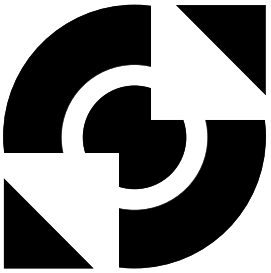
Universiteit Twente

faculteit EWI



Practicumhandleiding

Basisbegrippen Digitale techniek

E. Molenkamp

Vakcode: 192130014
November 2011
Versie 1.1

Inhoudsopgave

Practicumhandleiding Digitale techniek

1 Inleiding

Het doel van dit practicum is om met behulp van moderne gereedschappen daadwerkelijk een digitale schakeling te ontwerpen en deze te realiseren in een Field Programmable Gate Array (FPGA). Door het gebruik van gereedschappen hoeven arbeidsintensieve minimalisaties niet met de hand te worden uitgevoerd, zodat het mogelijk is vrij complexe systemen te maken.

Er is een **VHDL tutorial** (een apart document) waarmee ervaring op gedaan wordt met de omgeving, een **tussenopdracht** en een **eindopdracht**.
Voor de tussenopdracht is het te ontwerpen systeem al opgedeeld in subsystemen. Een subsysteem, of samenstelling van subsystemen, is in deze handleiding geformuleerd als een deelopdracht. In de eindopdracht wordt het ontwerp uitgebreid.

Voor de opdrachten die betrekking hebben op de tussen- en eindopdracht moet een journaal worden bijgehouden.

Het practicum wordt inhoudelijk beoordeeld en het journaal wordt beoordeeld. Beide moeten voldoende zijn. De inhoudelijke beoordeling is deels gebaseerd op de werkwijze tijdens het gehele practicum maar vooral op de technische kwaliteit van het ontwerp. Een onvoldoende beoordeling kan niet worden verbeterd.
Zie de studiehandleiding voor meer informatie over de beoordeling.

Alvorens met het practicum te kunnen beginnen, is het noodzakelijk dat men VHDL zorgvuldig heeft bestudeerd. Zonder kennis van VHDL is het practicum niet te doen. Verder moet men zich voor de te realiseren subsystemen houden aan de synthetiseerbare subset (zie o.a. paragraaf 5.2). Een assistent zal geen begeleiding geven indien blijkt dat de student zich onvoldoende heeft voorbereid en/of zich niet houdt aan de synthetiseerbare subset.

2 Planning en aftekenen van deelopdrachten

Er zijn 8 dagdelen verroosterd voor het practicum. Globaal kan de volgende planning worden gehanteerd:

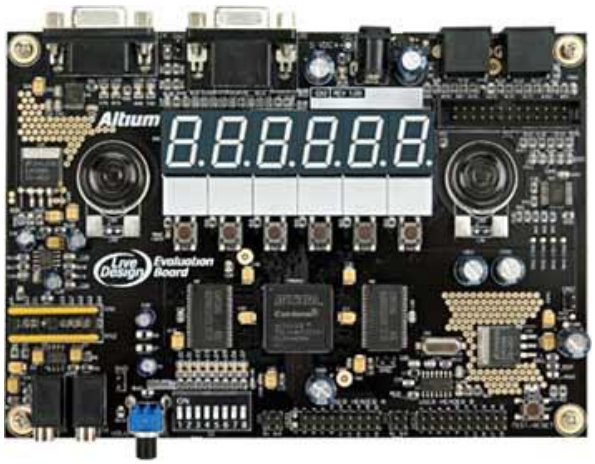
Dagdeel	
1	VHDL tutorial, Deelopdracht showkey
2	Deelopdrachten: Realisatie van showkey en constantkey
3	Deelopdrachten: Readkey, key2pulselength en “vermenigvuldig met een macht van twee”
4	Deelopdrachten: pulse2audio, tone generation en orgel
5	Eindopdracht
6	Eindopdracht
7	Eindopdracht
8	Eindopdracht

Zonder een goede voorbereiding voor het practicum loop je heel snel achter!

Indien men achter dreigt te raken is het verstandig thuis aan de opdracht verder te werken. De software kan thuis worden geinstalleerd en op andere momenten kan soms ook gebruik worden gemaakt van de practicumruimten.

Zorg dat het journaal up-to-date is. De assistent zal regelmatig aftekenen tot hoever de opdracht is gevorderd (dat is dus inclusief het journaal).
Bij een aantal opdrachten is expliciet aangegeven dat het resultaat ook aan de assistent moet worden getoond.

3 Apparatuur



Figuur 1: LiveDesign Evaluation Kit met Altera device.

Tijdens dit practicum wordt gebruik gemaakt van een experimenteerbordje (figuur 1) meer informatie over dit bordje is hier te vinden www.cs.utwente.nl/~molenkam/midP

Er is een beperkt aantal experimenteerbordjes beschikbaar, zodat niet iedereen een eigen bordje heeft. **Het grootste deel van de tijd heb je ook geen experimenteerbordje nodig.** Voordat je gebruik kunt maken van het bordje moet eerst de simulatie goed zijn en mogen er tijdens synthese ook geen ernstige waarschuwingen zijn gemeld tot slot genereer je op je eigen werkplek een 'programmeerfile'. Het experimenteerbordje heb je dus slechts enkele minuten nodig!

