Opdrachten practicum

DIGSE2

DE HAAGSE HOGESCHOOL

J.E.J op den Brouw
De Haagse Hogeschool
Opleiding Elektrotechniek
2 mei 2018
J.E.J.opdenBrouw@hhs.nl

Inleiding

Het practicum is zodanig van opzet en moeilijkheidsgraad dat iedere student die voor aanvang van het practicum zijn/haar opdrachten goed voorbereidt en altijd op het practicum aanwezig is een voldoende kan halen.

Het is de bedoeling dat je op dit practicum digitale systemen leert beschrijven, een competentie die onmisbaar is voor elke elektrotechnische ingenieur. Je kunt dit alleen leren door zelfstandig alle opdrachten uit te voeren. Het kopiëren van code of codedelen van het internet of van collega-studenten is niet leerzaam! Het boek biedt voldoende voorbeelden die je ter inspiratie kunt gebruiken. Gebruik nooit code die je zelf niet begrijpt in je beschrijvingen.

Als je niet weet hoe je moet beginnen of als je een bepaalde fout niet kunt vinden of als je niet weet hoe je een bepaalde actie in VHDL beschrijft of . . . vraag het dan aan de docent! Van vragen kun je veel leren. Je kunt je vraag ook altijd mailen naar J.E.J.opdenBrouw@hhs.nl.

Natuurlijk kun je ook dingen vragen aan medestudenten. Als jou iets wordt gevraagd geef je medestudent dan niet simpel een oplossing voor zijn/haar probleem maar help hem/haar om zelf het probleem op te lossen!

Thuis voorbereiden

De practicumopdrachten kunnen thuis uitgewerkt worden. Van Quartus is een zogenaamde Web Edition-versie[1] beschikbaar. Deze heeft geen licentie nodig. Er zijn versies voor Windows en Linux. Er is geen versie voor OS-X. Voor alle versies van Windows en Linux wordt aangeraden om Quartus versie 13.0sp1 te installeren. Lagere versies geven driver-problemen bij Windows 8. De projecten zijn uitwisselbaar met de op school geïnstalleerde versie 11.1sp1.

De software is te downloaden via http://dl.altera.com/13.0sp1/?edition=web#tabs-1. Je moet wel een account aanmaken; dat is gratis. Let erop dat de volledig geïnstalleerde versies bij elkaar zo'n 11 GB aan harddiskruimte in beslag nemen. Vergeet niet dat de download zelf zo'n 4,5 GB in beslag neemt. Dat geldt ook voor het uitpakken, dat is ook 4,5 GB groot.

Op het practicum wordt gebruik gemaakt van het DE0-bordje[2]. In een later project wordt ook gebruik gemaakt van het DE2-70-bordje. De laatste versie die beide borden ondersteunt is 13.0sp1, hogere versies kunnen niet gebruikt worden.

[1] Vanaf versie 13.0 is ModelSim geïntegreerd in het installiepakket van Quartus
[2] Zie http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=364

Huisregels

Tijdens het gebruik van de practicumruimte en -apparatuur gelden de volgende huisregels:

- Niet eten en/of drinken in het laboratorium.
- De apparatuur wordt alleen gebruikt voor practicum-doeleinden.
- Zet de apparatuur in de juiste staat terug.
- Praat rustig, niet schreeuwen.
- Telefoons graag op stil zetten.
- Altijd de aanwijzingen van het personeel opvolgen.

Practicumregels

Voor het praticum geldt een aantal regels:

1. Aanwezigheid tijdens het practicum is verplicht.
2. Bij ziekte e.d. zo spoedig mogelijk (liefst voorafgaand aan het practicum) contact opnemen met de docent (liefst per mail) om inhaalmogelijkheden te bespreken.
3. Een practicumopdracht moet uiterlijk één week later worden afgerond dan de week waarin de opdracht moet worden uitgevoerd.
4. Te laat ingeleverde opdrachten zijn automatisch onvoldoende en kunnen niet worden ingehaald!
5. De student moet tijdens de practicumuren aan de opgegeven practicumopdrachten werken.
6. Een ingeleverde opdracht die niet met een voldoende wordt beoordeeld kan (een week later) worden aangevuld. Als je code niet helemaal perfect blijkt te zijn is dat dus geen probleem. Als je pas begint met beschrijven is het normaal dat alles niet meteen perfect is. Natuurlijk moet je wel proberen om je code zo goed te maken als je zelf kunt.
7. Het laten nakijken van VHDL-code die je niet zelf bedacht en beschreven hebt, wordt beschouwd als fraude (net zoals het spieken bij tentamens). Als je code die je niet zelf hebt gemaakt probeert in te leveren krijg je meteen een onvoldoende voor het gehele practicum. De fraude wordt ook gemeld bij de examencommissie. Fraude kan leiden tot een schorsing.
8. De deadline is week 7 van het blok, de student krijgt dan zijn eindbeoordeling: voldoende of onvoldoende.
9. Een onvoldoende als eindresultaat kan worden herkanst in week 10. De opdrachten die nog niet met een voldoende zijn beoordeeld, dienen binnen 15 minuten te worden gedemonstreerd en beoordeeld.

Practicumkaart

Je ontvangt tijdens de eerste practicumbijeenkomst een practicumkaart. Op deze kaart worden aanwezigheid en opdrachten die goedgekeurd zijn afgetekend. Je moet je kaart voor elke bijeenkomst meenemen.

BlackBoard

Om de practicumopdrachten te kunnen maken, moet je aangemeld zijn bij BlackBoard[3] van De Haagse Hogeschool. Je moet inloggen met de gegevens die je van de IT-dienst hebt gehad.

Op BlackBoard worden ook mededelingen gepost. Hou BlackBoard dus altijd in de gaten.

Website

Alle practicumopdrachten en bestanden die je nodig hebt kan je ook vinden op de website http://ds.opdenbrouw.nl.

Algemene leerdoelen

De algemene leerdoelen zijn:

- Leren omgaan met de pakketten Quartus en ModelSim.
- Bediening DE0-experimenteerbord.

Deze leerdoelen gelden voor elke opdracht.

Afrondmoment opdrachten

Een practicumopdracht moet uiterlijk één week later worden afgerond dan de week waarin de opdracht moet worden uitgevoerd. De uitvoerweek staat vermeld bij de opdracht.

Resultaat practicum

Het practicum wordt met een voldoende beoordeeld als:

- de student alle practicumsessies aanwezig is geweest.
- alle opdracht correct zijn afgerond.

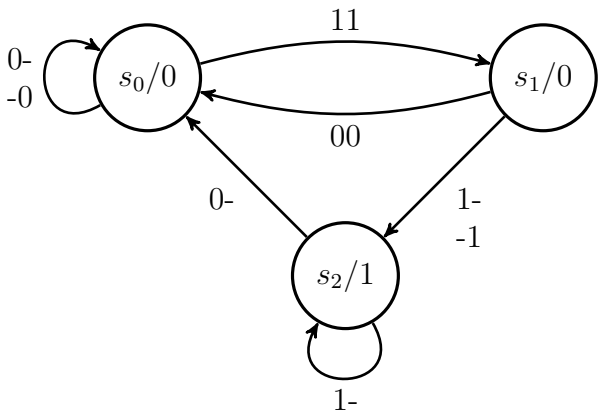
[3] Zie http://blackboard.hhs.nl/

Opdracht week 1 – Moore-machine

Inleiding

De opdracht voor deze week is het coderen van de Moore-machine in Quartus. Het toestandsdiagram van de machine is gegeven in figuur 1.

De machine heeft twee ingangen `x` en `y` en een uitgang `z`. Voor de ingangen kunnen twee schuifschakelaars worden gebruikt, voor de uitgang een led. Omdat het DE0-bordje een zeer snelle klok heeft, is het verstandig om het kloksignaal zelf toe te dienen door middel van één van de BUTTON's. Noot: het coderen van een toestandsmachine is nog niet ter sprake geweest. Het kan daarom wat lastig zijn. In de slides staan voorbeelden hoe het coderen in zijn werk gaat. Het testen van de machine is een lastige zaak. Je weet nooit precies in welke toestand de machine zich bevindt (als de machine goed werkt, en je weet wat de ingangswaarden over verloop van tijd zijn geweest, dan weet je het natuurlijk wel). Aan de buitenkant kan je niet zien in welke toestand de machine zich bevindt. Dit kan wel door middel van simulatie. Stel voor de ingevoerde machine een simulatie op, compleet met VHDL-testbench en simulatiecommandobestand. Je kan uiteraard de Mealy-machine als voorbeeld gebruiken.



