UNIVERSITEIT ANTWERPEN
Academiejaar 2015-2016

Faculteit Toegepaste Ingenieurswetenschappen

Labo FPGA

4-Computerarchitectuur

Student
Kris Van de Voorde

Bachelor of Science in de industriële wetenschappen

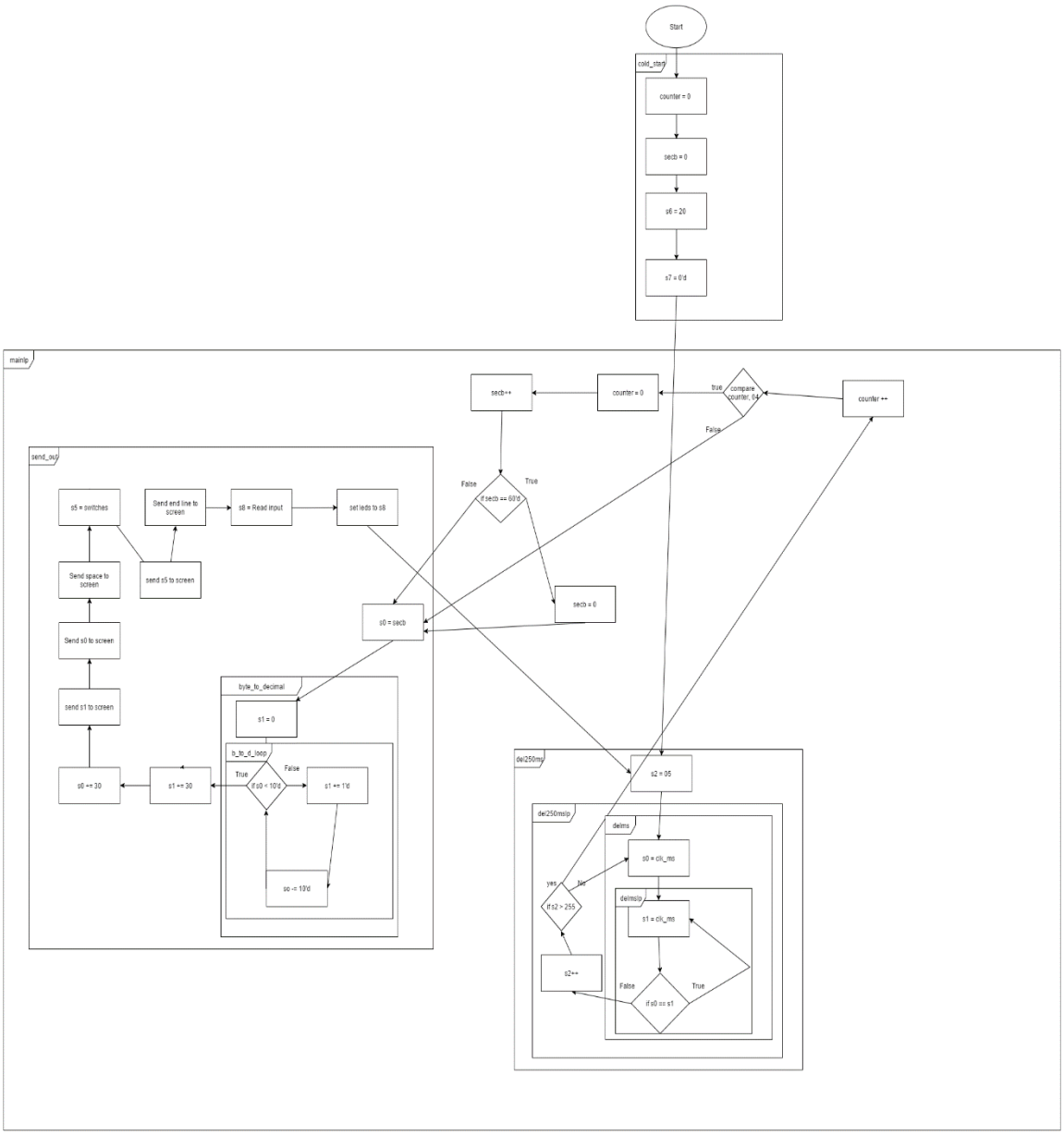
Universiteit Antwerpen

Inhoud

Labo 1

Flowchart

In dit deel wordt beschreven hoe het programma softwarematig werkt. Het programma maakt gebruik van verschillende subroutines. De werking van die subroutines wordt in enkele hoofdstukken toegelicht. De algemene bedoeling van het programma is dat de seconden bijgehouden worden en dat er een input van de gebruiker doorgestuurd kan worden naar de picloblaze en omgekeerd.



Register renaming / declaration

Vooraleer het programma effectief start, worden er nog enkel acties uitgevoerd. Deze acties zijn niet noodzakelijk maar maken het programmeren efficiënter en leesbaarder. Bij “register renaming” wordt ervoor gezorgd dat enkele registers die vaak gebruikt worden een zelfgekozen naam krijgen.

Bij declaration wordt ervoor gezorgd dat er bij het gebruik van de I/O geen hexadecimale getallen gebruikt moeten worden. Er wordt een zelfgekozen naam gegeven aan de hexadecimale waardes.

Cold start

Vanaf deze sectie worden de stappen genoteerd in de flowchart. Cold start wordt één keer aan het begin van het programma uitgevoerd. Op deze manier initialiseren we de variabelen. De ‘counter’ en ‘secb’ worden op nul gezet, terwijl register s6 op 20 gezet wordt. Deze register wordt gebruikt wanneer er nood is aan een spatie op het scherm. Met andere woorden: is 20 de ASCII code voor spatie.

Er wordt gebruikt gemaakt van register s7 voor een “carriage return”. De register s7 kan dan gebruikt worden wanneer er op het scherm naar een volgende lijn gegaan moet worden. De ASCII code hiervoor is 0D.

Mainlp

Dit is het hart van het programma. Het is de loop die oneindig blijft lopen. In het begin van deze loop wordt er gebruik gemaakt van een subroutine genaamd ‘del250ms’. Kort samengevat zorgt die subroutine ervoor dat er 250 ms gewacht wordt voordat er verder gegaan wordt met de mainloop.

In de mainloop wordt door het gebruik van een counter variabele gekeken wanneer er een seconde gepasseerd is. Omdat er 250 ms gewacht wordt, moet er eerst vier keer geloopd worden. Wanneer er een seconde gepasseerd is, wordt de counter variabele terug op nul gezet. Het is dan de bedoeling dat de variabele ‘secb’ met één verhoogd wordt.

In dit stuk is het de bedoeling dat we de seconden behandelen en verhogen met één. Wanneer de seconden groter is dan de decimale waarde 60 moeten de seconden teruggezet worden naar nul. Er wordt in dit stuk gebruik gemaakt van de variabelen ‘secb’ die de hexadecimale waarde bijhoudt van de seconden.

Als de waarde niet gelijk is aan de decimale waarde 60, wordt er rechtstreeks doorgegaan naar de subroutine ‘send_out’. Anders wordt eerst de bewerking gedaan die ‘secb’ terug op nul zet en wordt er dan pas doorgegaan naar de subroutine ‘send_out’. Kort samengevat zorgt de subroutine ‘send_out’ ervoor dat het scherm geüpdate wordt.

Send_out

In deze subroutine wordt de subroutine ‘byte_to_decimal’ opgeroepen om de bytes die in ‘secb’ zitten over te zetten naar een decimaal getal. Wanneer het een decimaal getal is, voegen we er de waarde 30 aan toe zodat het getal een juiste ASCII waarde bevat.