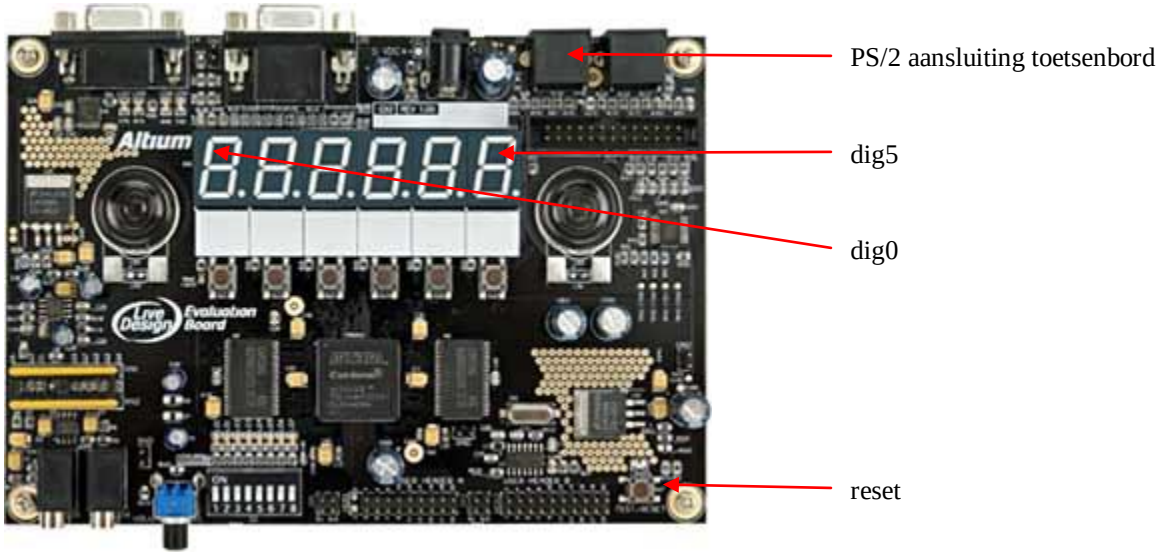
4 VHDL tutorial

Doorloop van de VHDL tutorial de hoofdstukken 1 t/m 6. Het programmeren van een FPGA (hoofdstuk 6) wordt nu nog niet uitgevoerd.

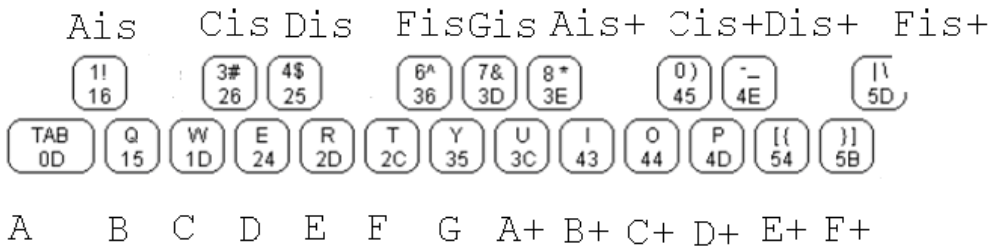
5 Globale specificatie

5.1.1 Inleiding

Bij dit practicum wordt gebruik gemaakt van het LiveDesign experimenteerbord (figuur 5.1). In deze handleiding zullen de relevante aspecten van dit bordje aan de orde komen.



Figuur 5.1:LiveDesign experimenteerbordje



Figuur 5.2 PS/2 keyboard met de toetsen van ons orgeltje

Een PS/2 toetsenbord wordt met het LiveDesign bord verbonden. Met behulp van de aangegeven toetsen in figuur 5.2 moet een (eenvoudig) deuntje kunnen worden gespeeld. De zwarte noten zijn de toetsen op de bovenste rij, de witte toetsen zijn de toetsen die overeenkomen met de onderste rij.
Een organist zal probleemloos twee of meer toetsen tegelijk indrukken. Dat zal ons orgeltje niet ondersteunen! Slechts één toets mag ingedrukt worden. Als compensatie worden de muzikale kwaliteiten niet meegenomen in de beoordeling.

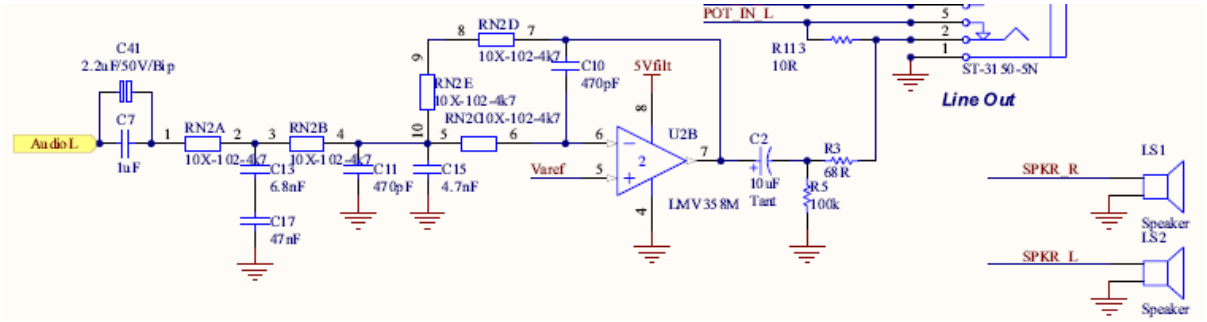
De achtergrond van de toonladder is (zie ook figuur 5.2):

- De toonladder bestaat uit 12 (hele/halve) noten A, Ais, B, C, Cis, D, Dis, E, F, Fis, G, Gis en dan weer A+ etc..
- De toon A heeft een frequentie van 440 Hz.
- De toon A+ is 1 octaaf hoger en heeft dus twee maal de frequentie van 440 Hz: 880 Hz.
- Er is een constante C zodanig dat $toon_{i+1}=C\times toon_i$. Verder geldt dus dat $toon_{i+12}=2\times toon_i$. Dus $C^{12}=2$

$$C = 2^{\frac{1}{12}} \sim 1{,}05946$$

Na het indrukken van de toets ‘A’ zijn alle tonen 1 octaaf hoger en na het indrukken van de toets ‘Z’ zijn alle tonen 1 octaaf lager. Wordt tweemaal op de toets ‘A’ gedrukt dan worden alle tonen 2 octaven hoger. Het

ontwerp dat gemaakt moet worden kan m.b.v. deze twee toetsen een gegenereerde toon van een toets over 6 octaven kunnen schuiven