Figuur 1: *Toestandsdiagram van de Moore-machine.*

Leerdoelen

De leerdoelen van deze opdracht zijn:

- Gebruik van de *state machine editor* van Quartus.
- Beschrijven van een eenvoudige toestandsmachine in VHDL.
- Opzetten van een simulatie van een eenvoudige toestandsmachine in VHDL.

Opdrachten

De volgende stappen moeten worden doorlopen:

a) Codeer de Moore-machine in VHDL (of gebruik de state machine editor).

b) Simuleer de machine met behulp van een testbench en een simulatiecommando-bestand. Simuleer zo veel mogelijk situaties. De toestanden s_0, s_1 en s_2 moeten minstens twee keer doorlopen worden.

c) Implementeer het geheel op een DE0-bordje. Gebruik de schuifschakelaars als ingangen. Gebruik een drukknop als kloksignaal.

d) Laat de docent het geheel controleren.

Opmerkingen

Het is mogelijk om de toestandsmachine met de State Machine Editor in te voeren. Er is een tutorial samengesteld door Pieter van der Star. Alleen het eerste deel ("State Machine Editor") moet doorlopen worden. In Bijlage 1 en 3 van de tutorial zijn een VHDL-testbench en ModelSim Command File (.do file) te vinden. Beide bestanden zijn niet compleet.

Gebruik een drukknop (bv. BUTTON2) als kloksignaal.

Opdracht week 2 – Personenteller

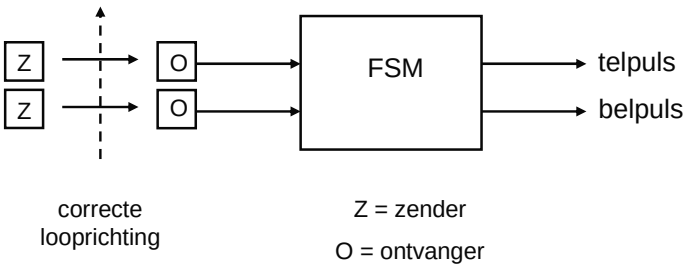
Inleiding

Bij attracties is het vaak noodzakelijk om het aantal personen te tellen, bijvoorbeeld het juiste aantal mensen dat in de wagentjes van de Python mogen plaatsnemen. Hiervoor zijn diverse onderdelen nodig:

- een richtingsdetector,
- een teller
- een zeven-segmenten decoder

Deze opdracht bestaat uit het ontwerpen, implementeren en beproeven van het telsysteem.

De richtingsdetector bij de Python moet detecteren of personen bij de ingang naar binnen gaan. Als dat zo is, moet een puls worden afgegeven waarmee de teller aangestuurd kan worden. Als een persoon via de ingang naar buiten gaat, moet een puls worden afgegeven zodat een bel kan gaan rinkelen. Het systeem heeft twee optoelektronische sensoren (ook wel genoemd lichtstraalonderbrekers) die een logische '1' afgeven als de straal is onderbroken. De sensoren staan zo geplaatst dat een persoon niet tussen de twee lichtstralen kan staan. Een persoon kan wel beide lichtstralen tegelijk onderbreken. De aansluitingen van de richtingsdetector wordt toegelicht in figuur 2.



Figuur 2: *Aansluitingen van de richtingsdetector*

De teller telt het aantal personen dat de goede richting is opgelopen. Het aantal te tellen personen is maximaal 9. De telstand wordt met behulp van een zeven-segmenten deccder afgebeeld op één zeven-segmenten display.

Leerdoelen

De leerdoelen van deze opdracht zijn:

- Ontwerpen van een blokschema van het complete digitale systeem.
- Ontwerpen van een toestandsmachine vanuit een geschreven specificatie.
- Coderen van de toestandsmachine in VHDL (eventueel met *state machine editor*).

- Opzetten van een simulatie van de toestandsmachine in VHDL.

Opdrachten

De volgende stappen moeten worden doorlopen:

a) Teken (op papier) een blokschema van het geheel met daarin alle modules én de signalen tussen de modules.

b) Ontwerp (op papier) het toestandsdiagram voor het richtingsdetector. Denk hierbij aan het volgende: een persoon kan beginnen met een loop in de correcte richting, maar bedenkt zich en loopt terug naar de ingang. Een persoon kan langdurig op de zelfde plaats blijven staan en de lichtstralen langdurig onderbreken. *Laat het toestandsdiagram controleren voordat je verder gaat.*

c) Codeer het geheel van a) en b) in VHDL. Maak eventueel gebruik van de *state machine editor*. Maak gebruik van structural VHDL.

d) Simuleer alleen de detector met behulp van een testbench en een simulatiecommandobestand. Simuleer zo veel mogelijk situaties, maar minimaal één keer een correcte en één keer een foutieve doorloop.

e) Implementeer het geheel op een DE0-bordje. Gebruik de schuifschakelaars als sensoren. Gebruik een drukknop (bv. BUTTON2) om een klokpuls toe te dienen.

Opmerkingen

De teller en zeven-segmenten decoder kunnen uit een eerder project gebruikt worden.

De richtingsdetector en de teller moeten op hetzelfde kloksignaal geklokt worden (kloksynchroon ontwerp).

De state machine editor slaat het ingevoerde toestandsdiagram op in een SMF-bestand. Vanuit dit bestand wordt een VHDL-bestand gegenereerd die in het project kan worden gebruikt. Wijzigingen in het toestandsdiagram moeten worden doorgevoerd in het SMF-bestand, dus niet het gegenereerde VHDL-bestand aanpassen.

Let op contactdender. Een probleem kan zijn dat bij het inklokken de ingangswaarden meerdere malen veranderen. Gebruik dubbele synchronisatie.

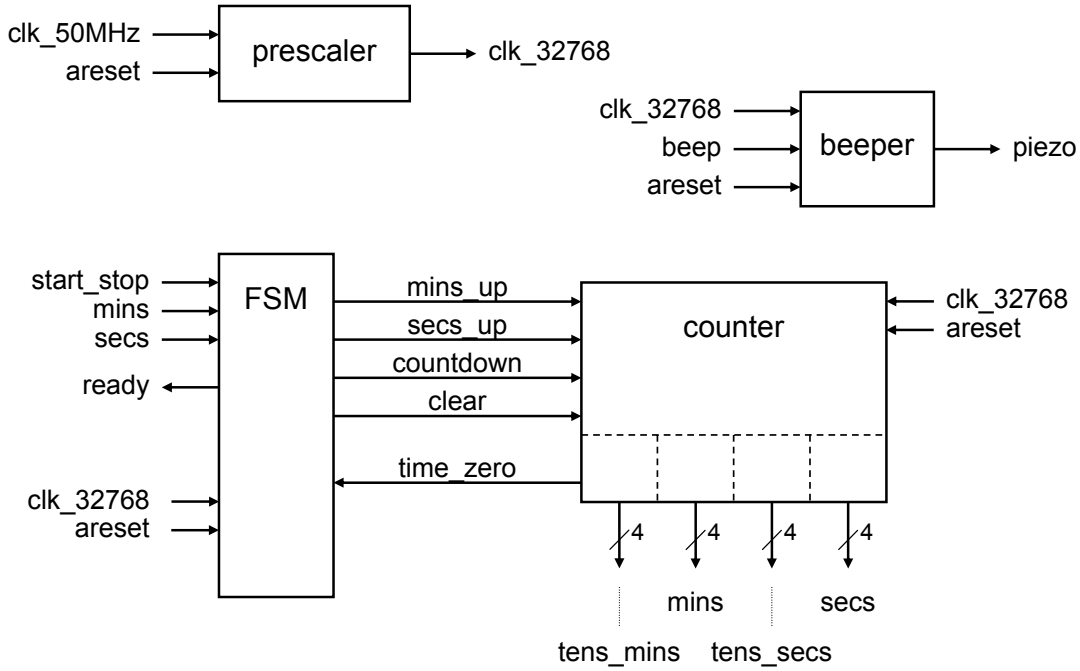
Opdracht week 3 en 4 – kookwekker

Inleiding

Het koken van een eitje lukt de meeste mensen nog. Toch zijn er wel mensen die dat niet zonder een kookwekker kunnen, met als gevolg een hard gekookt ei (of juiste te zacht). Elk huishouden heeft wel zo'n kookwekker in huis. Twee voorbeelden van zo'n typische kookwekker zijn te vinden in figuur 3.