De register s0 stelt de tientallen van het getal voor terwijl s0 de eenheden voorstellen. Deze registers worden gebruikt voor de output, samen met het register waar de spatie inzit. Doormiddel van de mutiplexer worden ook de binaire waarden van de switches ingelezen. Deze waarden wordt daarna op het scherm getoond. Na deze waardes wordt er gebruik gemaakt van de cariage return. De invoer via de UART wordt opgeslagen in register s8. Daarna kan dit register gebruikt worden om de leds in binaire vorm te laten branden. Op het einde van deze subroutine wordt er terug gegaan naar de mainloop.

Delms / delmslp

In dit deel wordt er een delay uitgevoerd van één milliseconde. Dit wordt gedaan aan de hand van de clock van de PicoBlaze. In register s0 wordt één keer de input van de clock gestoken. Er wordt dan over gegaan naar een subroutine ‘delmslp’. In deze subroutine wordt in register s1 ook de clock output gestoken. Dit wordt constant herhaald totdat s1 en s0 niet meer overeenkomen. Dat betekent dat er één milliseconde voorbij is gegaan.

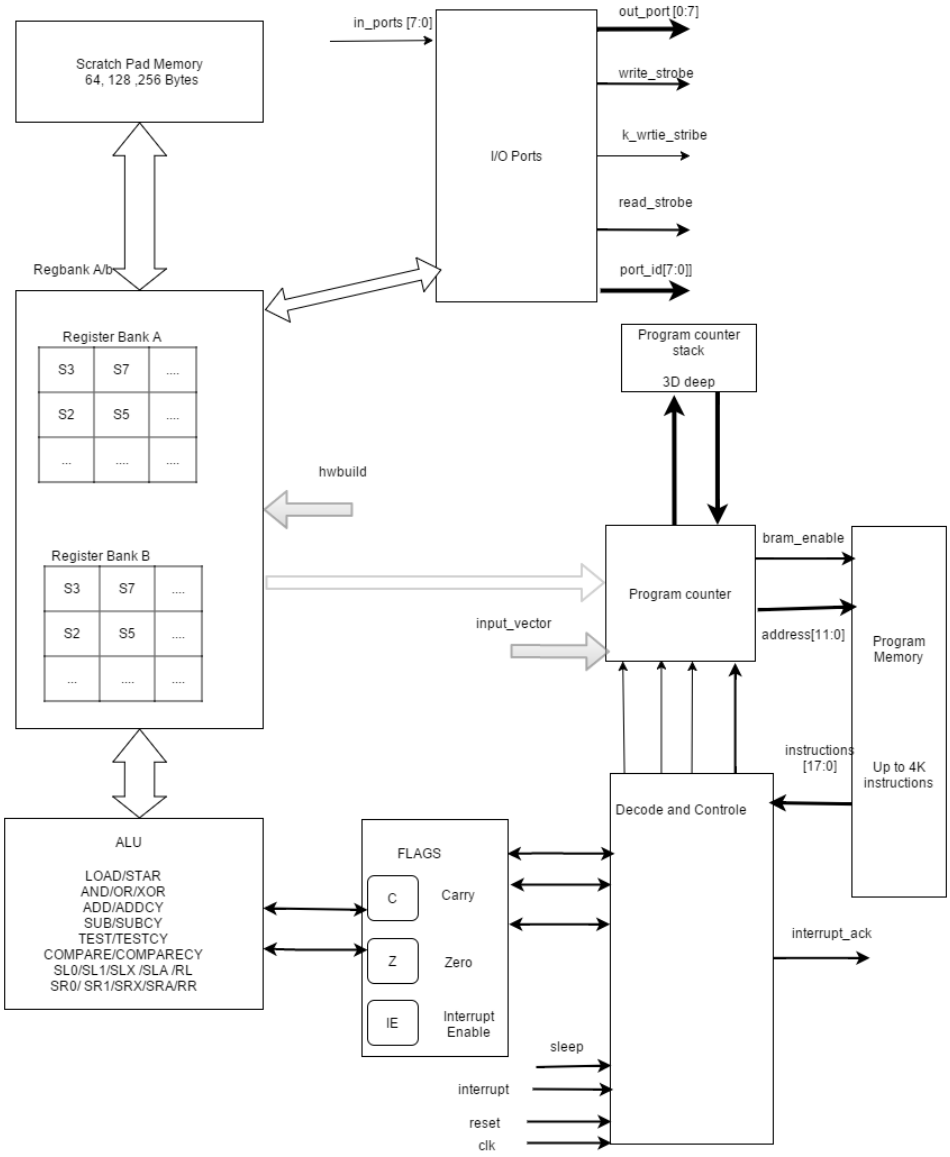
Del250ms/ del250mslp

Dit is de subroutine die gebruikt wordt om de mainloop 250ms te blokkeren. Eerst wordt in del250ms register s2 geïnitialiseerd op 05. Dan wordt er gebruik gemaakt van de subroutine ‘del250mslp’ die gebruik maakt van ‘delms’. Deze loop wordt 250 keer herhaald voordat er verder kan gegaan worden met de mainloop. Zo wordt de delay van 250 ms uitgevoerd.

Byte_to_decimal / b_to_loop

De bedoeling van dit stuk is dat het mogelijk wordt om hexadecimale waardes om te zetten naar decimale waardes. ‘Byte_to_decimal’ initialiseert register s1 die de tientallen moet voorstellen. Register s0 wordt als een soort van buffer gebruikt, waar de waardes gezet worden die omgezet moeten worden naar decimaal. In de subroutine ‘b_to_loop’ wordt er geloopd zolang s0 groter is dan 10 decimaal. Wanneer de waarde kleiner is dan 10 wordt er teruggegaan naar de main_loop. Wanneer dit niet het geval is, wordt er 10 decimaal afgetrokken van s0 en wordt s1 met één verhoogd. De waarden s0 en s1 kunnen gebruikt worden als decimaal getal.

Blokschema



Scratch pad memory

Scrath pad memory wordt gebruikt om data op te slagen die groter is dan wat de registers kunnen opslagen. De grootte van de data kan variëren tussen 64, 128 en 256 bytes.

Regbank A/B

Er zijn twee banken van elk zestien registers. De 8-bit informatie die in deze registers opgeslagen wordt, komt meestal vanuit de input ports. De informatie kan dan doorgestuurd worden naar de output ports. Standaard wordt er gebruik gemaakt van bank 'A', maar om meerdere subroutines te gebruiken, kan er ook gebruik gemaakt worden van bank 'B', of wanneer het om een grote hoeveelheid data gaat, kan deze doorgestuurd worden naar de scratch pad memory.

ALU

De Algoritmic logic controller wordt gebruikt om data in de registers te manipuleren. De ALU houdt instructies in zoals: AND, OR,… De ALU communiceert ook met de flags die dan invloed hebben op de volgorde van de uitvoering van het programma.

Flags

Er worden drie verschillende flags gebruikt:

- Carry flag: Deze flag wordt op verschillende manieren gebruikt. Hij wordt onder andere op één gezet wanneer een bit waarde te groot wordt bij een optelling.
- Zero flag: Wanneer je verschillende bits van elkaar zou aftrekken en een nul zou uitkomen, wordt deze flag op één gezet.
- Interrupt enbale flag: Geeft aan of het programma interrupts toelaat.

Decode and control

De decode and control zorgt voor de flow van de applicatie. Het kan de program counter aanpassen gebaseerd op de informatie die het krijgt van de flags. Hier worden dan ook de instructies behandeld zoals 'JUMP' en 'CALL'. De decode and control verwerkt de instructies. Het geeft ook informatie door aan de ALU en I/O poorten. Er zijn nog enkele externe instructies zoals sleep die invloed hebben op dit blok.

Program memory

In de program memory staan alle instructies van het programma. Doormiddel van de program counter die het adres van de instructie doorstuurt, kan het programma weten welke instructie het moet doorgeven aan de decode and control block.

Program counter

De program counter wordt gebruikt om de juiste instructie van de program memory op te halen. Normaal gezien begint een programma altijd van nul.