Figuur 5.3 Filter tussen AudioL en luidspreker

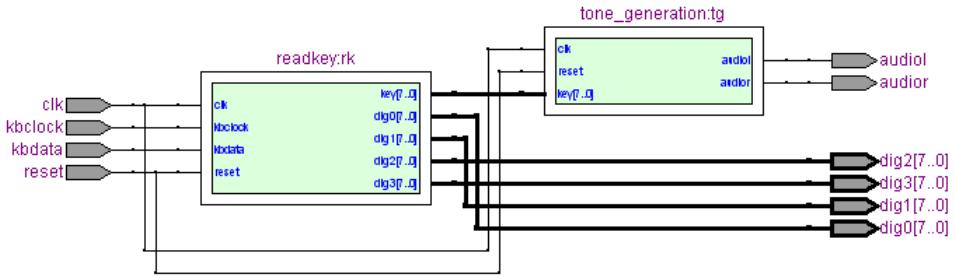
De PS/2 aansluiting is rechtstreeks verbonden met de programmeerbare device (een Altera FPGA). De twee uitgangspinnen (stereo) voor audio zijn niet rechtstreeks aangesloten op de luidsprekers. Figuur 5.3 geeft het analoge circuit aan dat nog op het experimenteerbordje aanwezig is. Dit is een interessant circuit, maar niet voor dit vak. De werking van (in ieder geval) de verschillende onderdelen komt bij de vakken ELBAS en ELFUN aan de orde. Voor deze opdracht is het volgende voldoende:

> Een toon met een frequentie van 100 Hz heeft een periodetijd van 10 ms. Als op de AudioL uitgang een periodiek signaal staat van 5 ms hoog (logisch ‘1’) en daarna 5 ms laag (logisch ‘0’), wordt de toon van 100 Hz gegenereerd.

Het filter zorgt er dus voor dat de gegenereerde blokgolf op de uitgang AudioL (en eveneens AudioR) van het programmeerbare device wordt omgezet in, bij benadering, een sinus met dezelfde periodetijd.

5.1.2 Herkennen en genereren van een toon

Ongetwijfeld zijn er vele mogelijkheden om het orgeltje te realiseren. In het kader van dit vak worden een groot aantal aanwijzingen gegeven zodanig dat na afloop ook een werkend orgeltje kan worden getoond. Men is verplicht zich te houden aan de opdeling zoals in dit document is behandeld.



Figuur 5.4 Opdeling van het orgel in twee subsystemen

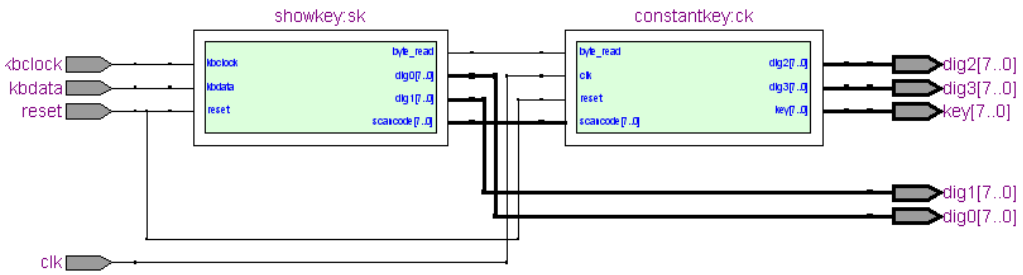
De communicatie van het toetsenbord met het programmeerbare device gaat door middel van KBCLOCK en KBDATA. Het toetsenbord genereert een code die afhankelijk is van de toets.
Wordt bijvoorbeeld de ‘TAB’ eenmaal ingedrukt dan wordt de hexadecimale code 0D gegenereerd. Blijf je de toets ingedrukt houden dan wordt deze code repeterend gegenereerd. Bij het loslaten van de ‘TAB’ toets worden achtereenvolgens twee bytes verzonden F0 gevolgd door 0D (de code van de TAB toets die eerder was ingedrukt). Vervolgens moet de gegenereerde code worden omgezet in een toon. Figuur 5.4 geeft dan ook de opdeling in de twee subsystemen:

- Readkey
  Als op een toets wordt gedrukt wordt, zolang deze toets is ingedrukt, continue de daarbij behorende key code (byte) op de uitgang van readkey gezet.
- Tone_generation
  Zolang er een keycode ongelijk aan 00_{16} op de ingang van de tone_generation unit staat wordt de daarbij behorende blokgolf (=toon) op de beide audio aansluitingen gegenereerd. AudioL en AudioR

krijgen hetzelfde signaal aangeboden (mono). Als op de ingang 00_{16} staat wordt er geen blokgolf gegenereerd: op beide uitgangen staat een ‘0’ (geen geluid).

In figuur 5.4 zijn nog 4 extra uitgangen aangegeven: dig0 t/m dig3. Deze vier 7-segmentdisplays worden gebruikt voor het debuggen.
De uitgangen dig0 en dig1 laat, in hexadecimaal notatie, de laatst gegenereerde code van het toetsenbord zien. Hiermee is redelijk goed te controleren of de hardware de code van het toetsenbord correct heeft ontvangen. De uitgangen dig2 en dig3 laten constant, in hexadecimaal notatie, de code zien van de toets die op dat moment is ingedrukt of 00_{16} indien er geen toets is ingedrukt.

5.1.3 Readkey



Figuur 5.5 Readkey

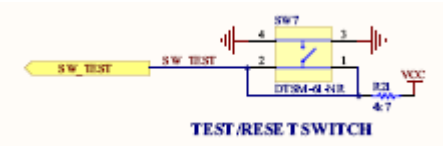
Het is nu tijd om in meer detail te kijken naar de werking van het toetsenbord. **Lees daarom eerst de pagina’s 1 t/m 4 van het document keyboard.pdf**.
We zijn alleen geïnteresseerd in de werking van de communicatie van het keyboard naar de pc, daarbij beperken we ons ook nog tot die toetsen die zijn weergegeven in figuur 5.2 met daarbij nog de toetsen ‘A’ (octaaf hoger) en ‘Z’ (octaaf lager).
Voor al deze toetsen geldt:

- als de toets wordt ingedrukt wordt een byte gegenereerd: < keycode >
- als deze toets lang wordt ingedrukt wordt deze byte repeterend verzonden: < keycode >,< keycode >..
- als de toets wordt losgelaten worden twee opeenvolgende bytes verstuurd: $F0_{16}$, < keycode >

N.B.: **Er mag slechts één toets ingedrukt worden op het toetsenbord**. Het protocol wordt aanzienlijk complexer als twee of meer toetsen tegelijk worden ingedrukt.