Figuur 3: *Twee voorbeelden van een kookwekker*

Deze opdracht bestaat uit het ontwerpen, implementeren en beproeven van een kookwekker. In figuur 4 is het blokschema gegeven.



Figuur 4: *Blokschema van de kookwekker*

Het systeem bestaat uit een prescaler, een teller, een FSM die de teller bestuurt en een *beeper*. Niet getekend in het blokschema zijn de zeven-segment-decoders voor het aansturen van de displays. Dat moet wel gebeuren.

De prescaler wordt gebruikt om de systeemklok van 50 MHz terug te brengen naar een interne klok van 32768 Hz. Dit is de frequentie waarop de meeste kookwekkers en polshorloges draaien. Normaal gesproken is dit een kristal. Alle onderdelen werken op deze frequentie. De prescaler wordt gerealiseerd met behulp van een *Phase Locked Loop* (PLL), een stuk deels analoog, deels digitale schakeling die al op de FPGA is geplaatst. Meer informatie over de PLL is gegeven in paragraaf Code Phase Locked Loop. De code van de PLL is gegeven is listing 4.

De FSM zorgt er samen met de teller voor dat de tijd kan worden ingesteld en dat de ingestelde tijd wordt afgeteld. De teller geeft aan wanneer de tijd op nul staat. De ingestelde tijd loopt van 00:00 (0 minuten en 0 seconden) tot 99:59 (99 minuten en 59 seconden). Meer informatie over hoe de teller kan worden opgebouwd is te vinden inde paragraaf De teller. De beeper levert een pieptoon van ongeveer 2 kHz als het signaal `beep` geactiveerd wordt. De code van de beeper is gegeven in paragraaf Code Beeper en listing 5.

De besturing is als volgt. Met behulp van de ingangen `mins` en `secs` kan een bepaalde tijd ingesteld worden. Minuten en seconden kunnen apart worden ingesteld. Met behulp van de ingang `start/stop` wordt het aftellen van de kookwekker gestart en gestopt (gestopt als in pauze). Als de teller bij aftellen op 0 (00:00) komt, wordt het aftellen gestopt, wordt uitgang `ready` geactiveerd en moet de bekende pieptoon te horen zijn. Bij het gelijktijdig indrukken van `mins` en `secs` wordt de teller op 0 (00:00) gezet.

Leerdoelen

De leerdoelen van deze opdracht zijn:

- Ontwerpen van een complexe toestandsmachine bestaande uit een datapad (de teller) en een control (FSM) vanuit een geschreven specificatie.
- Beschrijven van de functionaliteit van een systeem.
- Coderen van het complete systeem in VHDL.
- Opzetten van een simulatie voor (een deel van) het systeem.

Opdrachten

De volgende stappen moeten worden doorlopen:

a) Beschrijf kort de functionaliteit van de kookwekker. Bedenk wat het systeem wel en niet moet kunnen.

b) Ontwerp (op papier) het toestandsdiagram voor de FSM van de kookwekker. Bedenk goed welke toestanden het systeem allemaal kan hebben. Laat je ontwerp controleren door de docent voordat je verder gaat.

c) Codeer het geheel in VHDL. Neem dit keer leuke namen voor de toestanden. Het gebruik van de State Machine Editor wordt *sterk afgeraden*.

d) Simuleer alleen de FSM van de kookwekker met behulp van een testbench en een simulatiecommandobestand. Simuleer zo veel mogelijk situaties.

e) Implementeer het geheel op een DE0-bordje. Gebruik de drukknoppen voor de minuten, seconden en start/stop. Synchroniseer de ingangen!

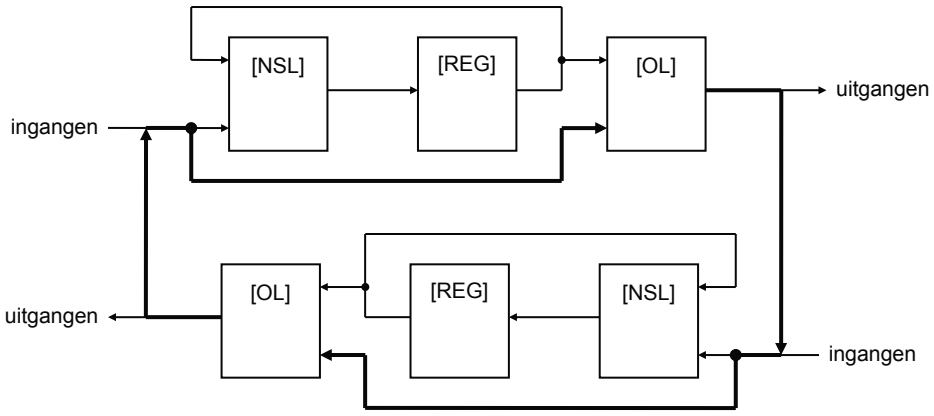
Opmerkingen

Ontwerp volgens de regels van de digitale techniek:

- De ingangen van het systeem (`start/stop`, `mins` en `secs`) moeten eerst gesynchroniseerd worden d.m.v. dubbele synchronisatie.
- Alle geheugenelementen zijn flankgevoelig (flipflops). Er mogen geen latches gebruikt worden.
- Alle flipflops moeten d.m.v. een asynchrone reset in een bekende toestand gedwongen kunnen worden.
- Alle flipflops worden gevoed door hetzelfde kloksignaal, de interne klok van 32768 Hz. Geen andere prescalers/kloksignalen gebruiken!
- Alle interne en externe signalen moeten synchroon verwerkt worden, met uitzondering van de asynchrone reset. Zo mag de teller alleen synchroon op 0 gezet worden via een interne `clear`-ingang.

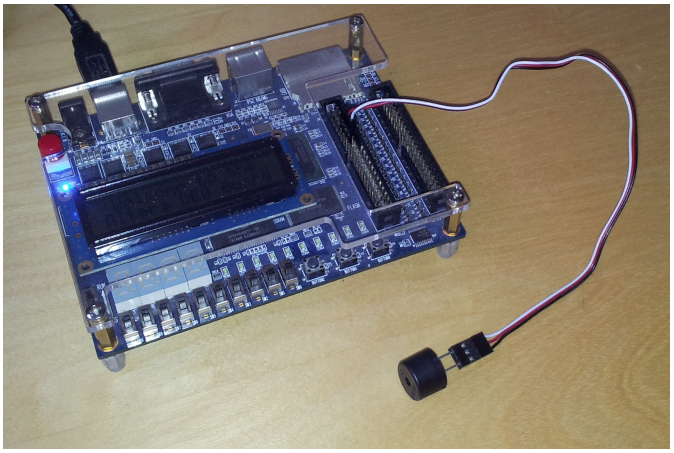
Verder zijn er nog wat punten om mee te nemen in het ontwerp:

- Gebruik drukknoppen voor de ingangen. De drukknoppen zijn actief laag.
- Bij deze opdracht is het de bedoeling dat het systeem d.m.v. gescheiden datapad en control wordt gerealiseerd.
- Gebruik dezelfde namen voor de signalen zoals gegeven in figuur 4.
- Let op dat de FSM en de teller twee *rondgekoppelde* toestandsmachines zijn die last kunnen hebben van asynchrone lusvorming. Zie figuur 5.