Program counter stack

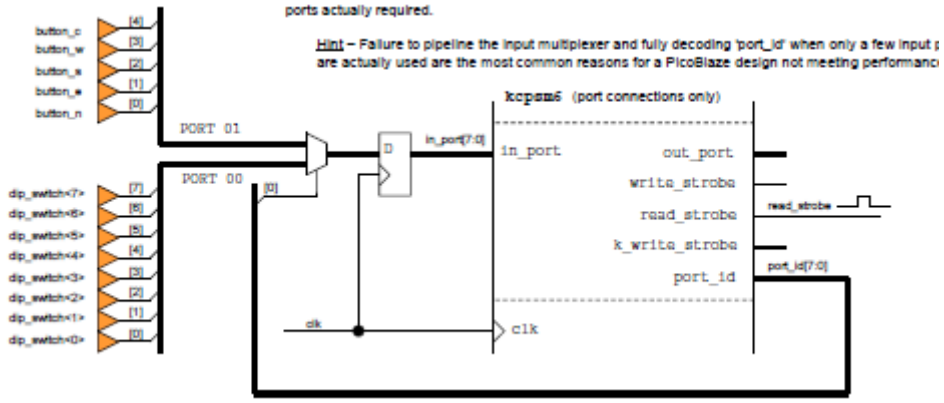
De program counter stack wordt gebruikt wanneer er een interrupt of een call plaatsvindt. Het houdt de program counter bij van waar het programma gebleven is. Zodat wanneer er een “RETURN” gedaan wordt het programma terug kan gaan naar de juiste flow. Er is maar een mogelijkheid van 30 subroutines.

I/O Ports

In I/O wordt er een interactie met de gebruiker van het programma gemaakt.

Input:

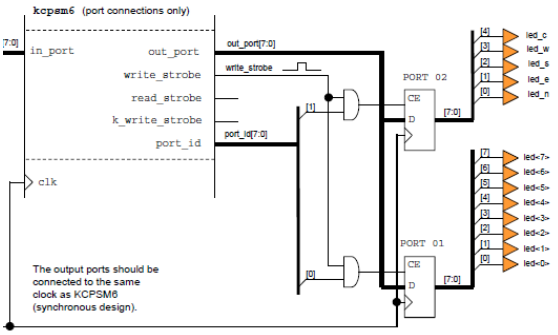
Bij het gebruik van de instructie ‘INPUT zet het de ‘port_id’ voor één van de 256 poorten te specificeren. Het is de taak van het hardware circuit dat er 8-bit data bij de in_port geraakt. Wanneer er gebruik gemaakt wordt van de instructie ’INPUT’ wordt ‘read_strobe’ op één gezet doormiddel van een pulse.



Output:

Bij het gebruik van de instructie ‘OUTPUT’ zet het de ‘port_id’ voor één van de 256 poorten. Daardoor kan de 8-bit data op de “out_port” doorgestuurd worden naar de juiste poort. Er wordt één clock cycle signaal gegenereerd om de write_strobe in te schakelen. Op die manier kan de output poort gebruikt worden.

‘OUTPUTK’ is een uitzondering. Het grote verschil bij deze instructie is dat er enkel gebruikt kan gemaakt worden van de vier laagste port_id bits. Het grote voordeel is dat er geen register als tweede parameter meegestuurd moet worden, maar dit kan een getal, hexadecimaal of binaire waarde zijn. Door het gebruik van ‘OUTPUTK’ kan de clock cycle van de instructies verminderd worden.



Code aanpassing klok

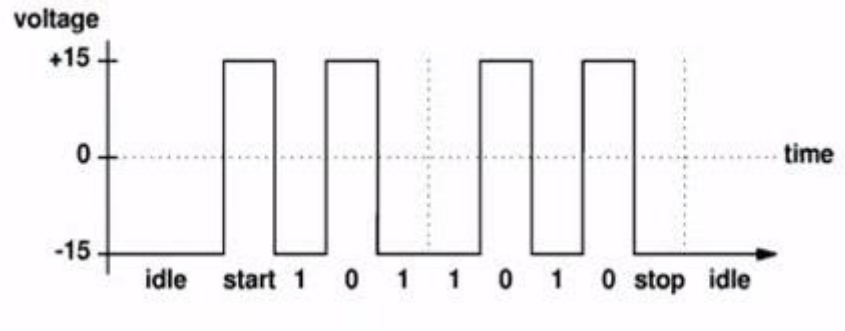
testprogmklok.psm

Assembler versturen via RS232

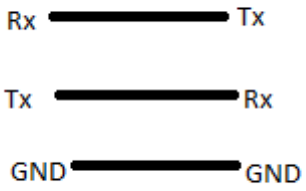
Inleiding RS232

Wanneer er gebruik gemaakt wordt van de UART, kan dit gedaan worden doormiddel van RS232. RS232 is een standaard die vaak gebruikt wordt in seriële communicatie tussen dergelijke apparaten.

Het is een asynchrone communicatie, dat wil zeggen dat wanneer er geen communicatie is, er ook geen data verzonden wordt. Er wordt meestal gebruik gemaakt van 7 tussen bits, een startbit en een stopbit.



Bij RS232 wordt er vooral gedacht aan een point to point connectie. Er wordt gebruik gemaakt van een cross over om de communicatie tot stand te brengen.



De transmitter wordt geconnecteerd met de Receiver en omgekeerd.

Programma met RS232

De bedoeling van het programma zoals het er nu staat, is dat er data vanuit de microcontroller verstuurd kan worden via het RS232 protocol via de TX. Er kan ook data ontvangen worden op de RX. Het programma zit in een oneindige loop. In die loop worden er enkele subroutines opgeroepen.

Rddata

Dit is de eerste subroutine die in het programma gebruikt wordt. In deze subroutine wordt gecontroleerd of de waarde die register 'inputb' bevat een '$' teken is. Als dit het geval is, weet het programma dat er een mogelijkheid is om een commando te starten.

Strtcmd

Nu het programma weet dat er een '$' teken gepasseerd is, kan het programma commando's van elkaar onderscheiden. Afhankelijk van de input springt het programma naar de juiste subroutine. Mocht er een 'A' ingegeven worden dan zal het programma naar de subroutine 'acmd' springen. Mocht er een 'B' ingegeven worden zal het programma naar de subroutine 'bcmd' springen. Mocht er een waarde gegeven worden die niet van toepassing is, zal het programma zich resetten en opnieuw loopen. Dit wordt gedaan tot dat het programma aan 'ACC' zit.

_CMD/_CMD2

Het programma heeft al 'ACC' als input gehad. Het programma moet als volgt drie keer het teken '_' binnenkrijgen. Dit wordt gedaan aan de hand van een subroutine die controleert of de tekens overeenkomen. Wanneer er een verkeerd teken ingegeven wordt, zal het programma zich resetten. Wanneer het programma 'ACC___' gepasseerd is, zal er nog gecontroleerd worden of er een spatie in het commando zit. Als het commando een spatie bevat, is het de bedoeling dat het programma de X, Y en Z leest.

Cmddata1

Het programma heeft nu 'ACC___' ontvangen. Er wordt eerst nog gecontroleerd op een spatie. Daarna kunnen de waardes tussen 300 en 1300 ingelezen worden. Per getal worden er vier registers gebruikt. In het eerste register wordt er een één of een nul gestoken, afhankelijk of het getal hoger is dan duizend. Per cijfer in een getal wordt er gecontroleerd of de input een cijfer is. Als het geen cijfer is, wordt er terug gesprongen naar de 'mainlp'. Wanneer het wel een cijfer is, dan wordt het cijfer in het register gestoken. Per cijfer wordt een register gebruikt.

Psarc

Wanneer het commando compleet is, wordt er gesprongen naar ‘psarc’. In dit geval gaat ‘psarc’ gebruik maken van RS232 om een één door te sturen.

