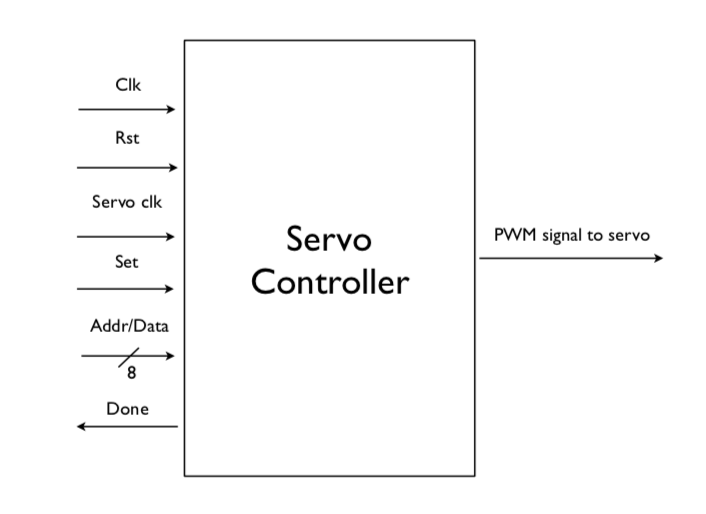
Verslag

# Inleiding

De doelstelling van dit project is om een servomotor aan te sturen. Hiervoor werd een controller ontworpen in vhdl. Naast de controller werd ook een testbench ontworpen om de werking van de controller te kunnen verifiëren.

# Controller

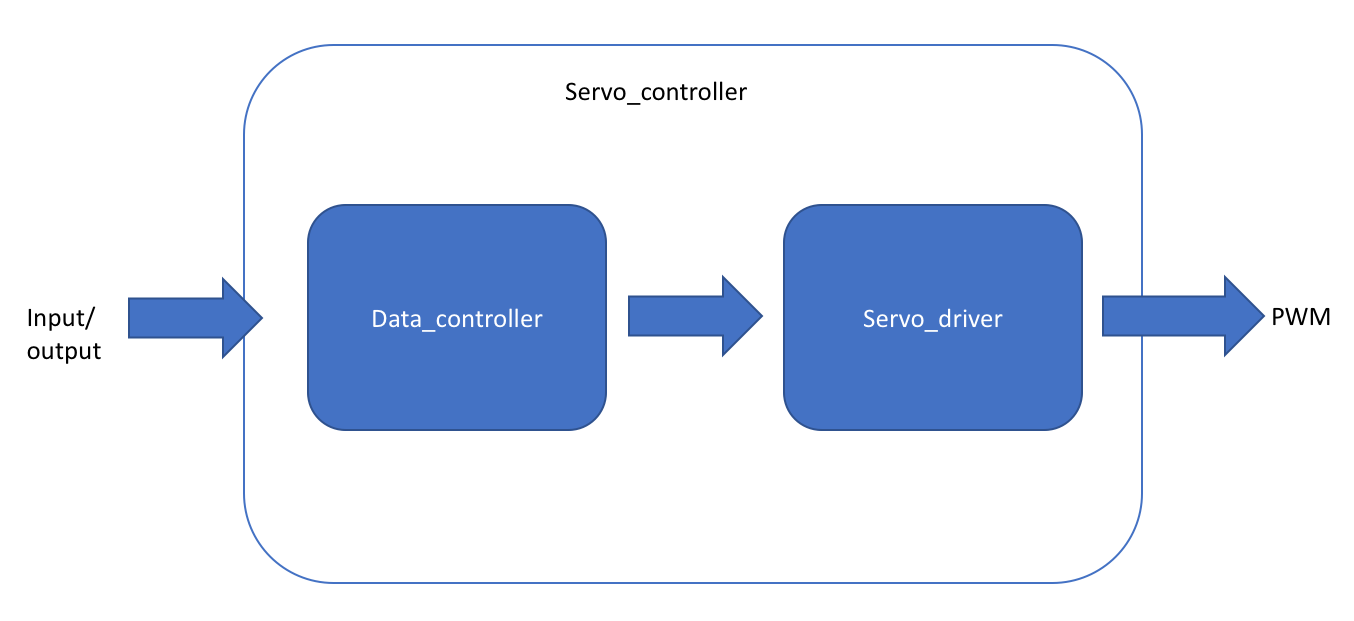
Voor de het besturen van de servomotor werd een controller ontworpen die voldeed aan het schema in Figuur 1. In de vereiste specificaties werd gesteld dat de controller een klokperiode heeft van 20ms.



Figuur 1 schematische voorstelling servocontroller

## Opbouw

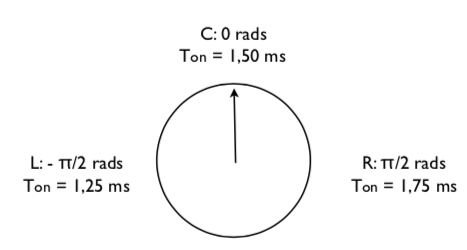
Naar analogie van de labolessen hebben we besloten de controller te ontwerpen uit meerdere entiteiten die elk verantwoordelijk zijn voor hun eigen taak. We maakten een data\_controller-file die verantwoordelijk is voor het wijzigen van staat en een servo-file die verantwoordelijk was voor het uitsturen van het PWM-signaal. De twee componenten worden verbonden met elkaar in de servo\_controller-file (Figuur 2).