5.1.3.1Showkey

Het subsysteem Showkey krijgt zijn input alleen van het toetsenbord. Dit onderdeel werkt op de gegenereerde klok van het toetsenbord: KBCLOCK. Daarnaast heeft dit subsysteem, en ook de andere subsystemen, een reset. Het relevante deel van het schema voor de reset is gegeven in figuur 5.6. Hieruit valt op te maken dat als op de reset een ‘0’ wordt aangeboden (de schakelaar wordt bediend) de reset geactiveerd dient te worden.

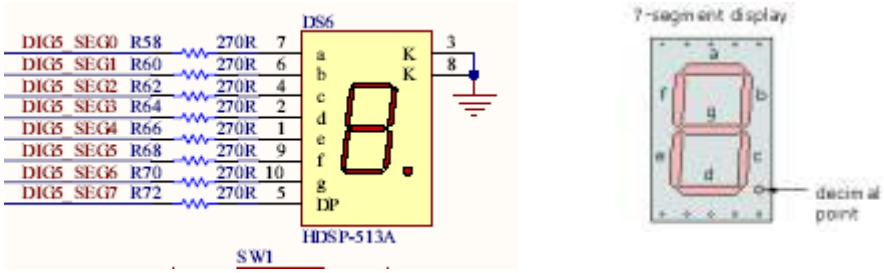


Figuur 5.6 Resetcircuit op het LiveDesign bord

Realiseer je dat in RUST het toetsenbord géén klok genereert. Deze kloklijn, KBCLOCK, is dan continue hoog. Als een toets, uit onze beperkte set van toetsen, wordt ingedrukt wordt 11 bits verzonden:
Start bit, byte met de keycode (van LSB naar MSB), Odd parity bit, Stop bit.

N.B. 1: In het ontwerp negeren we het parity bit.
N.B. 2: Realiseer je dat de gegenereerde data, KBDATA, stabiel is op de neergaande flank van KBCLOCK. (zie ook keyboard.pdf)

Het subsysteem genereert ook nog het signaal BYTE_READ. Dit uitgangssignaal is ‘1’ zodra de 11 verzonden bits zijn ontvangen. Dit signaal wordt door het subsysteem CONSTANTKEY gebruikt; daarover later meer.
Zodra het toetsenbord weer een byte verstuurt (bijv. bij het lang indrukken van de toets, dan wel loslaten van de toets) zal BYTE_READ eerst weer ‘0’ worden en na 11 ontvangen bits weer ‘1’.
Voor het debuggen heeft dit systeem nog de twee 7-segment uitgangen: dig0 en dig1. Op dig0 komt, in hexadecimale notatie, de waarde van de vier hoogstwaardige bits van de laatst gegenereerde code en op dig1 komt, in hexadecimale notatie, de waarde van de vier laagstwaardige bits

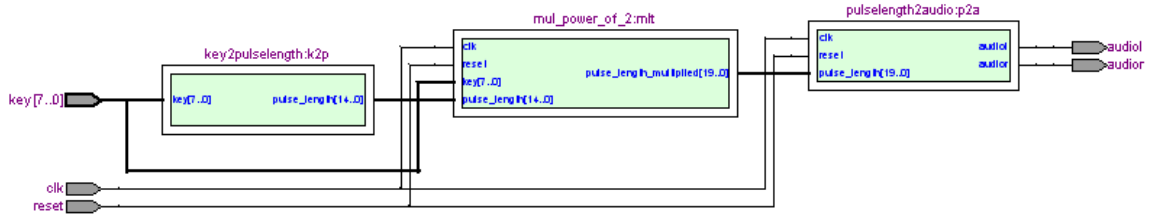


Figuur 5.7 fragment schema van demonstratiebordje met 7-segmentdisplay met gebruikelijke codering van de leds

5.1.3.2Constantkey

Het subsysteem CONSTANTKEY werkt op de klok van het experimenteerbordje (50 MHz). Zoals eerder is beschreven geeft het uitgangssignaal BYTE_READ aan dat de SCANCODE geldig is.
Wanneer een toets wordt ingedrukt moet de daarbij behorende scancode continue op de uitgang van CONSTANTKEY komt te staan. Er zijn hier vele mogelijkheden. In dit practicum zullen we kijken naar het moment waarop in de vorige klokperiode (van de 50 MHz klok) nog een ‘0’ op BYTE_READ stond en in de huidige klokperiode BYTE_READ ‘1’. De informatie van de scancode (keycode) op de ingang wordt dan gebruikt. Hierover later meer.
Op de uitgangen dig2 en dig3 komen respectievelijk de constante waarden in hex van de 4 meest significante bits en de 4 minst significante bits; ook weer t.b.v. debug doeleinden.

5.1.4 Tone generation



Figuur 5.8 Toongeneratie

Het subsysteem `tone_generation` (figuur 5.8) bestaat uit een drietal subsystemen. Het subsysteem `tone_generation` krijgt zijn input van het subsysteem `readkey` (zie ook figuur 5.4).
Het gedrag van de 3 subsystemen is globaal als volgt te omschrijven:

- Key2pulselength
  Als één van de toetsen van figuur 5.2 is ingedrukt moet een blokgolf worden gegenereerd met dezelfde periodetijd als die van de gewenste toon. De uitgang `pulse_length` geeft dus het aantal klokperioden van `clk` dat het uitgangssignaal repeterent hoog en laag moet zijn. De uitgang `pulse_length` is dus zelf een constante waarde.
- Mul_power_of_2
  Door middel van de toetsen ‘A’ en ‘Z’ (input `key`) wordt de octaafverandering gerealiseerd. Een octaaf hoger/lager betekent dat de frequentie wordt verdubbeld resp. gehalveerd.
- Pulselength2audio
  Dit subsysteem genereert een blokgolf op de beide uitgangen met een periodetijd van 2× `pulse_length` ×periodetijd van `clk`.

5.2 Subset VHDL: om te onthouden!

We ontwerpen alleen combinatorische of synchrone systemen. Wijk je af van onderstaande raamwerken dan zal de assistent geen ondersteuning kunnen geven. Zorg er ook voor de beschrijving netjes is uitgelijnd (inspringen).