Rdbuff / Bfrempty

Dit is een belangrijke subroutine in het programma. Met deze subroutine kan het programma nagaan of er iets in de receiver zit. Het programma doet dit doormiddel van de buffer. De subroutine controleert eerst of de buffer niet vol zit. Als de buffer niet vol zit, gaat het programma na of de buffer niet leeg is zodat het commando niet onnodig gereset wordt. De subroutine blokkeert het programma en wacht tot er iets in de UART geschreven wordt. Dan kan er doorgegaan worden met het programma. Deze subroutine wordt meestal in een call opgeroepen omdat de microprocessor moet weten waar het naar toe moet springen als de subroutine afgehandeld is.

Nxtknmbr

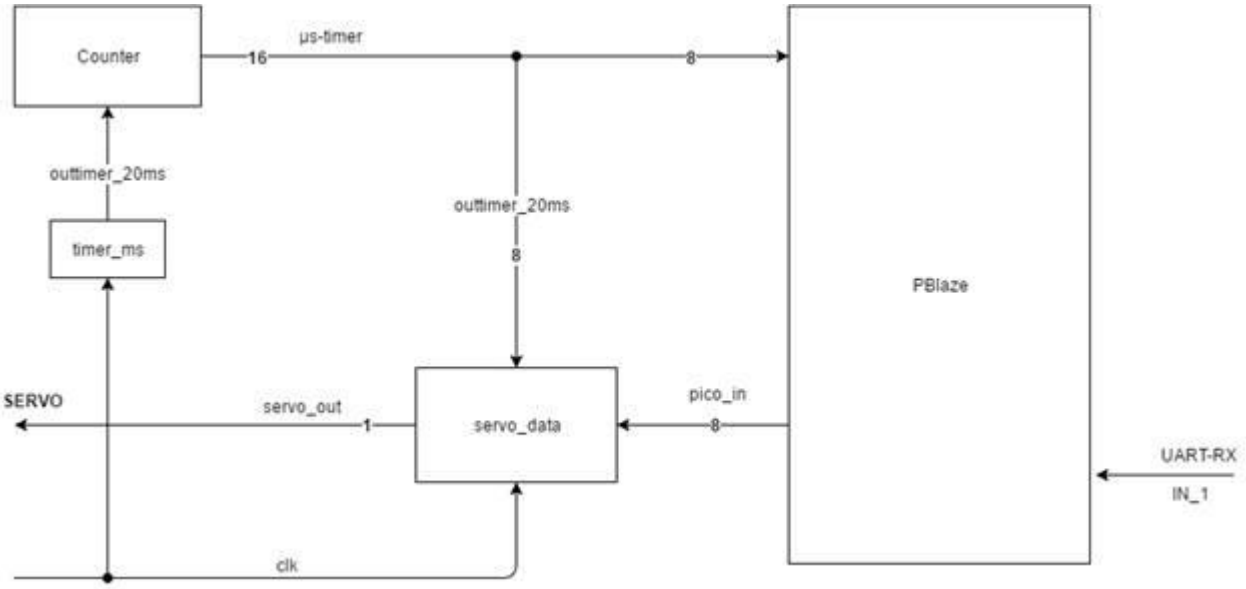
Dit is een handige subroutine die gebruikt wordt als er een getal tussen de nul en de tien verwacht wordt. Wanneer er iets anders binnenkomt, wordt er gesprongen naar de ‘mainlp’. Deze subroutine wordt meestal in een call opgeroepen omdat de microprocessor moet weten waar hij naar toe moet springen als de subroutine afgehandeld is.

Code

codeLabo2.psm

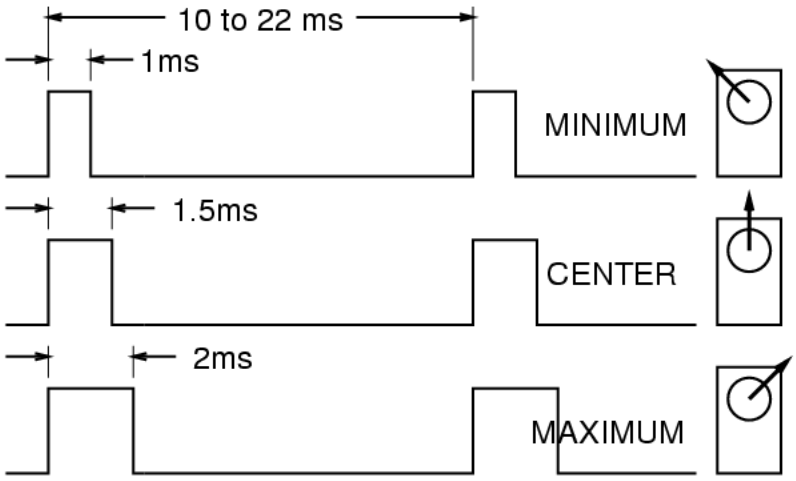
Motorcontroller aansturen in VHDL

Ik had bij dit stuk geen voorkennis van VHDL. De bedoeling van VHDL is dat er een 8-bit input signaal omgezet kan worden naar een output die een servo motor kan aansturen.



Deze afbeelding is een samenvatting van hoe het VHDL programma moet werken. Er komt data binnen via de UART-RX die verbonden zou zijn met de UART-TX van een microcontroller. Deze data wordt verwerkt in de assembler code.

De PicoBlaze haalt 8-bit data uit het commando dat via een microcontroller binnenkomt. In VHDL wordt dan de PWM (Pulse Width Modulation) berekend.



De bedoeling van zo een PWM is dat er elke 20 milliseconden informatie naar de PicoBlaze gestuurd wordt. Per microseconde wordt er een één of een nul gestuurd naar de servo. De servo bekijkt deze data per 20 milliseconden. In de door ons gebruikte servo's wordt er naar een puls gekeken die kan variëren tussen één of twee milliseconden. 1.5 milliseconden wordt behandeld als neutraal. Twee milliseconden wordt behandeld als -90°. Één milliseconde wordt behandeld als 90°.

Om dit te realiseren wordt er gebruik gemaakt van de clock in de PicoBlaze. Er wordt een module gebruikt die de aantal µseconden van een cycle bijhoudt.

Code van de µcounter

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

-- uncomment the following library declaration if using
-- arithmetic functions with signed or unsigned values
--use ieee.numeric_std.all;

-- uncomment the following library declaration if instantiating
-- any xilinx primitives in this code.
--library unisim;
--use unisim.vcomponents.all;

ENTITY timer IS
	PORT (
		clk : IN std_logic;
		outtimer_20ms : OUT NATURAL RANGE 0 TO 20000
	);

END timer;

ARCHITECTURE behavioral OF timer IS
	SIGNAL prescaler : NATURAL RANGE 0 TO 99 := 0; --getal dat zorgt dat er gerekend wordt in µseconden.
	SIGNAL count_20ms : NATURAL RANGE 0 TO 20000; --getal dat de aantaal µseconden bijhoudt tot er een nieuwe cycle is.
BEGIN
	counter_process : PROCESS(clk)
	BEGIN
		IF rising_edge(clk) THEN

