR ( 8 downto 0); -- Free to use Input
  FreeOut : out STD_LOGIC_VECTOR ( 8 downto 0)); -- Free to use Output


end LedCube_Exercise_06;

architecture Structural of LedCube_Exercise_06 is

signal FrequencyArray : std_logic_vector ( 3 downto 0);
signal color : std_logic_vector ( 1 downto 0);
signal freq25k6 : std_logic;
signal xyz : std_logic_vector ( 8 downto 0);

component StageCounter is
    generic (   Countlimit : integer := 7;
                Stages : integer := 3;
                BitLength : integer := 3);
    Port    (   Clk : in  STD_LOGIC;
                Reset : in  STD_LOGIC;
                Enable : in  STD_LOGIC;
                UpDown : in  STD_LOGIC;
                Stage : out  STD_LOGIC_VECTOR ((stages*bitlength-1) downto
0);
                carry : out STD_logic_VECTOR ( stages-1 downto 0));
end component;

component Counter is
    Generic (   CountLimit : INTEGER := 200;
                bitlength : integer :=8;
                increment : integer :=1);
        Port(   Clk : in  STD_LOGIC;
                Reset : in  STD_LOGIC; -- High active
                Enable : in  STD_LOGIC; -- High active
                UpDown : in  STD_LOGIC; -- High -> up
                Count : out  std_logic_vector ( bitlength-1 downto 0);
                carry : out std_logic);
end component;

component Lookup is
    Port ( X : in  STD_LOGIC_VECTOR (2 downto 0);
           Y : in  STD_LOGIC_VECTOR (2 downto 0);
           Z : in  STD_LOGIC_VECTOR (2 downto 0);
           color : out  STD_LOGIC_VECTOR (1 downto 0));
end component;
```

```
begin

ClockDivider: counter generic map(1900,11,1)
       port map (clk,not reset,enable,'0',open,freq25k6);

Counter_3_stage: Stagecounter generic map ( 7,3,3)
       port map ( clk,not reset,freq25k6,'0',xyz,open);

table : Lookup
      port map ( xyz(2 downto 0),xyz(5 downto 3),xyz(8 downto 6),color);

FreeOut<=FreeIn;
colorOut<=color;
x<=xyz(2 downto 0);
y<=xyz(5 downto 3);
z<=xyz(8 downto 6);
end Structural;
```

Bijlage 15: VHDL statisch patroon Lookuptable

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Lookup is
    Port ( X : in  STD_LOGIC_VECTOR (2 downto 0);
           Y : in  STD_LOGIC_VECTOR (2 downto 0);
           Z : in  STD_LOGIC_VECTOR (2 downto 0);
           color : out  STD_LOGIC_VECTOR (1 downto 0));
end Lookup;

architecture Behavioral of Lookup is
type xContainer is array ( integer range 0 to 7) of std_logic_vector ( 1
downto 0);
type zContainer is  array ( integer range 7 downto 0) of xContainer;


signal  xBuffer : xContainer;
signal PortdataBuf : std_logic_vector ( 1 downto 0 );
constant STable : zContainer :=
(("00","11","11","11","11","11","11","00"),

("11","00","11","11","11","11","00","11"),

("11","11","00","11","11","00","11","11"),

("11","11","11","00","00","11","11","11"),

("11","11","11","00","00","11","11","11"),

("11","11","00","11","11","00","11","11"),

("11","00","11","11","11","11","00","11"),

("00","11","11","11","11","11","11","00"));
begin
xBuffer <= STable(to_integer(unsigned(Z)));
color <= xBuffer(to_integer(unsigned(X)));

end Behavioral;
```