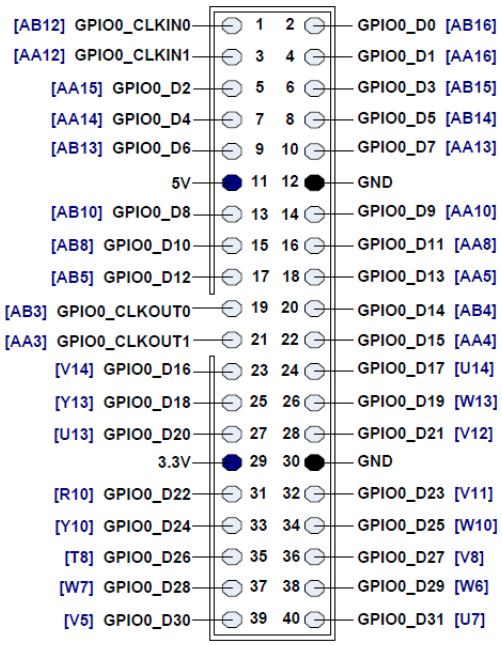


Figuur 5: *Rondgekoppelde Mealy-machines met mogelijk een combinatorische lus.*

Tussen de uitgang `piezo` en de ground wordt een piezo-buzzer geplaatst zodat een pieptoon van 2 kHz te horen is. Zie figuur 6. De uitgang `piezo` moet gekoppeld worden aan pin AB14 (pin 8 op de connector, GPIO0_D5) en de *ground* (pin 12 op de connector, GND). Zie figuur 7.



Figuur 6: *Aansluiting van de piezo-buzzer op een DE0-bordje.*



Figuur 7: *Pinaansluitingen van de GPIO0*

De teller

De teller bestaat uit vier secties te weten: seconden, tientallen seconden, minuten en tientallen minuten. Natuurlijk kan deze teller beschreven worden met gecascadeerde tellers (volgens het terminal count/enable principe), maar met VHDL is een veel krachtigere methode mogelijk: een teller binnen een teller. We doen dat aan de hand van een twee-cijferige BCD-teller.

Laten we eerst eens kijken naar een teller voor één BCD-cijfer. Dit is te zien in listing 1. De telstand wordt op 0000_{BCD} gezet als de teller op 1001_{BCD} staat. Zo niet, dan wordt

```
if rising_edge(clk) then
    if count = "1001" then
        count <= "0000";
        -- *
    else
        count <= count + 1;
    end if;
end if; -- edge
```

Listing 1: *Een 1-digit BCD-teller*

de telstand met 1 verhoogd. We zouden kunnen aangeven dat de teller in de hoogste stand staat op het punt gemarkeerd met een *, bijvoorbeeld met een terminal count.

Het is echter ook mogelijk om op gemarkeerd met een * de beschrijving van een tweede teller te plaatsen.

In listing 2 is een voorbeeld te zien van een twee-cijferige BCD-teller. Deze telt van $0000\ 0000_{\text{BCD}}$ tot en met $1001\ 1001_{\text{BCD}}$. Het principe is eenvoudig. Het bestaat uit twee BCD-tellers, één voor de eenheden en één voor de tientallen. Het zijn in feite identieke tellers die in elkaar verweven zitten.

Er wordt eerst getest of de eenheden op telstand 1001_{BCD} staan (uiteraard maakt het niet uit of we binair of BCD testen, het bitpatroon is identiek). Zo ja, zetten we de eenheden op 0000_{BCD} en starten we de test voor de tientallen. Staan de tientallen op 1001_{BCD}, dan zetten we die weer op 0000_{BCD}. De tellers worden met één verhoogd als de telstand niet op 1001_{BCD} staat.

Honderdtallen kunnen eenvoudig worden gerealiseerd door de beschrijving van een derde teller op het punt gemarkeerd ** toe te voegen.

```
-- Start counter for units (ones)
if countones = "1001" then
    countones <= "0000";
    -- Start counter for tens
    if counttens = "1001" then
        counttens <= "0000";
        -- ** (no hunderds...)
    else
        counttens <= counttens + 1;
    end if;
    -- End counter for tens
else
    countones <= countones + 1;
end if;
-- End counter for units
```

Listing 2: *Voorbeeld van een 2-digit BCD-teller*

De ontwikkelde code van de teller kan getest worden met een simulatie-project dat op BlackBoard staat. Als alternatief kan je het project downloaden van http://ds.opdenbrouw.nl/digse2/digse2_kookwekker_counter_sim.zip.

Code Phase Locked Loop

Phase Locked Loops zijn analoge/digitale componenten die gebruikt worden voor het genereren of terugwinnen van kloksignalen. Met deze component is het mogelijk om de frequentie van een referentieklok te verlagen of te verhogen. Bij verlagen wordt over *clock divider* gesproken, bij verhogen over *clock multiplier*. Het gaat te ver om gedetailleerd de werking van een PLL te bespreken, zie http://www.silabs.com/Support%20Documents/TechnicalDocs/AN575.pdf voor een korte introductie en zie http://www.cppsim.com/PLL_Lectures/digital_pll_cicc_tutorial_perrott.pdf voor een detailuitleg.

In listing 3 is een opzet voor instantiëring van de PLL gegeven. Deze PLL dient als vervanging van de eerder gebruikte prescaler, vandaar dat de entity-naam nog die term bevat. Het signaal `areset` is de asynchrone reset, actief hoog. Signaal `inclk0` is de ingaande referentieklok. Op het DE0-bordje is dat de 50 MHz oscillator op pin G21. Het signaal `c0` is de uitgaande klok voor intern gebruik. De frequentie is 32768 Hz. Het signaal `locked` geeft aan of het interne kloksignaal beschikbaar is. Het wordt in het ontwerp verder niet gebruikt.

```
u0: prescaler32768
    port map (areset => areset, inclk0 => clock_50MHz,
              c0 => clock_32768, locked => open);
```

Listing 3: *Opzet instantiëring en port map PLL*

In listing 4 is de code gegeven voor de PLL. Deze code is gegenereerd m.b.v. Quartus' Mega Function. Merk op dat in deze code de `altera_mf`-bibliotheek gebruikt wordt. Hierin is de beschrijving van de PLL opgeslagen.

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

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY prescaler32768 IS
    PORT
    (
        areset      : IN STD_LOGIC  := '0';
        inclk0      : IN STD_LOGIC  := '0';
        c0      : OUT STD_LOGIC ;
        locked      : OUT STD_LOGIC
    );
END prescaler32768;

ARCHITECTURE SYN OF prescaler32768 IS

    SIGNAL sub_wire0    : STD_LOGIC ;
    SIGNAL sub_wire1    : STD_LOGIC_VECTOR (4 DOWNTO 0);
    SIGNAL sub_wire2    : STD_LOGIC ;
    SIGNAL sub_wire3    : STD_LOGIC ;
    SIGNAL sub_wire4    : STD_LOGIC_VECTOR (1 DOWNTO 0);
    SIGNAL sub_wire5_bv : BIT_VECTOR (0 DOWNTO 0);
    SIGNAL sub_wire5    : STD_LOGIC_VECTOR (0 DOWNTO 0);
```

```
    COMPONENT altpll
    GENERIC (
        bandwidth_type      : STRING;
        clk0_divide_by      : NATURAL;
        clk0_duty_cycle     : NATURAL;
        clk0_multiply_by        : NATURAL;
        clk0_phase_shift        : STRING;
        compensate_clock        : STRING;
        inclk0_input_frequency      : NATURAL;
        intended_device_family      : STRING;
        lpm_hint        : STRING;
        lpm_type        : STRING;
        operation_mode      : STRING;
        pll_type        : STRING;
        port_activeclock        : STRING;
        port_areset     : STRING;
        port_clkbad0        : STRING;
        port_clkbad1        : STRING;
        port_clkloss        : STRING;
        port_clkswitch      : STRING;
        port_configupdate       : STRING;
        port_fbin       : STRING;
        port_inclk0     : STRING;
        port_inclk1     : STRING;
        port_locked     : STRING;
        port_pfdena     : STRING;
        port_phasecounterselect     : STRING;
        port_phasedone      : STRING;
        port_phasestep      : STRING;
        port_phaseupdown        : STRING;
        port_pllena     : STRING;
        port_scanaclr       : STRING;
        port_scanclk        : STRING;
        port_scanclkena     : STRING;
        port_scandata       : STRING;
        port_scandataout        : STRING;
        port_scandone       : STRING;
        port_scanread       : STRING;
        port_scanwrite      : STRING;
        port_clk0       : STRING;
        port_clk1       : STRING;
        port_clk2       : STRING;
        port_clk3       : STRING;
        port_clk4       : STRING;
        port_clk5       : STRING;
        port_clkena0        : STRING;
        port_clkena1        : STRING;
        port_clkena2        : STRING;
        port_clkena3        : STRING;
        port_clkena4        : STRING;
        port_clkena5        : STRING;
        port_extclk0        : STRING;
        port_extclk1        : STRING;
        port_extclk2        : STRING;
        port_extclk3        : STRING;
```