			IF prescaler = 99 THEN -- er is een µseconden voorbij.
				prescaler <= 0;
				IF count_20ms = 20000 THEN -- Er zijn 20 microseconden voorbij.
					count_20ms <= 0; -- de cycle begint opnieuw.
				ELSE
					count_20ms <= count_20ms + 1; -- Er wordt een 1 bijgeteld bij de cycle
				END IF;
			END IF;
			IF prescaler < 99 THEN -- Er moet geteld worden op de µseconden, niks laten gebeuren want we zitten tussen een µsecond
				prescaler <= prescaler + 1;
			END IF;
		END IF;
	END PROCESS;
	outtimer_20ms <= count_20ms; --We geven de aantal µseconden terug van een huidige cycle
END behavioral;
```

Nu dat de cycle bijgehouden kan worden, is het de bedoeling dat er berekend wordt hoelang een PWM moet duren in een cycle. Dit wordt gedaan doormiddel van een andere module.

Code van Servo_data

Deze module heeft twee verschillende einddoelen in het eerste proces, wordt er per cycle de aantal milliseconden van een PWM berekend. In het tweede proces wordt het uitgangsignaal verstuurd. Dit proces geeft per µsecond aan of er een één of een nul doorgegeven moet worden aan de servo.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;

-- uncomment the following library declaration if using
-- arithmetic functions with signed or unsigned values
--use ieee.numeric_std.all;

-- uncomment the following library declaration if instantiating
-- any xilinx primitives in this code.
--library unisim;
--use unisim.vcomponents.all;

ENTITY servo IS
	PORT (
		clk : IN std_logic;
		pico_in : IN std_logic_vector(7 DOWNTO 0); -- data die binnenkomt van de RS232
		servo_out : OUT std_logic; -- bit die gebruikt wordt voor de status van de PWM
		outtimer_20ms : IN NATURAL RANGE 0 TO 20000 --gebruiken voor 2000 µsecoden(cycle)
	);
END servo;

ARCHITECTURE behavioral OF servo IS

	SIGNAL time_servo : INTEGER RANGE 1000 TO 2000 := 1500; -- Hoelang een pulse moet duren;
1500 is a neutral pulse
	SIGNAL data_in : std_logic_vector(7 DOWNTO 0); --extra signaal voor pic_in
	SIGNAL output : std_logic := '0'; --extra signaal voor servo_out

BEGIN
	servo_time : PROCESS(clk, pico_in, outtimer_20ms) -- Bereken de aantal µseconden van een
pulse
	BEGIN
		IF rising_edge(clk) THEN
			IF outtimer_20ms = 0 THEN -- nieuwe cycle begint
				IF pico_in > "11111010" THEN -- 255. Anders gaat erover de 2000
µseconde gezeten worden.
					data_in <= "11111010"; -- data gelijk zetten aan 250
				ELSE
					data_in <= pico_in;
				END IF;
				time_servo <= 1000 + (to_integer(unsigned(data_in))) * 4; --Bereken
de tijd van de pulse (tussen 1000-2000)
			END IF;
		END IF;
	END PROCESS;

	servoprocess : PROCESS(clk, outtimer_20ms, time_servo)
	BEGIN
		IF rising_edge(clk) THEN
			IF time_servo > outtimer_20ms THEN -- zolang de time_servo niet boven de
lengte van de cycle zit worden er een 1 doorgestuurd.
				output <= '1';
			ELSE
				output <= '0'; -- anders een 0 sturen
			END IF;
```

```
                END IF;
        END PROCESS;

        servo_out <= output; -- signaal terug doorgeven;

END behavioral;
```

Code verbinden

Dit zijn de signalen die gebruikt worden in de top_module om de modules met elkaar te connecteren.

```
SIGNAL outtimer_20ms : NATURAL RANGE 0 TO 20000; --Houdt de cycle bij.
SIGNAL pico_in : std_logic_vector(7 DOWNTO 0);
```

In de volgende code wordt de ‘timer’ module met de klok verbonden. Ook wordt de cycle output van de ‘timer’ module verbonden met de ‘servo’ module. Als output geeft de ‘servo’ module een één of een nul door aan de ‘servo_out’ poort.

```
COMPONENT servo
        PORT (
                clk : IN std_logic;
                pico_in : IN std_logic_vector(7 DOWNTO 0);
                servo_out : OUT std_logic;
                outtimer_20ms : IN NATURAL RANGE 0 TO 20000
        );
END COMPONENT;
--
--
--
COMPONENT timer
        PORT (
                clk : IN std_logic;
                outtimer_20ms : OUT NATURAL RANGE 0 TO 20000
        );
END COMPONENT;
Pico_in <= pico_in;
timer_ms : timer
PORT MAP(clk => clk, outtimer_20ms => outtimer_20ms );
servo_data2 : servo
PORT MAP(clk => clk, outtimer_20ms => outtimer_20ms, pico_in => pico_in, servo_out => servo_out);
```

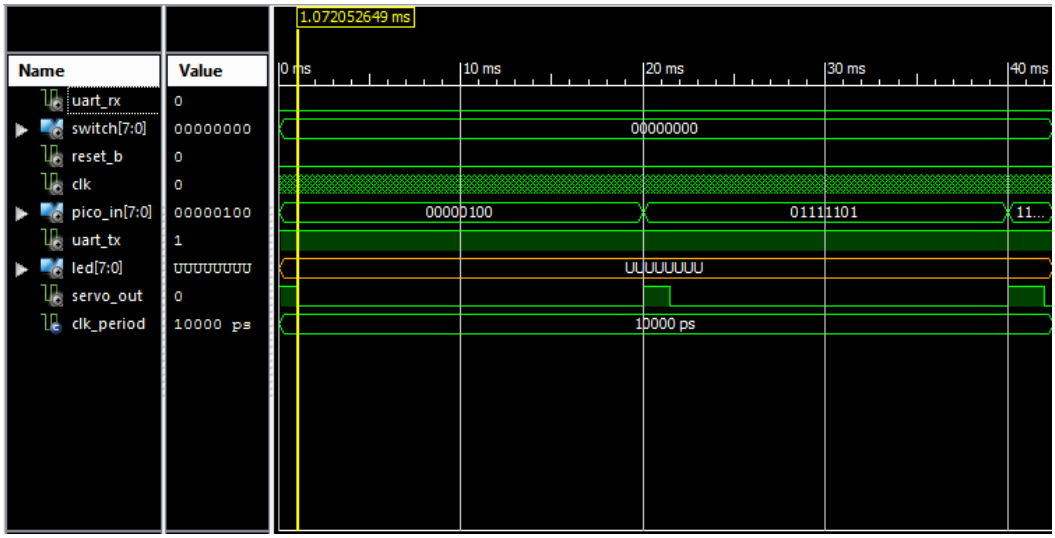
Code testbench

Er wordt gebruik gemaakt van een testbench. Doormiddel van een testbench kan er getest worden of de VHDL de juiste puls genereerde. Een testbench kan gebruikt worden om een omgeving te simuleren.

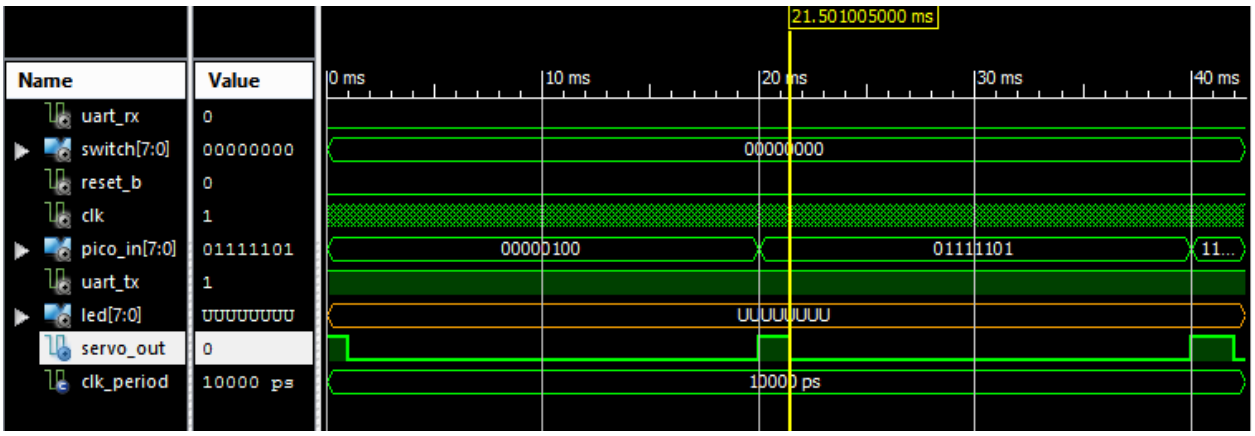
```
stim_proc : PROCESS
BEGIN
        --4
pico_in <= "00000100";
-- insert stimulus here

WAIT FOR 20 ms;

--125
pico_in <= "01111101";
-- insert stimulus here

WAIT FOR 20 ms;
```

```
--255
pico_in <= "11111111";

WAIT FOR 20 ms;
END PROCESS;
```