5.2.1 Synchroon systeem

Gebruik voor een synchroon systeem uitsluitend onderstaand raamwerk:

```
process(reset,clk)
 <hier locale declaraties>
begin
  if reset='0' then       -- keuze van '0' is handig voor later!
    <hier het resetten van signalen/variabelen>
  elsif rising_edge(clk) then
    <hier het gewenste gedrag, sequentieel uiteraard>
  end if;
  <hier NIETS!!!!>
end process;
```

Het is soms heel verleidelijk om op een verandering van een ingangssignaal te reageren. Dus bijv:
`if rising_edge(button) then`
Tijdens simulatie wil dit nog wel goed gaan maar in de echte hardware krijg je te maken met grote en meestal lastig op te sporen problemen. Dus alleen rising_edge of falling_edge gebruiken voor een klok! In één synchroon systeem niet rising_edge en falling_edge door elkaar gebruiken.

5.2.2 Combinatoriek

Combinatoriek kan door middel van een concurrent statement worden beschreven. Soms kan het prettig zijn om hiervoor een process te gebruiken. Wanneer een process wordt gebruikt moet rekening worden gehouden met de volgende voorwaarden (helaas niet voldoende):

- gebruik een process met een sensitivity list: **process**(lijst met signalen).
- alle signalen die je leest (in de expressie) komen in de sensitivity list. In ModelSim kan hier op worden gecontroleerd met de optie "check_synthesis". Zie paragraaf 7.1.1.
- variabelen eerst een waarde geven voordat je deze leest, immers je wilt geen geheugenwerking!
- gebruik **geen** 'event, rising_edge of falling_edge.

5.2.2.1Combinational loop

Als je tijdens synthese de volgende melding krijgt:
Warning: Timing Analysis is analyzing one or more combinational loops as latches
dan heb je ergens in je beschrijving geheugenwerking beschreven waarbij geen gebruik is gemaakt van een latch. Corrigeer dit eerst.

5.2.3 Verschil variabelen en signalen

Het kan niet vaak genoeg worden vermeld:

- variabelen worden direct aangepast! De toekenningsoperator is `:=`
- signalen veranderen nooit direct! De toekenningsoperator is `<=`
  Een uitzondering is gemaakt voor de initiële waarde: `:=`

5.3 Journaal

Zorg er voor dat je in het journaal tenminste het volgende op neemt: VHDL beschrijvingen (inclusief VHDL testbench), relevante delen van een waveform (en een toelichting hierbij), schema's (RTL view) van het syntheseresultaat en natuurlijk de problemen en oplossingen. Een journaal is geen verslag! In een journaal staan de onderwerpen in de volgorde waarin je ze hebt gedaan (inclusief de problemen)!
Begin direct met het bijhouden van het journaal! Een journaal hoort altijd up-to-date te zijn.

De assistent kan alleen aan de hand van het journaal vaststellen hoe ver jullie zijn gevorderd.

Voor het journaliseren wordt verwezen naar de **Algemene Practicum Handleiding** (APH) Elektrotechniek.
Zie: http://onderwijs.el.utwente.nl/Documenten/Documenten

5.4 Opdrachten

In deze paragraaf vind je de opdrachten. Een globale omschrijving van de subsystemen is al eerder in dit hoofdstuk gegeven.

5.4.1 Deelopdracht Showkey

LET OP: deze deelopdracht kan alleen gemaakt worden nadat paragraaf 5.1.3 goed is bestudeerd. Daarnaast moeten nog de pagina's 1 t/m 4 van het document keyboard.pdf zijn bestudeerd.