```
        self_reset_on_loss_lock     : STRING;
        width_clock     : NATURAL
    );
    PORT (
            areset  : IN STD_LOGIC ;
            clk : OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
            inclk   : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            locked  : OUT STD_LOGIC
    );
    END COMPONENT;

BEGIN
    sub_wire5_bv(0 DOWNTO 0) <= "0";
    sub_wire5    <= To_stdlogicvector(sub_wire5_bv);
    locked    <= sub_wire0;
    sub_wire2    <= sub_wire1(0);
    c0    <= sub_wire2;
    sub_wire3    <= inclk0;
    sub_wire4    <= sub_wire5(0 DOWNTO 0) & sub_wire3;

    altpll_component : altpll
    GENERIC MAP (
        bandwidth_type => "AUTO",
        clk0_divide_by => 390625,
        clk0_duty_cycle => 50,
        clk0_multiply_by => 256,
        clk0_phase_shift => "0",
        compensate_clock => "CLK0",
        inclk0_input_frequency => 20000,
        intended_device_family => "Cyclone III",
        lpm_hint => "CBX_MODULE_PREFIX=prescaler32768",
        lpm_type => "altpll",
        operation_mode => "NORMAL",
        pll_type => "AUTO",
        port_activeclock => "PORT_UNUSED",
        port_areset => "PORT_USED",
        port_clkbad0 => "PORT_UNUSED",
        port_clkbad1 => "PORT_UNUSED",
        port_clkloss => "PORT_UNUSED",
        port_clkswitch => "PORT_UNUSED",
        port_configupdate => "PORT_UNUSED",
        port_fbin => "PORT_UNUSED",
        port_inclk0 => "PORT_USED",
        port_inclk1 => "PORT_UNUSED",
        port_locked => "PORT_USED",
        port_pfdena => "PORT_UNUSED",
        port_phasecounterselect => "PORT_UNUSED",
        port_phasedone => "PORT_UNUSED",
        port_phasestep => "PORT_UNUSED",
        port_phaseupdown => "PORT_UNUSED",
        port_pllena => "PORT_UNUSED",
        port_scanaclr => "PORT_UNUSED",
        port_scanclk => "PORT_UNUSED",
        port_scanclkena => "PORT_UNUSED",
        port_scandata => "PORT_UNUSED",
```

```
        port_scandataout => "PORT_UNUSED",
        port_scandone => "PORT_UNUSED",
        port_scanread => "PORT_UNUSED",
        port_scanwrite => "PORT_UNUSED",
        port_clk0 => "PORT_USED",
        port_clk1 => "PORT_UNUSED",
        port_clk2 => "PORT_UNUSED",
        port_clk3 => "PORT_UNUSED",
        port_clk4 => "PORT_UNUSED",
        port_clk5 => "PORT_UNUSED",
        port_clkena0 => "PORT_UNUSED",
        port_clkena1 => "PORT_UNUSED",
        port_clkena2 => "PORT_UNUSED",
        port_clkena3 => "PORT_UNUSED",
        port_clkena4 => "PORT_UNUSED",
        port_clkena5 => "PORT_UNUSED",
        port_extclk0 => "PORT_UNUSED",
        port_extclk1 => "PORT_UNUSED",
        port_extclk2 => "PORT_UNUSED",
        port_extclk3 => "PORT_UNUSED",
        self_reset_on_loss_lock => "OFF",
        width_clock => 5
    )
    PORT MAP (
        areset => areset,
        inclk => sub_wire4,
        locked => sub_wire0,
        clk => sub_wire1
    );

END SYN;
```

Listing 4: *VHDL code van de Phase Locked Loop*

Code Beeper

Onderstaande listing is de code voor de *beeper*. Merk op dat er gebruik gemaakt wordt van twee zogenaamde *generic constants* te weten `freq_sys` en `freq_beep`. Bij *instantiëring* van de beeper moeten resp. de waarden 32768 en 2048 opgegeven worden.

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

entity beeper is
    generic (freq_sys : positive := 50000000;
             freq_beep : positive := 2000);
    port    (clk    : in std_logic;
             areset : in std_logic;
             beep   : in std_logic;
             piezo  : out std_logic
            );
end entity beeper;

architecture rtl of beeper is
constant nr_pulses : integer := freq_sys/ freq_beep / 2;
constant nr_nr_pulses : integer := freq_beep / 8;
```

```
signal counter : integer range 0 to 15;
signal count_nr_pulse : integer range 0 to nr_pulses-1;
signal count_nr_nr_pulse : integer range 0 to nr_nr_pulses-1;
signal piezo_int : std_logic;
signal beep_s, beep_ss : std_logic;
begin
    process (clk, areset, beep) is
    begin
        if areset = '1' then
            counter <= 0;
            count_nr_pulse <= 0;
            count_nr_nr_pulse <= 0;
            piezo_int <= '0';
            beep_s <= '0';
            beep_ss <= '0';
        elsif rising_edge(clk) then
            beep_s <= beep;  -- synchronize
            beep_ss <= beep_s;
            if beep_ss = '1' then
                if count_nr_pulse = nr_pulses-1 then
                    count_nr_pulse <= 0;
                    if count_nr_nr_pulse = nr_nr_pulses-1 then
                        count_nr_nr_pulse <= 0;
                        if counter = 15 then
                            counter <= 0;
                        else
                            counter <= counter + 1;
                        end if;
                    else
                        count_nr_nr_pulse <= count_nr_nr_pulse + 1;
                    end if;
                    if counter < 10 then
                        if (counter mod 2) = 0 then
                            piezo_int <= not piezo_int;
                        else
                            piezo_int <= '0';
                        end if;
                    else
                        piezo_int <= '0';
                    end if;
                else
                    count_nr_pulse <= count_nr_pulse + 1;
                end if;
            else
                counter <= 0;
                count_nr_pulse <= 0;
                piezo_int <= '0';
                count_nr_nr_pulse <= 0;
            end if;
        end if;
    end process;

    piezo <= piezo_int;
end architecture rtl;
```

Listing 5: *VHDL code van de beeper*

Opdracht week 5 en 6 – Seriële communicatie

Inleiding

Communicatie tussen systemen is noodzakelijk om informatie uit te wisselen. Hierdoor is het mogelijk om informatie (data) op verschillende systemen te bewaren en te gebruiken. Informatie-overdracht kan in feite op twee manieren: parallelle data-overdracht en seriële data-overdracht. Bij parallelle data-overdracht voeren meerdere signaallijnen de data van het ene systeem naar het andere systeem. Dat is zeer geschikt voor kleine afstanden. Voor grotere afstanden levert dit problemen op: de parallelle bedrading levert een aanzienlijke kostenpost op en de signalen onderling hebben last van *skew*: de aankomsttijden zijn niet gelijk. Seriële data-overdracht heeft deze problemen niet, maar hier moet je natuurlijk langer wachten tot alle data verzonden is. Seriële communicatie is tegenwoordig zeer snel: 10 Gb/s is makkelijk te halen.

Een zeer eenvoudige vorm van seriële communicatie is in 1962 ontwikkeld onder de officiële naam EIA/TIA-232-F[4], beter bekend als RS-232[5] (RS staat voor "Recommended Standard"). De specificatie beschrijft een tal van zaken zoals de opbouw van de connectoren, de signaalnamen en de betekenis van de signalen, logica- en spanningsniveau's, frame-opbouw. Het beschrijft niet wat de betekenis van de verstuurde data is, dat wordt aan de applicatie overgelaten.