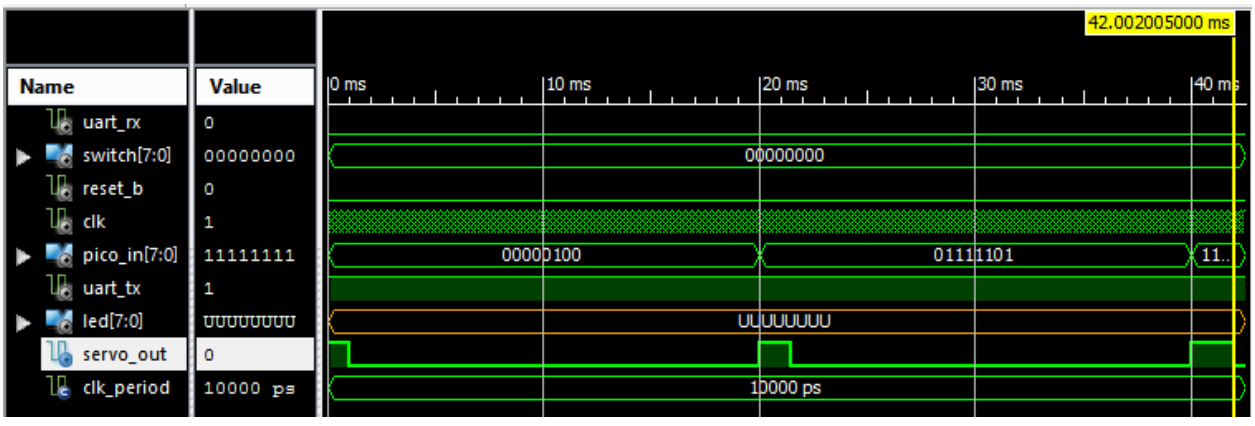
Als de testbench gesimuleerd wordt, worden de volgende afbeelding gegeneerd.



De eerste input “00000100” genereerd een puls van 1,07 ms in de cycle van 20 ms.



De tweede input “01111101” genereerd een puls van 1.5 ms in de cycle van 20 ms.



De derde input “11111111” genereerd een puls van 2 ms in de cycle van 20 ms.

Labo 4

Nu is het de bedoeling dat een motor vanuit assembler aangestuurd kan worden. Om dit het best te realiseren, wordt de VHDL code uit het vorige project samengevoegd met de VHDL uit het voorbeeldproject. Ook wordt de assembler code aangepast.

Er wordt gezorgd dat de servo veranderd wordt van positie wanneer er een knop ingedrukt wordt. In ons voorbeeld begint de servo neutraal. Wanneer er op ‘z’ gedrukt wordt, gaat de servo sneller naar rechts. Wanneer er op ‘s’ gedrukt wordt, gaat de servo sneller naar links. Wanneer er op ‘p’ gedrukt wordt is er een pauze en wordt de wheelpstn neutraal gezet.

Elke keer als er een actie plaatsvindt wordt er een output instructie uitgevoerd naar servo(0x03). Die in de VHDL dan wordt opgevangen.

Assembly code

testprgwheel.psm

VHDL code

Er worden ook enkele aanpassingen in VHDL gedaan zodat de motor vanuit assembler aangestuurd kan worden.

Uart6_atlys.UCF

In het UCF bestand worden de poorten van de hardware beschreven. Via pin ‘T3’ worden de pulsen doorgestuurd naar de servo. De naam van de pin is in dit geval ‘servo_out’.

```
NET "servo_out" LOC = "T3" | IOSTANDARD = LVCMOS33; # Bank = 2, Pin name = IO_L62N_D6, Sch name = JA - D0_N
```

Uart6_atlys.VHD

Nu dat het mogelijk is dat er data doorgestuurd kan worden via een pin. Moet het ook mogelijk zijn om data te verkrijgen vanuit assembler. Daarvoor worden er enkele aanpassingen gedaan in het bestand uart6_atlys.VHD.

Het bestaande proces ‘output_ports’ wordt aangepast zodat de info voor de servo binnenkomt op adres 0x03.

```
output_ports : PROCESS(clk)
BEGIN
	IF clk'EVENT AND clk = '1' THEN
		-- 'write_strobe' is used to qualify all writes to general output ports.
		IF write_strobe = '1' THEN
			-- Write to LEDs at port address 02 hex
			IF port_id = "000000010" THEN
				led(6 DOWNTO 0) <= out_port(6 DOWNTO 0);
			END IF;
			-- Write to servo_1 at port address 03 hex
			IF port_id = "00000011" THEN
				pico_in <= out_port;
			END IF;
		END IF;
	END IF;
END PROCESS output_ports;
```

Het is belangrijk dat het volgende stuk code ook gewijzigd wordt. Als dit niet gewijzigd wordt dan zal de tx op niet relevante delen verschijnen.

```
write_to_uart_tx <= '1' WHEN (write_strobe = '1') AND (port_id = "00000001")
                    ELSE '0';
```

In het project wordt er een extra poort ‘servo_out’ toegevoegd. Deze poort staat verbonden met de pin die in het ucf bestand staat.

```
ENTITY uart6_atlys IS
	PORT (
		uart_rx : IN std_logic;
```

```
                uart_tx : OUT std_logic;
                led : OUT std_logic_vector(7 DOWNTO 0);
                switch : IN std_logic_vector(7 DOWNTO 0);
                -- button : in std_logic_vector(4 downto 0);
                reset_b : IN std_logic;
                clk : IN std_logic;
                servo_out : OUT std_logic;
        );
END uart6_atlys;
```

Extra signalen die gebruikt worden voor de servo.

```
SIGNAL time_servo : std_logic_vector(7 DOWNTO 0);
SIGNAL outtimer_20ms : NATURAL RANGE 0 TO 20000; --Houdt de cycle bij.
SIGNAL pico_in : std_logic_vector(7 DOWNTO 0);
```

De signalen worden verbonden met de modules. De output wordt rechtstreeks naar de pin gestuurd.

```
timer_ms : timer
PORT MAP(clk => clk, outtimer_20ms => outtimer_20ms );
servo_data2 : servo
PORT MAP(clk => clk, outtimer_20ms => outtimer_20ms, pico_in => pico_in, servo_out => servo_out);
```

Labo 5

Dit is de finale labo. In dit labo is het de bedoeling dat het project werkt met twee servo's. De servo's samen vormen een kleine auto. Vanuit het assembler programma moeten er twee verschillende signalen gestuurd worden naar de servo's. Hierbij is het de bedoeling dat er data ontvangen wordt van een microcontroller. Het protocol van de data gaat als volgt "ACC___ x y z\r\n". 'x', 'y' en 'z' zijn waardes die van -255 tot 255 kunnen gaan. Deze waardes zijn gebaseerd op de accelerometer van de gamecontroller.

Assembly

Het programma in het project bestaat uit twee grote delen. Het eerste deel leest de waardes uit die de PicoBlaze binnenkrijgt via de UART_RX. Deze code is gebaseerd op labo 2. Er zijn veel aanpassingen geweest in die code. De grootste verschillen met labo 2, is dat de waardes van -255 tot 255 kunnen variëren. Voor elke waarde worden er twee registers gebruikt. Elke waarde heeft een register voor een min en een register voor het getal zelf.