Op pagina 4 van het document keyboard.pdf is een tijddiagram van het PS/2 protocol gegeven dat wordt gebruikt door het toetsenbord. Afhankelijk van de toets die wordt ingedrukt wordt een andere "scancode" gegenereerd (pagina 3 in het document keyboard.pdf). De data en clock zijn respectievelijk verbonden met `kbdata` en `kbclock` van de ingangen van entity `showkey`.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY showkey IS
  PORT (
    reset     : IN std_logic;
    kbclock   : IN std_logic; -- low freq. clk (~ 20 kHz) from keyboard
    kbdata    : IN std_logic; -- serial data from the keyboard
    dig0, dig1: OUT std_logic_vector(7 DOWNTO 0);
    -- shows the key pressed on display in Hex dig0 (upper 4 bits) dig1 (lower 4 bits)
    scancode  : OUT std_logic_vector(7 DOWNTO 0);
    byte_read : OUT std_logic
    );
END showkey;
```

Opdracht 1: De entity beschrijving van `showkey` is hierboven gegeven. Gevraagd wordt een architecture beschrijving voor `showkey`. (Maak alleen gebruik van de subset van VHDL die ook wordt ondersteund door de synthestool.)

N.B. De VHDL bestanden in dit document staan op Blackboard. Dus niet kopieren uit dit document (de formatering gaat dan verloren).

5.4.1.1Testbench voor Showkey

Het subsysteem `showkey` gaan we ook realiseren op het experimenteerbordje. Echter voordat het systeem wordt gerealiseerd moet dit eerst worden gesimuleerd. Om dit mogelijk te maken is een (te) eenvoudige testopstelling in VHDL beschreven, zie figuur 5.9 (file `showkey_simple_test.vhd`).

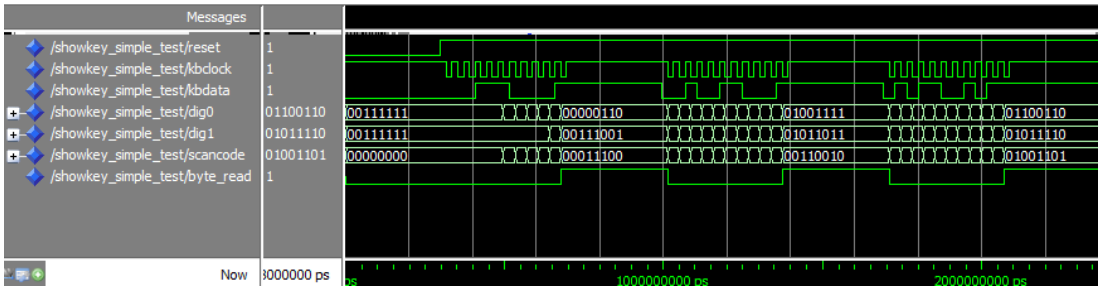
Een toelichting bij deze beschrijving:

- Function `hex2display`
  Als een byte wordt verstuurd zien we op het display de hexadecimale notatie van de 4 hoogstwaardige en 4 laagstwaardige bits van de byte. Deze functie verzorgt de conversie.
  Tip: deze functie is ook synthetiseerbaar!
- Procedure `send_byte`
  In de testomgeving willen we data oversturen van toetsenbord naar `showkey`. Het protocol hiervoor is te vinden in het document keyboard.pdf. In de procedure `send_byte` wordt dit protocol uitgevoerd. De te versturen byte (`keycode`) is input. Vervolgens wordt in de procedure het odd parity bit bepaald (waar wij verder niets mee zullen doen) en ook start en stop bit worden toegevoegd. Vervolgens wordt in een loop deze data serieel op de uitgang gezet.
- Instantiatie van `showkey`. Hier valt weinig over te vertellen.
- In een proces worden drie bytes verzonden door het 'toetsenbord'. Na voltooiing van de `send_byte` moet, als het goed is, de correcte waarde op de uitgang van `showkey` staan. Met het ASSERT statement worden de beide uitgangen dig0 en dig1 vergeleken met de verwachte waarde. Indien dit afwijkt wordt de melding "expected .." weergegeven.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY showkey_simple_test IS
END showkey_simple_test;

ARCHITECTURE test OF showkey_simple_test IS
  SIGNAL reset : std_logic := '0';
  SIGNAL kbclock : std_logic := '1';
  SIGNAL kbdata : std_logic := '0';
  SIGNAL dig0, dig1: std_logic_vector(7 DOWNTO 0);
  SIGNAL scancode : std_logic_vector(7 DOWNTO 0);
  SIGNAL byte_read : std_logic;

  FUNCTION hex2display (n:std_logic_vector(3 DOWNTO 0)) RETURN std_logic_vector IS
  BEGIN
    CASE n IS  --       hgfedcba
          WHEN "0000" => RETURN "00111111";
          WHEN "0001" => RETURN "00000110";
          WHEN "0010" => RETURN "01011011";
          WHEN "0011" => RETURN "01001111";
          WHEN "0100" => RETURN "01100110";
          WHEN "0101" => RETURN "01101101";
          WHEN "0110" => RETURN "01111101";
          WHEN "0111" => RETURN "00000111";
          WHEN "1000" => RETURN "01111111";
          WHEN "1001" => RETURN "01101111";
          WHEN "1010" => RETURN "01110111";
          WHEN "1011" => RETURN "01111100";
          WHEN "1100" => RETURN "00111001";
          WHEN "1101" => RETURN "01011110";
          WHEN "1110" => RETURN "01111001";
          WHEN OTHERS => RETURN "01110001";
    END CASE;
  END hex2display;

  PROCEDURE send_byte (
      CONSTANT byte    : IN std_logic_vector(7 DOWNTO 0);
      SIGNAL kbclock   : OUT std_logic;
      SIGNAL kbdata    : OUT std_logic)
  IS
    VARIABLE odd_parity : std_logic;
    VARIABLE data     : std_logic_vector(10 DOWNTO 0);
    CONSTANT half_period_kbclock : time := 18 us; -- kbclock ~ 27 KHz
  BEGIN
    -- parity
    odd_parity:='1'; -- needed in the next loop to generate ODD parity
    FOR i IN 7 DOWNTO 0 LOOP
      odd_parity := odd_parity XOR byte(i);
    END LOOP;
    data := '1' & odd_parity & byte & '0';
    -- send data
    FOR i IN 0 TO 10 LOOP
      kbdata <= data(i);
      kbclock <= '1';
      WAIT FOR half_period_kbclock;
      kbclock <= '0';
      WAIT FOR half_period_kbclock;
    END LOOP;
    kbclock <= '1';
  END send_byte;

BEGIN
  sk:ENTITY work.showkey PORT MAP (
    reset     => reset,
    kbclock   => kbclock,
    kbdata    => kbdata,
    dig0      => dig0,
    dig1      => dig1,
    scancode  => scancode,
    byte_read => byte_read
  );

  PROCESS
  BEGIN
```

```
    reset <='0';
    WAIT FOR 300 us;
    reset <= '1';

    -- key A: 1C (hex) => 0001 1100 of hexadecimaal X"1C"
    send_byte(X"1C", kbclock, kbdata);
    ASSERT (dig0=hex2display(X"1") AND dig1=hex2display(X"C")) REPORT "expected: 1C (hex)"
           SEVERITY error;
    WAIT FOR 300 us;

    -- key B: 32 (hex) => 0011 0010
    send_byte(X"32", kbclock, kbdata);
    ASSERT (dig0=hex2display(X"3") AND dig1=hex2display(X"2")) REPORT "expected: 32 (hex)"
           SEVERITY error;
    WAIT FOR 300 us;

    -- key P: 4D (hex) => 0011 0010
    send_byte(X"4D", kbclock, kbdata);
    ASSERT (dig0=hex2display(X"4") AND dig1=hex2display(X"D")) REPORT "expected: 4D (hex)"
           SEVERITY error;
    WAIT FOR 300 us;
    WAIT;
  END PROCESS;

END test;
```

Figuur 5.9 Testen van SHOWKEY



Figuur 5.10 Resultaat van een simulatierun

Opdracht 2: Test je ontwerp met de bijgeleverde testomgeving (figuur 5.9)

- Compileer je eigen `showkey` ontwerp. Zorg er daarbij voor dat je de entity beschrijving niet wijzigt.
- Compileer daarna de file met de testomgeving: `showkey_simple_test.vhd`
- Selecteer `showkey_simple_test` om te simuleren, en vervolgens:
  - add wave *
  - run -all

In figuur 5.10 is een simulatierun gegeven van een test van `showkey` van het ontwerp van de docent. Belanrijk is het dat `dig0` en `dig1` correcte waarden hebben als `byte_read` is 1. Als `byte_read` is 0 kan `dig0` en `dig1` afwijken van de hier gegeven simulatierun.

5.4.2 Deelopdracht Realisatie Showkey

Als je je hebt gehouden aan de voorwaarden voor synthese dan moet het ontwerp redelijk gemakkelijk gesynthetiseerd kunnen worden.

5.4.2.1Synthese met Quartus II.