De RS-232-standaard is zeer uitgebreid. De standaard beschrijft een tal van signaalnamen en configuraties. Veel van deze zaken zijn terug te voeren naar het gebruik van modems en verbindingen via de ouderwetse telefoonlijnen. Wij zullen ons houden aan een eenvoudige variant.

Signalen

De meest eenvoudige vorm van RS-232-communicatie bestaat uit slechts drie signalen en dus drie draden: TxD (transmit data), RxD (receive data) en GND (ground). Hardwarematige *handshaking* is niet mogelijk en dat houdt in dat zender en ontvanger voldoende snel moeten zijn om de datastroom te kunnen verwerken. Merk op dat er geen kloksignaal is. Een mogelijke connector is de zogenaamde 9-pins Sub-D connector[6]. Pin 2 op de connector wordt gebruikt voor RxD en pin 3 wordt gebruikt voor TxD, tenminste als je het over een PC hebt. Bij een modem is het andersom. Daarom staat RS-232 ook wel voor "Regelmatig Solderen van pin 2 naar pin 3 naar pin 2".

Spanningsniveau's

De fysieke spanningen voor de datalijnen van RS-232 zijn als volgt gedefinieerd[7]

logisch '0'	+3 V tot +25 V	'space'
logisch '1'	−3 V tot −25 V	'mark'

[4] Versie 12 juli 2012
[5] http://en.wikipedia.org/wiki/RS-232
[6] http://www.dataip.co.uk/Reference/Serial9DPinOut.php
[7] https://www.lammertbies.nl/comm/info/RS-232_specs.html

Het spanningsbereik tussen −3 V en +3 V is dus ongeldig. Wat opvalt is dat een negatieve spanning een logische '1' representeert en een positieve spanning een logische '0'. Hoewel de spanningen in het gebied van −25 V tot +25 V liggen zijn de *nominale* spanningen −12 V en +12 V. Merk op dat de zender een iets hogere minimale spanning moet leveren in verband met spanningsval op de signaallijnen.

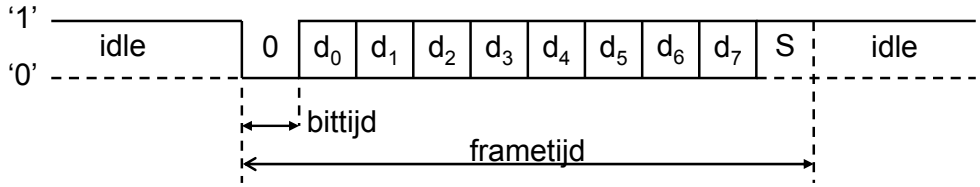
Omdat er veel gebruik wordt gemaakt van microcontrollers en digitale systemen bestaat er ook zoiets als TTL-level RS-232. Daarin wordt een logische '0' voorgesteld met 0 V (ground) en een logische '1' met de voedingsspanning bv. +3,3 V of +5 V. Om signalen uit te wisselen tussen echte RS-232-systemen en TTL-level wordt gebruik gemaakt van level shifters zoals de MAX232[8]. In deze opdracht wordt gebruik gemaakt van TTL-level RS-232 met een voedingsspanning van +3,3 V.

Seinsnelheid

RS-232 is eigenlijk bedoeld voor seinsnelheden tot 20000 bps (bits/sec). In RS-232-speak wordt dit de *baud rate*[9] genoemd. Deze term heeft meer met signalering tussen modems te maken dan met zuivere bits. In de praktijk worden snelheden van 115200 bps en hoger gebruikt. Windows stelt de seriële poorten standaard in op 9600 bps. Wij zullen in deze opdracht 9600 bps en 115 200 bps[10] gebruiken als seinsnelheid.

Frameformaat

Het frameformaat bestaat uit een startbit, gevolgd door een aantal databits, een optionele *pariteitsbit* en één of meer stopbits. De pariteitsbit wordt gebruikt voor foutdetectie. Het geeft aan of het aantal bits in de te verzenden data even (Engels: even) of oneven (Engels: odd) is. In de praktijk wordt de pariteitsbit nauwelijks gebruikt vanwege de beperkte foutdetectiemogelijkheden. Het aantal databits varieert tussen vijf en negen. De meest gebruikte combinatie is acht databits, geen pariteit en één stopbit. Dit wordt aangegeven met de afkorting 8N1. In figuur 8 is het frameformaat geschetst.



Figuur 8: *Het 8N1-formaat van een RS-232 frame*

In rust (idle mode) is de zendlijn (TxD) logisch '1'. De startbit heeft de logische waarde '0' (zie uitleg verderop). Daarna volgen de acht databits beginnend met het minst significante bit. Daarna volgt een stopbit die logisch '1' is. Je zou kunnen denken dat deze weggelaten kan worden omdat de lijn in rust toch al logisch '1' is. Maar dan kan je het verschil niet zien tussen een laatste databit die '0' is en een startbit. De

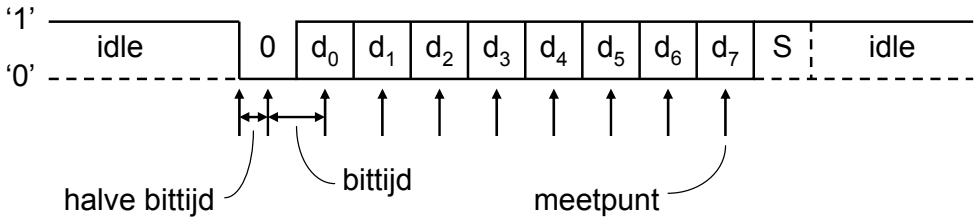
[8] http://www.ti.com/lit/ds/symlink/max232.pdf
[9] http://en.wikipedia.org/wiki/Baud
[10] http://www.finseth.com/parts/serial.php

stopbit is dus nodig voor synchronisatiedoeleinden tussen zender en ontvanger. In deze opdracht gebruiken we het formaat 8N1.

Zenden en ontvangen

Het is mogelijk om tegelijk te zenden en te ontvangen, er zijn immers aparte zend- en ontvangstlijnen. Dat wordt *full duplex*[11] genoemd. Dat houdt in dat het systeem een aparte zend- en ontvangstmodule bevat.

Het zenden gaat als volgt. Eerst wordt de startbit "op de lijn gezet". Daarna volgen de acht databits en vervolgens volgt de stopbit. Daarmee is de transmissie klaar. Bij het ontvangen komt wat meer kijken. De ontvanger wacht tot er op de ontvangstlijn een hoog-laag wisseling plaatsvindt. Dat is het teken dat er een startbit is verzonden. Om er zeker van te zijn dat het echt om een startbit gaat en niet om een *spike* of *glitch* wordt na een *halve* bittijd de ontvangstlijn nog eens bekeken. Blijkt het echt om een startbit te gaan, dan wordt de ontvangst doorgezet. Daarbij is het verstandig om niet te dicht bij de signaalwisselingen van de bits te bemonsteren[12]. De exacte momenten liggen namelijk niet geheel vast (jitter van de klok aan de zendkant) en er is sprake van stijg- en daaltijden van de signalen. Merk ook op dat er geen kloksignaal wordt meegestuurd, het is dus van belang dat de klokken aan de zend- en ontvangstkant op dezelfde frequentie lopen. De maximale afwijking is ongeveer 5%[13]. Het is dus verstandig om in het midden tussen twee signaalwisselingen te bemonsteren. Zie figuur 9.



Figuur 9: *Timing binnen RS-232 frame.*

Klok en systeemklok

Het gebruikte frameformaat is 8N1, de seinsnelheid is 9600 bps. Merk op dat het DE0-bord een 50 MHz klokoscillator heeft. Het aantal klokpulsen dat gelijk is aan één bittijd is dan:

$$N = \frac{f_{clk}}{S} = \frac{50000000}{9600} = 5208{,}333\cdots \tag{1}$$

Helaas is het antwoord geen geheel getal. Dat houdt in dat de seinsnelheid van 9600 bps niet exact kan worden gehaald. We ronden dit af naar 5208. Terugrekenen:

$$S = \frac{f_{clk}}{N} = \frac{50000000}{5208} = 9600{,}6144\cdots \tag{2}$$

[11] http://uva.ulb.ac.be/cit_courseware/datacomm/dc_014.htm

[12] Bemonsteren = samplen

[13] Dat is eenvoudig uit te rekenen. Een 8N1 RS-232-frame bestaat uit 10 bits. Het bemonsteren begint halverwege de startbit. Dan kan de ontvanger een afwijking hebben van een halve bittijd op 10 bittijden = 1/20 = 5%.

