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[1], beter bekend als RS-232[2] (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[3]. 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[4]

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

[1] Versie 12 juli 2012
[2] http://en.wikipedia.org/wiki/RS-232
[3] http://www.dataip.co.uk/Reference/Serial9DPinOut.php
[4] 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.

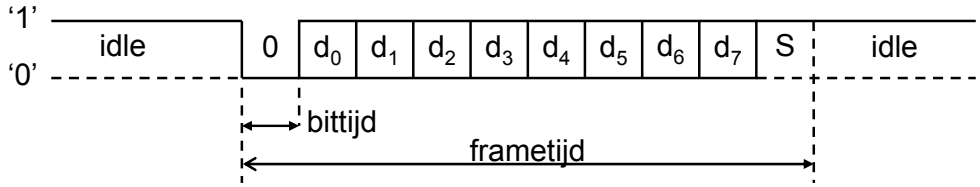
Omdat er veel gebruik wordt gemaakt van microcontrollers en digitale systemen bestaat er ook zoiets als TTL-level RS-232. Daarin wordt een logische '0' voorgesteld met 0 V (ground) en een logische '1' met de voedingsspanning bv. +3,3 V of +5 V. Om signalen uit te wisselen tussen echte RS-232-systemen en TTL-level wordt gebruik gemaakt van level shifters zoals de MAX232[5]. 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*[6] 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[7] 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 1 is het frameformaat geschetst.



Figuur 1: *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

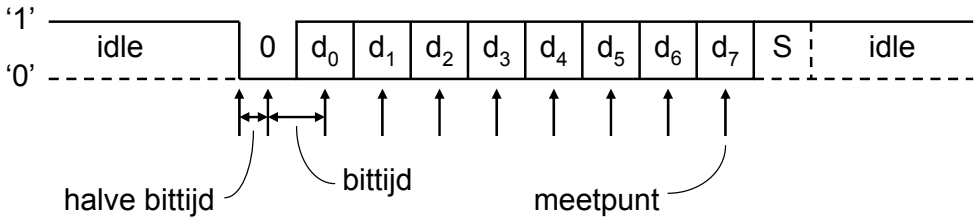
[5] http://www.ti.com/lit/ds/symlink/max232.pdf
[6] http://en.wikipedia.org/wiki/Baud
[7] http://www.finseth.com/parts/serial.php

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*[8] 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[9]. 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%[10]. Het is dus verstandig om in het midden tussen twee signaalwisselingen te bemonsteren. Zie figuur 2.



Figuur 2: *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:

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

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

[9] Bemonsteren = samplen

[10] 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%.

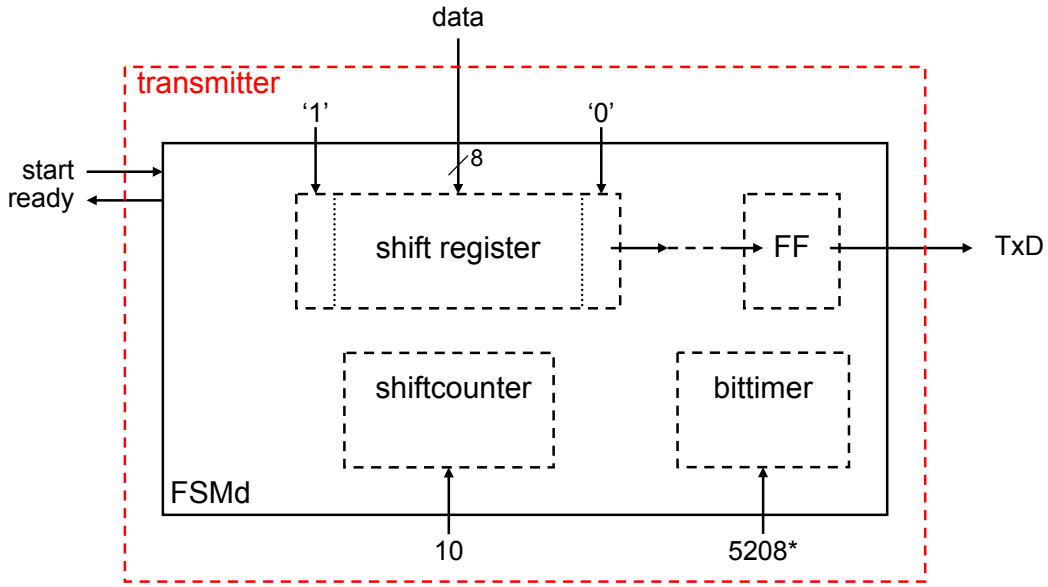
Dit levert een relatieve afwijking van:

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

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 3 is het blokschema van de zender gegeven.