Het andere deel van het programma bestaat uit het herrekenen van de waardes die verkregen werden uit de gamecontroller. Hoe hoger de 'x' is bij een positieve waarde, hoe sneller de auto vooruit moet rijden. Hoe hoger de 'x' is bij een negatieve waarde, hoe sneller de auto achteruit moet rijden. Ook wordt er gekeken naar de 'y' die bepaalt of de auto naar links of naar rechts moet rijden.

In het project worden de wielposities berekend door een simpele formule toe te passen. Voor een positieve x waarde wordt de formule X/4 + 127 +- Y/8 gebruikt. Als er een negatieve x is, wordt de formule 127-X/4 +- Y/8 gebruikt. Hierdoor kunnen beide wielen in theorie samen een maximale puls van 1.70ms -180ms hebben.

Cold_start

Cold_start gebeurt enkel wanneer het programma start. Cold_start initialiseert de wielen op een neutrale puls.

Cmdcar/stop

Omdat de x waarde van de accelerometer moeilijk exact een nul kan zijn, wordt ervoor gezorgd dat wanneer de waarde tussen -30 en 30 ligt, de motoren een neutrale puls krijgen.

Move/ backward/ forward/ calculate_directionF/ calculate_directionB

Wanneer x een andere waarde dan 30 heeft, wordt er eerst gekeken of x een negatief getal is. Als x een negatief getal is, wordt er naar de subroutine 'backward' gegaan. Als x positief is, wordt er naar forward gesprongen.

In de 'backward' en 'forward' subroutine wordt x gedeeld door vier. In 'backward' wordt het resultaat van 127 afgetrokken en wordt daarna in beide wielposities geladen. In 'forward' wordt het resultaat opgeteld en wordt daarna in beide wielposities geladen.

Afhankelijk of y positief is, wordt er bij het linker wiel of het rechter wiel de waarde y/8 afgetrokken of bijgeteld.

Send_data

Nadat de waardes berekend zijn voor beide wielen, worden de nieuwe waardes naar adres 0x03 en 0x04 gestuurd. In VHDL wordt de puls berekend op basis van de verzonden 8-bit data.

CmddataX/CmddataXneg/datacheckX

Tegenover labo 2 is er in deze procedure veel veranderd. Deze procedure leest de x, y en z in van het commando '$ACC___ X Y Z'. Er wordt altijd gewacht op een input. In deze subroutines wordt er eerst gecontroleerd of het getal negatief is. Als het getal negatief is, wordt het register 'sD' op 1 gezet. sD is het register dat aangeeft of x negatief is. Als volgt wordt er gecontroleerd of de volgende input een spatie of een getal tussen 0 en 10 is. Als het een spatie is, wordt er doorgesprongen naar het volgende getal. Wanneer het een cijfer tussen 0 en 10 is, wordt er een counter verhoogd en wordt het getal in een bufferregister gestoken.

De data kan niet rechtstreeks van de UART ingelezen worden. Omdat alle karakters één per één binnenkomen.

dataXinrgstr/calcData

Per cmddata wordt er op basis van de bufferregisters berekend, wat het totale getal is. Dat getal komt dan in het bufferregister s4. In sC wordt de counter bijgehouden hoeveel cijfers een getal bevat. In s0 wordt het eerste cijfer bijgehouden. In s1 wordt het tweede cijfer bijgehouden. In s3 wordt het derde cijfer bijgehouden. Op basis van de counter springt het programma naar de juiste subroutine 'onenmbr', 'twonmbr' of 'threenmbr'.

onenmbr/twonmbr/threenmbr/lp10th/lp100th

In deze subroutine wordt het uiteindelijk getal berekend.

Code


testprg.psm

VHDL

Ook in VHDL worden er enkele aanpassingen gedaan. Het meeste blijft hetzelfde. Het project maakt nu gebruik van twee motoren.

Uart6_atlys.UCF

Er wordt een pin toegevoegd voor de extra motor. De extra pin heeft als adres ‘R3’.

```
NET "servo_out_1" LOC = "T3" | IOSTANDARD = LVCMOS33; # Bank = 2, Pin name = IO_L62N_D6, Sch name
= JA - D0_N
NET "servo_out_2" LOC = "R3" | IOSTANDARD = LVCMOS33; # Bank = 2, Pin name = IO_L62N_D6, Sch name
= JA - D0_N
```

Uart6_atlys.VHD

Er wordt een extra poort toegevoegd zodat er signalen naar de tweede servo gestuurd kunnen worden.

```
NTITY uart6_atlys IS
        PORT (
                uart_rx : IN std_logic;
                uart_tx : OUT std_logic;
                led : OUT std_logic_vector(7 DOWNTO 0);
                switch : IN std_logic_vector(7 DOWNTO 0);
                -- button : in std_logic_vector(4 downto 0);
                reset_b : IN std_logic;
                clk : IN std_logic;
                servo_out_1 : OUT std_logic;
                servo_out_2 : OUT std_logic
        );
END uart6_atlys;
```

Er wordt een extra signaal toegevoegd voor de tweede servo. Dit signaal wordt gebruikt om de input op te slagen.

```
SIGNAL time_servo : std_logic_vector(7 DOWNTO 0);
SIGNAL outtimer_20ms : NATURAL RANGE 0 TO 20000; --Houdt de cycle bij.
SIGNAL pico_in_1 : std_logic_vector(7 DOWNTO 0);
SIGNAL pico_in_2 : std_logic_vector(7 DOWNTO 0);
```

Voor de nieuwe motor moet er ook een extra adres zijn zodat de VHDL de twee signalen uit elkaar kan houden. Voor het nieuwe signaal wordt het adres 0x04 gebruikt.

```
output_ports : PROCESS(clk)
BEGIN
        IF clk'EVENT AND clk = '1' THEN
                -- 'write_strobe' is used to qualify all writes to general output ports.
                IF write_strobe = '1' THEN
                        -- Write to LEDs at port address 02 hex
                        IF port_id = "000000010" THEN
                                led(6 DOWNTO 0) <= out_port(6 DOWNTO 0);
                        END IF;
                        -- Write to servo_1 at port address 03 hex
                        IF port_id = "00000011" THEN
                                pico_in_1 <= out_port;
                        END IF;
                        -- Write to servo_2 at port address 04 hex
                        IF port_id = "00000100" THEN
                                pico_in_2 <= out_port;
                        END IF;
                END IF;
        END IF;
END PROCESS output_ports;
```

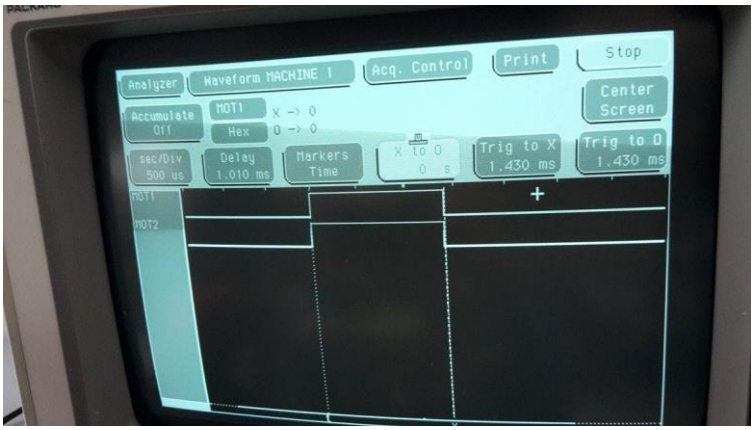
Beide servo's moeten een aparte module hebben voor het berekenen van de PWM's.

```
timer_ms : timer
PORT MAP(clk => clk, outtimer_20ms => outtimer_20ms );
servo_data2 : servo
PORT MAP(clk => clk, outtimer_20ms => outtimer_20ms, pico_in => pico_in_2, servo_out =>
servo_out_2);
servo_data1 : servo
PORT MAP(clk => clk, outtimer_20ms => outtimer_20ms, pico_in => pico_in_1, servo_out =>
servo_out_1);
```