Synthetiseer je ontwerp m.b.v. Quartus II.
Gebruik onderstaande gegevens voor de technologie:
- device family: Altera➔Cyclone
- device: EP1C12F324C8

Voor het uitvoeren van een postsimulatie moet je ook, in stap 4 van de New Project Wizard, aangeven dat je voor Simulation gebruik wilt maken van “ModelSim-Altera”

N.B.: Vergeet niet om de constraintfile (`showkey.qsf`) in dezelfde directory te plaatsen waarin ook `showkey.vhd` staat. De constraintfile, `showkey.qsf`, wordt door Quartus aangepast. Wellicht goed om ook een origineel te bewaren.

5.4.2.2Uitvoeren van een postsimulatie

In de vorige sectie is aangegeven dat voor simulatie gebruik wordt gemaakt van ModelSim-Altera. Na synthese is er dan ook de directory ..\simulation\modelsim aanwezig met de gegenereerde vhdl file (*.vho) en de daarbij behorende delay file (*.sdo). We kunnen nu dezelfde testomgeving (file `showkey_simple_test.vhd`) gebruiken voor de postsimulatie (zie ook VHDL tutorial). Merk op dat het `showkey` dus niet het topniveau is. Daarom moet je bij de postsimulatie ook de *region* aangeven zodat de delay file (*.sdo) correct wordt gekoppeld. Daarvoor wordt de label (SK) van de componentinstantiatie gebruikt. Dus:

Opdracht 1: Simuleer het gegenereerde ontwerp met de testomgeving (postsimulatie)

N.B. 1: De gegenereerde VHDL file is een beschrijving van de hardware zoals dat is gerealiseerd. In de port declaratie van de gegenereerde entity is dan ook gebruik gemaakt van de types std_logic en std_logic_vector. D.w.z. dat als in de port declaratie VHDL beschrijving van de specificatie staat:
`inp : IN integer range 0 to 3`
Dan zal in de gegenereerde VHDL beschrijving staan:
`inp : IN std_logic_vector(1 downto 0)`
M.a.w. de testomgeving moet soms iets worden aangepast zodat de typering overeenkomt.

N.B. 2: Postsimulatie wordt altijd uitgevoerd als er een ASIC wordt gemaakt. Bij een postsimulatie wordt o.a. de timing gecontroleerd (een spike op de kloklijn, de data op de data-ingang is niet stabiel bij de actieve flank van de klok van een flipflop, etc.). Het is een (financiele) ramp als dit pas wordt opgemerkt nadat de ASIC is gemaakt. Bij een FPGA wordt de postsimulatie wel eens overgeslagen omdat je vrij snel de FPGA kunt programmeren en dan kunt vaststellen of het werkt. Maar werkt het dan echt? De omgevingstemperatuur is bijvoorbeeld 21 ^{0}C maar bij de klant misschien 18 ^{0}C. In het college over timing komen we hierop terug.

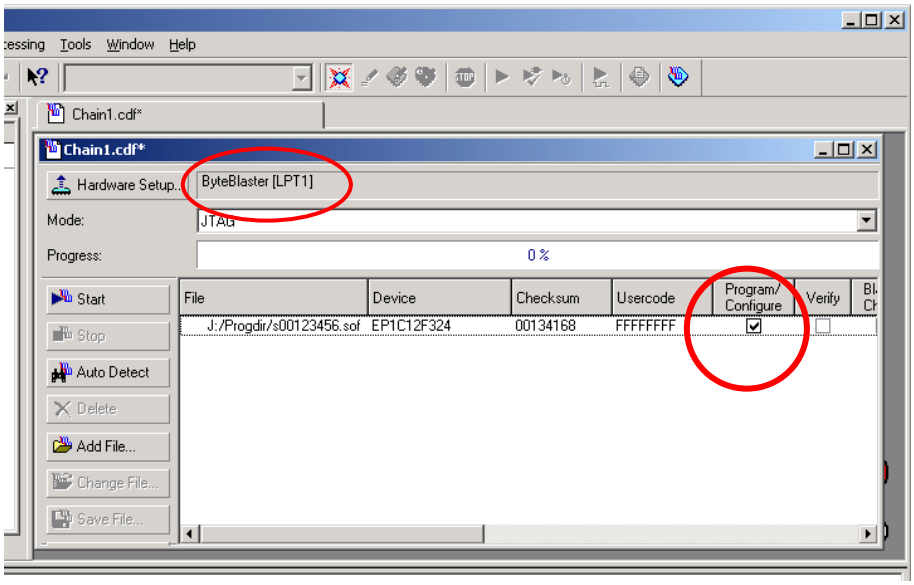
5.4.2.3Transport van de programmeerfile naar programmeerpc's.

Er is o.a. een programmeerfile `showkey.sof` gecreëerd.
Hernoem deze file in je studentnummer, dus bijv: **s0123456.sof**.
Kopieer deze file vervolgens naar `J:\progdir`

5.4.2.4 Ga vervolgens naar een van de programmeerpc's.

Waarschijnlijk staan de systemen al klaar om gebruikt te worden; d.w.z.
- quartus II is opgestart en staat in programmeermode
- livedesign bordje is aangesloten en daarop is een PS/2 toetsenbord aangesloten.

Waarschijnlijk staat Quartus II al gereed om gebruik te worden. Mocht dit niet het geval zijn dan:
- Start QuartusII.
- Ga naar tools➔programmer



Figuur 5.11:Instellingen voor het programmeren van het LiveDesign bordje

- Controleer dat er bij de hardware setup staat “byteblaster(LPT1)” of “byteblasterMV(lpt1)”
  - Indien er staat “no hardware” dan op “hardware setup”
  - In het window dat dan wordt geopend dubbelklikken op ByteblasterMV
  - Window weer afsluiten. Nu moet de correcte hardware zijn vermeld.
- Add file→ en selecteer de programmeer `J:\progdir\s0123456.sof`
- Aanvinken van “program/Configure” (zie figuur 5.11).
- vervolgens start.
- Vervolgens wordt het livedesignbord geprogrammeerd.

N.B.: Bij het programmeren krijg je soms een foutmelding m.b.t. JTAG. Dit is op te lossen door de voedingsspanning van het livedesign bordje af te halen en vervolgens weer aan te sluiten. Daarna lukt het programmeren wel.