Figuur 3: *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.

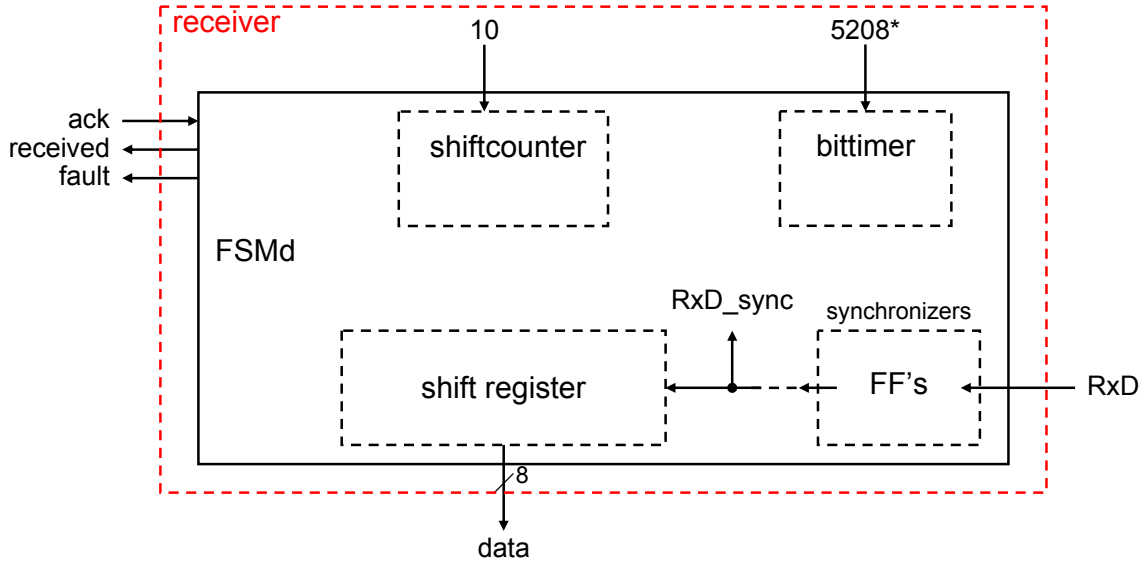
Het schuifregister wordt gebruikt om de data die verstuurd moet worden bit voor bit "naar buiten" te schuiven. Het schuifregister is 10 bits breed zodat ook de startbit en stopbit geladen en geschoven kunnen worden. De flipflop FF zorgt ervoor dat de verzonden bit *single transition* is. Let erop dat 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 4 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 4: *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 5 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.

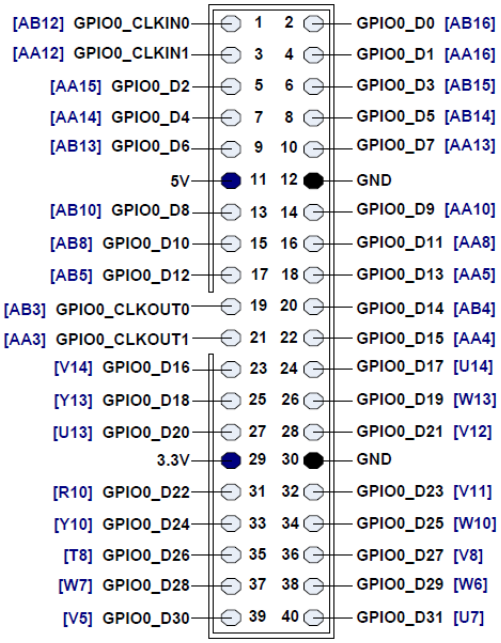
Pin 12 is de gemeenschappelijke ground.

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 5: *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 1 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 1: *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[11] 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).

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