Oscilloscoop lezing

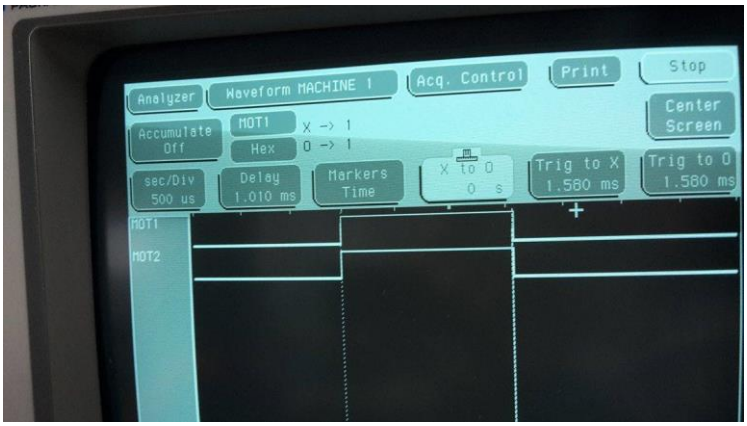
Doordat één van de motoren bij de auto niet werkt, is er gebruik gemaakt geweest van een oscilloscoop. Enkele metingen die genomen zijn geweest:

- $ACC___ -80 0 0



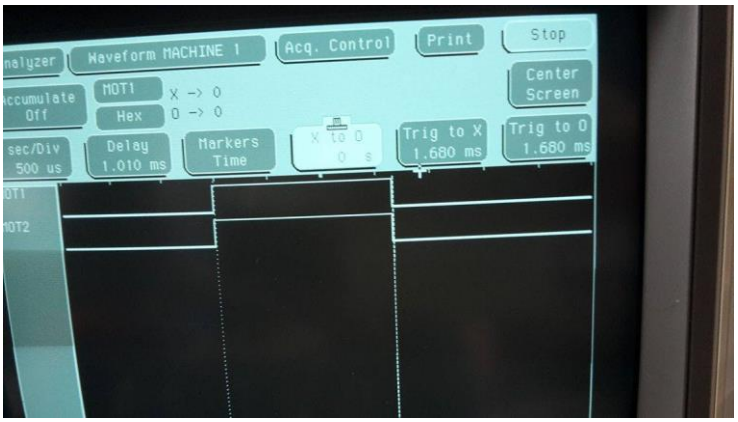
Beide servo's krijgen ongeveer een pulse van 1.430ms. De puls is kleiner dan 1.5ms dus de auto rijdt naar achter.

- $ACC___ 80 0 0



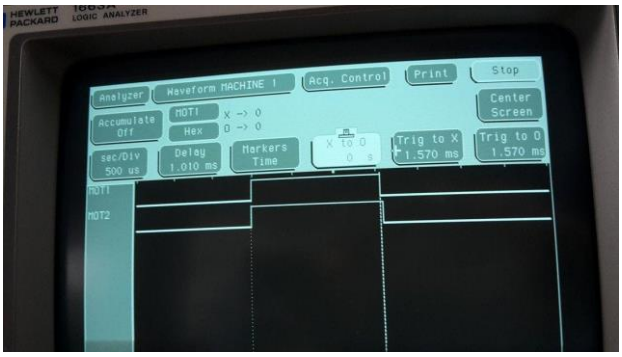
Beide servo's krijgen ongeveer een pulse van 1.570ms. De puls is groter dan 1.5ms dus de auto rijdt naar voor.

- $ACC___ 255 0 0



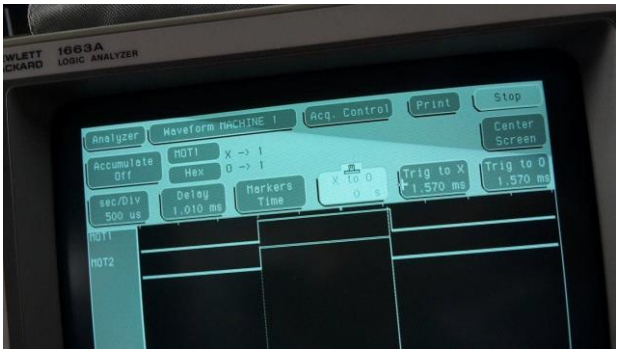
Beide servo's krijgen ongeveer een pulse van 1.680ms.

- $ACC___ 80 30 0



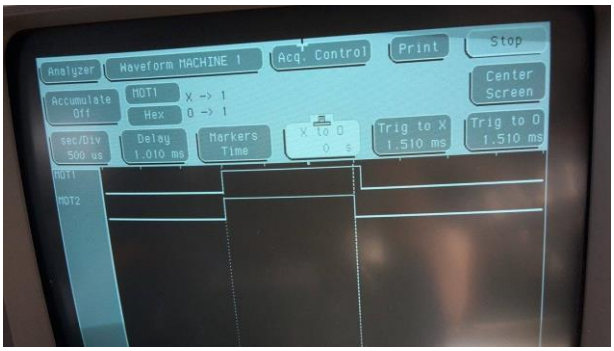
Servo1 krijgt een pulse van 1.570ms. Servo2 zijn puls is een beetje groter dan de pulse van servo1. De auto draait snel naar links.

- $ACC___ 80 -30 0



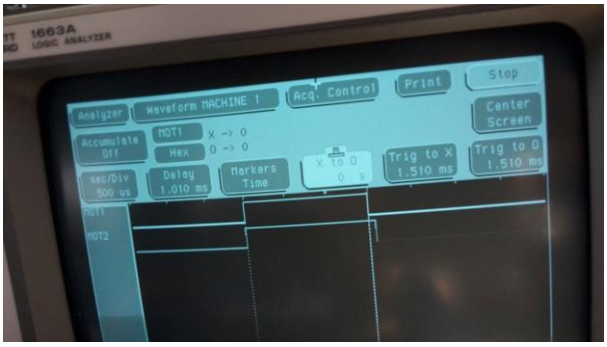
Servo2 krijgt een pulse van 1.570ms. Servo1 zijn puls is een beetje groter dan de pulse van servo2. De auto draait traag naar rechts.

- $ACC___ 80 -255 0



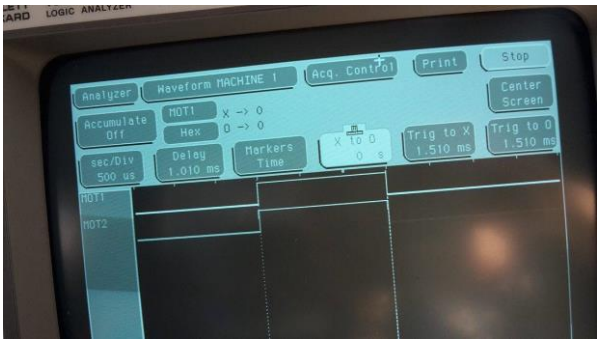
Servo2 krijgt een pulse van 1.510ms. Servo1 zijn puls is veel groter dan de pulse van servo2. De auto draait snel naar rechts.

- $ACC___ 80 255 0



Servo1 krijgt een pulse van 1.510ms. Servo2 zijn puls is veel groter dan de pulse van servo1. De auto draait snel naar links.

- **$ACC___ 20 0 0**



Beide servo's krijgen pulsen van 1.50ms. De motoren staan in rust.

4-Computerarchitectuur

Simon Sleutel
Eric Paillet
Koen Lostrie