Figuur 2 servocontroller schema

## Werking van de motor

De servo-motor heeft 256 mogelijke posities. De positie die de motor zal aannemen wordt bepaald door het PWM-signaal dat we doorsturen. De posities van de motor variëren tussen -pi/2 rad en +pi/2 rad. De duur dat het PWM-signaal hoog staat (Ton) bepaald de positie. Wanneer de Ton 1,25ms is, zal de motor plaats nemen op positie -pi/2 rad. Wanneer de Ton 1,75ms is, zal de motor plaats nemen op positie +pi/2 rad. Alle posities daartussen zijn evenredig verdeeld. Wanneer de Ton dus 1,5ms bedraagt, zal de motor positie 0 innemen. Een grafische voorstelling hiervan is te zien in Figuur 3



Figuur 3 servo posities

Doordat de Ton’s zo klein en fijngevoelig zijn is er een tweede klok nodig. De systeemklok van 20ms is namelijk veel te groot. De servo-klok zal gebruikt worden om het pwm-signaal op te stellen.

De frequentie van de servo-klok moest nog bepaald worden. Doordat de Ton varieert tussen 1,25ms en 1,75ms moesten er nog 256 posities verdeeld worden over 0,5ms.

Een periode van 1,96μs resulteert in een frequentie van 520,2kHz voor de servo-klok.

## Data\_controller

De data\_controller is verantwoordelijk voor het wijzigen van staat. Elke klokpuls wordt gekeken of de staat al dan niet gewijzigd moet worden. De mogelijke staten zijn idle en readingAddress. De staat mag pas readingAddress worden als zowel de set hoog staat als het juiste adres is meegegeven. Indien dit het geval is zetten we de done-flag laag en gaan we naar de volgende staat alwaar de data van de data\_bus wordt gelezen. Deze wordt toegekend aan het data\_out signaal dat vervolgens in de servo-file gebruikt zal worden om de juiste Ton te bepalen.

## Servo

In de servo-file wordt de uitgelezen data gebruikt om de Ton te berekenen. We maken gebruik van een teller die we elke klokpuls resetten. Vervolgens gaan we elke servo\_klokpuls de teller met 1 verhogen. Vervolgens berekenen we aan de hand van de servoklokperiode de tijd dat het pwm signaal hoog is. Vanaf dit de gewenste treshhold bereikt heeft sturen we een nul-signaal naar buiten.

## Servo\_controller

Deze file zal de poorten van de twee voorgaande entiteiten verbinden. Belangrijk is hier dat we een de data\_output uit de data\_controller opvangen in een data signaal in de architectuur. We doen dit omdat we deze zullen moeten doorsturen naar de servo\_file.

Alwaar ze gebruikt wordt om de pwm te berekenen.

# Code

### Data\_controller.vhdl

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity data\_controller is

generic(

        controller\_address : std\_logic\_vector(7 downto 0)

    );

port(

rst : in std\_logic;

clk : in std\_logic;

set : in std\_logic;

data\_bus : in std\_logic\_vector(7 downto 0); --data and address is shared on this bus

data\_out : out std\_logic\_vector(7 downto 0) := (others => '0');

done : out std\_logic := '1';

staat : out std\_logic\_vector(1 downto 0) := "00"

);

end entity;

architecture behaviour of data\_controller is

--we have 2 types of states idle and read

type state is (idle, readAddress);

signal currentState : state := idle;

begin

process(clk,rst)

begin

if (rst = '1') then

--servo to 0 rad ==> data is 127

data\_out <= "01111111";

currentState <= idle;

elsif rising\_edge(clk) then

case currentState is

when idle =>

staat <= "00";

if(set = '1') then

if((data\_bus = controller\_address) or (data\_bus="11111111")) then

currentState <= readAddress;

done <= '0';

end if;

else

done <= '1';

end if;

when readAddress =>

staat <= "01";

data\_out <= data\_bus;

currentState <= idle;

end case;

end if;

end process;

end architecture;

### Servo.vhdl

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

--this entity will tell the motor what to do

entity servo is

port(

rst : in std\_logic;

clk : in std\_logic;

sc: in std\_logic;

-- set : in std\_logic;

data : in std\_logic\_vector(7 downto 0); --data and address

pwm : out std\_logic

);

end entity;

architecture behaviour of servo is

constant servo\_freq : positive := 510200;

    constant servo\_period\_ms : real := 0.001961;

begin

process(sc, clk, rst)

variable pwm\_timer : integer := 0;

begin

if rising\_edge(clk) then

pwm\_timer := 0;

end if;

if rst = '1' then

pwm <= '0';

pwm\_timer := 0;

elsif rising\_edge(sc) then

pwm\_timer := pwm\_timer + 1;

if(real(pwm\_timer) < ((1.25/servo\_period\_ms) + real(to\_integer(unsigned(data))))) then

pwm <= '1';

else

pwm<= '0';

end if;

end if;

end process;

end architecture;

### Servo\_controller.vhdl

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

--this entity will tell the motor what to do

entity servo is

port(

rst : in std\_logic;

clk : in std\_logic;

sc: in std\_logic;

-- set : in std\_logic;

data : in std\_logic\_vector(7 downto 0); --data and address

pwm : out std\_logic

);

end entity;

architecture behaviour of servo is

constant servo\_freq : positive := 510200;

    constant servo\_period\_ms : real := 0.001961;

begin

process(sc, clk, rst)

variable pwm\_timer : integer := 0;

begin

if rising\_edge(clk) then

pwm\_timer := 0;

end if;

if rst = '1' then

pwm <= '0';

pwm\_timer := 0;

elsif rising\_edge(sc) then

pwm\_timer := pwm\_timer + 1;

if(real(pwm\_timer) < ((1.25/servo\_period\_ms) + real(to\_integer(unsigned(data))))) then

pwm <= '1';

else

pwm<= '0';

end if;

end if;

end process;

end architecture;