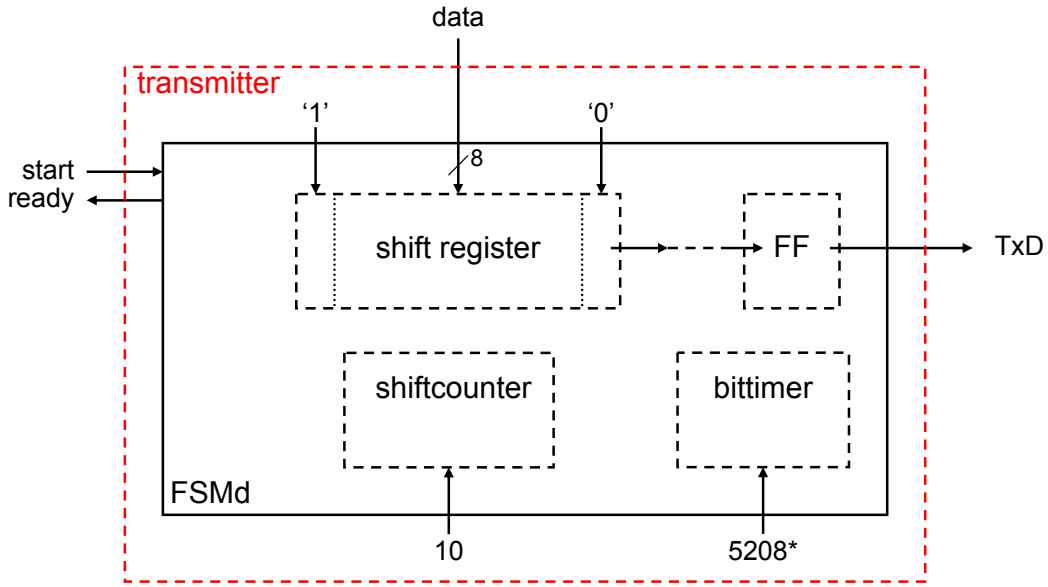
Dit levert een relatieve afwijking van:

$$\delta = \frac{0{,}6144\cdots}{9600} \cdot 100\% = 0{,}0065\% \tag{3}$$

Het mag duidelijk zijn dat dit geen probleem oplevert. Merk op dat de klokoscillator ook niet precies 50 MHz afgeeft.

Systeemopbouw

Het systeem heeft een aparte zend- en ontvangstmodule. In figuur 10 is het blokschema van de zender gegeven.



Figuur 10: *Opbouw van de zender*

Het blokschema is opgesteld volgens het "FSM met datapad"-model (FSMd). Hierin zijn de diverse datapad-onderdelen geïntegreerd in de FSM. Dit wordt in de figuur aangegeven met stippellijntjes om de diverse datapad-onderdelen. De datapad-onderdelen hebben diverse functies, zoals laden, schuiven en aftellen. VHDL leent zich uitstekend voor het integreren van tellers en schuifregisters in een FSM. Zie verder de paragraaf FSM met datapad.

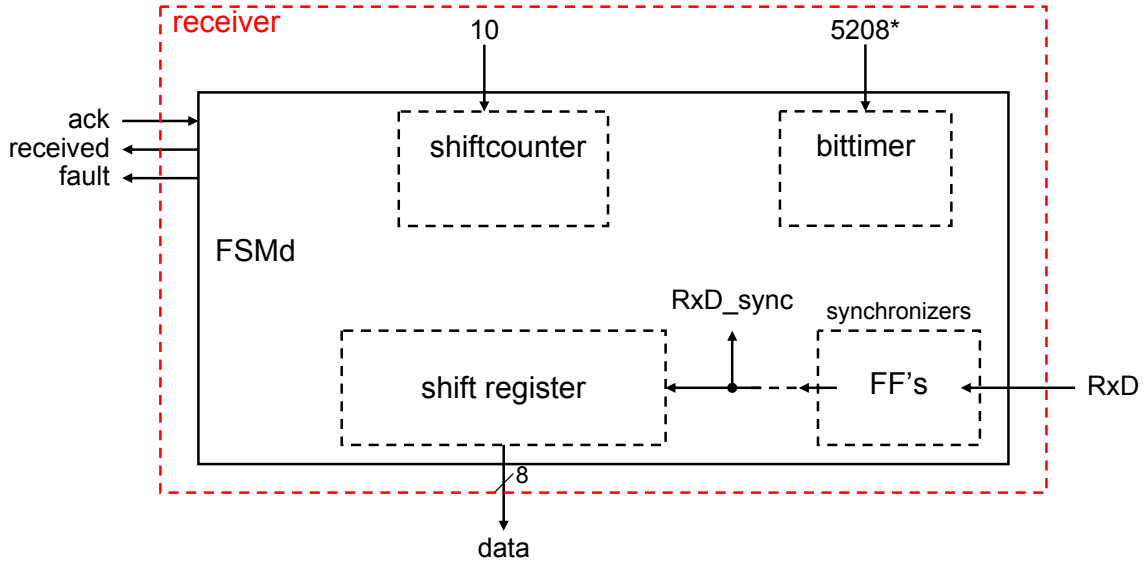
Het schuifregister wordt gebruikt om de data die verstuurd moet worden bit voor bit "naar buiten" te schuiven. Het schuifregister is 10 bits breed zodat ook de startbit en stopbit geladen en geschoven kunnen worden. De flipflop FF zorgt ervoor dat de verzonden bit *single transition* is. Let erop dat de TxD-lijn in rust logisch 1 moet zijn. De bittimer telt het aantal systeemklokpulsen en bepaalt hiermee de (tijd-)lengte van een bit. De waarde 5208 is eerder al uitgelegd. Zie hiervoor ook paragraaf Opzet van de VHDL-code. De shiftcounter houdt het aantal verzonden databits bij. Het geheel wordt geïntegreerd in en aangestuurd door een FSM.

In figuur 11 is het schema van de ontvanger gegeven. Het schuifregister wordt gebruikt om de te ontvangen bits in te lezen. De flipflops FF zijn twee synchronizers (dubbele

synchronisatie). De bittimer en de shiftcounter hebben dezelfde functie als bij de ontvanger. De bittimer heeft ook de mogelijkheid om een halve bittijd af te tellen om de test op een correct startbit mogelijk te maken. Het geheel wordt geïntegreerd in en aangestuurd door een FSM.

Let erop dat bij het verzenden eerst de minstwaardige bit moet worden verzonden!

Let erop dat bij het ontvangen eerst de minstwaardige bit wordt ontvangen!



Figuur 11: *Opbouw van de ontvanger*

Verbindingen

Op het DE0-bordje zijn drie signalen nodig voor communicatie: één voor zenden, één voor ontvangen en een gemeenschappelijk ground.

Het bordje heeft twee GPIO-connectoren. GPIO staat voor *General Purpuse I/O*. Ze worden ook wel *extension connectors* genoemd. In figuur 12 zijn de pinaansluitingen voor GPIO0 weergegeven.

Pin 10 (GPIO0_D7, `PIN_AA13`) wordt gebruikt voor zenden.

Pin 14 (GPIO0_D9, `PIN_AA10`) wordt gebruikt voor ontvangen.

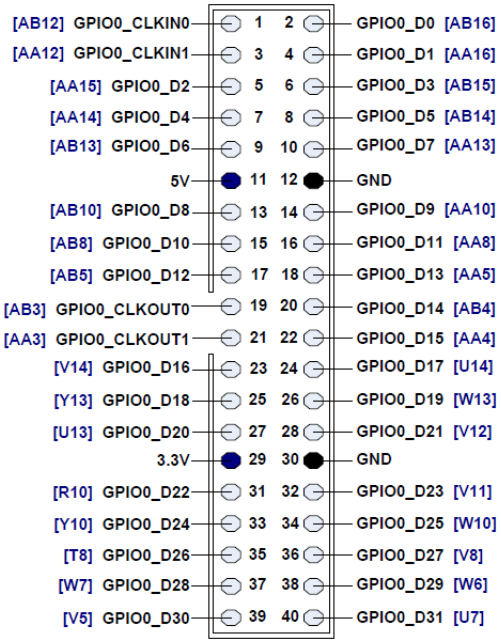
Pin 12 is de gemeenschappelijke ground.

Let op met aansluiten van de zend- en ontvangstdraden! De zenddraad bij het ene bordje is de ontvangstdraad bij het andere bordje!

De te verzenden data kan je het beste aanbieden via de schuifschakelaars. De signalen `start` en `ack` kan je het beste aanbieden via de drukknoppen.

Leerdoelen

De leerdoelen van deze opdracht zijn:



Figuur 12: *Pinaansluitingen van de GPIO0*

- Realiseren van eenvoudige datacommunicatie tussen twee systemen met behulp van een protocol.
- Ontwerpen van een complexe toestandsmachines met geïntegreerd datapad (FSMd, control en datapad ineen) vanuit een geschreven specificatie.
- Coderen van het complete systeem in VHDL.
- Opzetten van een simulatie van de FSMd's in VHDL.

Opdrachten

De opdracht luidt:

"Ontwerp, implementeer en test een serieel zend-ontvangstsysteem gebaseerd op de hierboven beschreven eenvoudige vorm van RS-232."

De ontvangen data moet als twee hexadecimale cijfers op de zeven-segmenten displays worden afgebeeld.

De volgende stappen moeten worden doorlopen:

a) Ontwerp (op papier) de toestandsdiagrammen voor de FSM's. De zender en ontvangen moeten in één entity worden ondergebracht.

b) Codeer het geheel in VHDL m.b.v. *FSM with Datapad* (FSMd). Het gebruik van de State Machine Editor is *niet* toegestaan.

c) Simuleer de zender en de ontvanger op de werkelijke snelheid. De systeemklok is dus 50 MHz.

d) Implementeer het geheel op een DE0-bordje. Synchroniseer de ingangen!

e) Test je eigen zend- en ontvangstmodule met behulp van een zogenaamde *loop-back*-verbinding.

f) Test je eigen zend- en ontvangstmodule door jouw systeem te koppelen aan dat van een ander (dat systeem moet natuurlijk wel correct werken). Je kan jouw ontwerp testen met een *referentie-implementatie*. Zie paragraaf Testen.

Opzet van de VHDL-code

Bij deze opdracht is het van belang dat je datapad en FSM geïntegreerd beschrijft. Je hoeft slechts één entity aan te maken. In deze entity komen twee processen: één voor de zender en één voor de ontvanger. Dat bespaart heel veel typewerk. In de processen beschrijf je kort en krachtig de FSM's voor zenden en ontvangen. In de FSM's wordt ook data verwerkt, zoals het laden van tellers met een beginstand en het schuiven van data. Dat kan alleen maar onder besturing van een klokklank.

Het is verstandig om een aantal constanten echt als constanten in de code op te nemen, bijvoorbeeld systeemfrequentie en seinsnelheid. Het is niet slim om het getal 5208 hard in je code te plaatsen. Gebruik i.p.v. daarvan VHDL-code die dit getal uitrekent vanuit de systeemfrequentie en de seinsnelheid. Wil je dan een andere seinsnelheid gebruiken, dan hoef je alleen deze constante aan te passen en wordt de bittijd bij hercompilatie automatisch bepaald.

Ontwerp volgens de regels van de digitale techniek:

- De ingangen van het systeem (`RxD`, `ack` en `start`) moeten eerst gesynchroniseerd worden d.m.v. dubbele synchronisatie.
- Alle geheugenelementen zijn flankgevoelig (flipflops). Er mogen geen latches gebruikt worden.
- Alle flipflops moeten d.m.v. een asynchrone reset in een bekende toestand gedwongen kunnen worden.
- Alle flipflops moeten worden aangesloten op hetzelfde kloksignaal, de interne klok van 50 MHz. Geen andere prescalers/kloksignalen gebruiken!
- Alle interne en externe signalen moeten synchroon verwerkt worden, met uitzondering van de asynchrone reset.

Opmerkingen

Bij deze opdracht is samenwerken met andere studenten zeer gewenst.

Let op: jij moet de seinsnelheid van 9600 bps en 115200 bps implementeren! De seinsnelheid moet bij compilatie worden ingesteld.

Let op: jij hoeft alleen de normale werking (dus geen valse start) te implementeren!

FSM met datapad

VHDL leent zich uitstekend voor het integreren van tellers en schuifregisters in een FSM. Het kenmerkende van een FSMd is dat zowel toestandstoekenning als datapadtoekenning in één proces onder klokflanksturing worden geplaatst. Dat levert krachtige, compacte code. Nadeel is dat het testen lastiger is dan bij gescheiden FSM en datapad, alles moet in één keer getest worden. Ook is het lastig om een goed blokschema te tekenen, er is immers maar één blok.

In listing 6 is een voorbeeld van een FSMd gegeven.

```
entity rs232_module is
  generic (sys_freq : integer := 50000000;
           baudrate : integer := 9600);
  ...
end entity;

...
-- number of system clock pulses per bit time
constant bittime : integer := sys_freq / baudrate;
-- counter to count the bit time
signal bittime_counter : integer range 0 to bittime-1;
...
signal receive_shifter : std_logic_vector(9 downto 0);
...
-- The state of the FSMd
type state_type is (..., load_timer, read_data_bit, next_bit, ...);
signal state : state_type;
...

  if areset = '1' then
    ...
  elsif rising_edge(clk) then
    case state is
      ...
      when load_timer =>
          bittime_counter <= bittime - 1;
          state <= read_data_bit;
      when read_data_bit =>
          if bittime_counter > 0 then
            -- bit time not expired, so update ...
            bittime_counter <= bittime_counter - 1;
          else
            -- bit timer expired, so restore bit time and shift in
            bittime_counter <= bittime - 1;
            receive_shifter <= input & receive_shifter(9 downto 1);
            state <= next_bit;
          end if;
        when next_bit => ...
      ...
    end case;
  end if;
```

Listing 6: *Voorbeeld FSMd met constanten*

Testen

Voor het testen van je eigen implementatie kan gebruik gemaakt worden van een referentie-implementatie en een tweede DE0-bordje.

Download het bestand `rs232-ref.sof` van BlackBoard[14] en programmeer het bestand in de FPGA. Dat kan via de programmer-software van Quartus.

De indeling van de schakelaars en leds is als volgt:

LEDG(9)	zender klaar (ready)
LEDG(8)	ontvanger klaar (received)
LEDG(7-0)	acht bits ontvangen data
SW(9)	selectie snelheid (0 = 9600, 1 = 115200)
SW(8)	selectie valse start (0 = gewone start, 1 = valse start)
SW(7-0)	te verzenden data
BUTTON(2)	start transmissie
BUTTON(1)	acknowlegde (bevestig ontvangst)
BUTTON(0)	asynchrone reset
HEX1-HEX0	ontvangen data in hex-formaat
HEX0(DP)	valse startbit gedetecteerd

De werking bij het zenden is als volgt:

a) Stel het te verzenden patroon in op de schakelaars SW7 - SW0.

b) Kies de transmissiesnelheid met SW9: 0 = 9600, 1 = 115200.

c) Kies voor normale transmissie of valse-start-transmissie met SW8 (0 = normaal, 1 = valse start).

d) Druk op BUTTON2 om de transmissie te starten.

e) Nadat de transmissie is afgelopen brandt LEDG9 (ready) totdat BUTTON2 is losgelaten.

De werking bij het ontvangen is als volgt:

a) Reset het systeem eventueel door op BUTTON0 te drukken.

b) Wacht op ontvangst.

c) Nadat een goede transmissie is gelukt wordt de data zichtbaar op 7-segment display in de vorm van een hexadecimaal getal.

d) Als er een valse start is gedetecteerd, gaat HEX0_DP branden.

e) Om opnieuw data te kunnen ontvangen moet BUTTON1 gedrukt worden (acknowledge).

[14] Als alternatief kan je het bestand downloaden van http://ds.opdenbrouw.nl/digse2/rs232-ref.sof