Werkt showkey?

- **Laat de correcte werking zien en aftekenen door de assistent.**
- Als het orgeltje niet werkt dan op de eigen werkplek het probleem herstellen!
- Verwijder je eigen programmeerfile van de directory `J:\progdir`.

5.4.3 Deelopdracht Constantkey

In paragraaf 5.1.3.2 is het gedrag van deze schakeling al beschreven.
De entity beschrijving is:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY constantkey IS
  PORT (
    reset      : IN std_logic;
    clk        : IN std_logic; -- 50MHz clock
    scancode   : IN std_logic_vector(7 DOWNTO 0);
    byte_read  : IN std_logic;
    dig2, dig3 : OUT std_logic_vector(7 DOWNTO 0);
                 -- show key pressed on display dig2 en dig3 (resp high & low).
    key        : OUT std_logic_vector(7 DOWNTO 0)
    );
END constantkey;
```

Als het `byte_read` signaal hoog wordt dan heeft `readkey` een byte ontvangen. De daarbij behorende code staat dan op de ingang `scancode`. Nu doet zich hier een probleem voor waarover later in het college nog wordt ingegaan. `Readkey` is een synchroon systeem dat werkt op de klok gegenereerd door het toetsenbord (~20kHz). `Constantkey` is ook een synchroon systeem, maar werkt op de 50 MHz klok van het

experimenteerbordje. Het kan dus voorkomen dat het `byte_read` verandert op het moment dat er een actieve flank van de 50MHz klok is. In het college over "timing" komen we hier op terug. Belangrijk is het dat je het ingangssignaal `byte_read` eerst door twee in serie geschakelde flipflops laat gaan waarbij de klokingang is aangesloten op de 50 MHz klok. Het uitgangssignaal van de tweede flipflop is dan een 'gesynchroniseerd' ingangssignaal en mag je verder gebruiken.

Opdracht 1: Zoals in paragraaf 5.1.3.2 is beschreven willen we het moment waarop `byte_read` van laag naar hoog gaat gebruiken om vast te stellen dat er een nieuwe scancode op de ingang staat. Ontwerp een systeem dat een output '1' maakt gedurende 1 klokperiode van de 50 MHz klok als `byte_read` van laag naar hoog gaat. Noem dit signaal `new_scancode_detected`.

N.B.: schrijf wel de VHDL code voor dit onderdeel. We zullen dit echter direct opnemen bij opdracht 2. Dus het mag eerst ook nog in het klad (in je journaal uiteraard!!!)

In paragraaf 5.1.3 is aangegeven hoe het verloop van het indrukken van een toets (en het eventueel ingedrukt houden ervan) t/m het loslaten ervan is. In dit subsysteem willen we zolang een toets is ingedrukt de daarbij behorende scancode constant op de uitgang hebben staan.

Tip: alleen als `new_scancode_detected` actief is (opdracht 1) kan er iets zijn veranderd.

Opdracht 2: Teken een toestandsmachine waarbij als input `scancode` en `new_scancode_detected` wordt gebruikt. Het uitgangssignaal is 00_{16} of de scancode van de toets die is ingedrukt.

N.B. 1: Als je het systeem helemaal als toestandsmachine zou gaan beschrijven heb je waarschijnlijk veel toestanden nodig: scancode heeft 2^8 mogelijkheden. Realiseer je dat het indrukken (en evt. ingedrukt houden) t/m het loslaten een patroon heeft dat onafhankelijk is van de toets die is ingedrukt. Je kunt dus met veel minder toestanden volstaan (De docent kon volstaan met 3 toestanden). Daarnaast is nog wel een register nodig waarin de scancode wordt opgeslagen.

N.B. 2: bij het loslaten van de toets worden twee bytes verstuurd F0 gevolgd door de keycode. Als je het orgeltje goed gebruikt kan dit alleen maar de keycode zijn van de toets die is ingedrukt. Als je 2 of meer toetsen tegelijk indrukt is de keycode die volgt op F0 lastig vast te stellen. Als toch per ongeluk twee toetsen worden ingedrukt moet het systeem zich snel weer normaal gedragen. Dit kun je o.a. bereiken door niet de inhoud van de keycode na de byte F0 te controleren.

Laat de assistent de oplossing zien.

Opdracht 3: Geeft een VHDL beschrijving van de architecture van `constantkey`. Hierin bevindt zich dus de generatie van `new_scancode_detected` (opdracht 1) en de toestandsmachine (opdracht 2).

Opdracht 4: Test deze schakeling met de testomgeving die te vinden is in file:
`constant_key_simple_test.vhd`

Opdracht 5: Synthetiseer je ontwerp. Dit onderdeel wordt nog niet gerealiseerd op het experimenteerbordje. Toch is het wel verstandig om `constantkey` te synthetiseren met Quartus en daarmee te controleren of het ontwerp synthetiseerbaar is.

5.4.4 Deelopdracht Readkey

In paragraaf 5.1.3 is al uitvoerig het subsysteem `readkey` besproken. De entity beschrijving is gegeven:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY readkey IS
  PORT (
    clk        : IN std_logic; -- high freq. clock (~ 50 MHz)
    reset      : IN std_logic;
    kbdata     : IN STD_LOGIC; -- low freq. clk (~ 20 kHz) serial data from the keyboard
    kbclock    : IN STD_LOGIC; -- clock from the keyboard
    key        : OUT std_logic_vector(7 DOWNTO 0);
                 -- I/O check via 7-segment displays
    dig0, dig1 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                 -- show key pressed on display
    dig2, dig3 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
                 -- show key pressed on display; after processed by constant key
   );
END readkey;
```

Opdracht 1: Beschrijf de architecture behorend bij entity `readkey`

Opdracht 2: Gebruik de testomgeving uit file `readkey_simple_test.vhd` om het ontwerp te testen

Opdracht 3: synthetiseer het ontwerp en realiseer dit op het experimenteerbordje:

N.B. 1: Als je een toets ingedrukt houdt zul je zien dat `dig0` en `dig1` gaan knipperen. De displays `dig2` en `dig3` mogen niet knipperen.

N.B. 2: Vergeet niet om de constraintfile toe te voegen!

Laat de correcte werking zien en aftekenen door de assistent.

5.4.5 Deelopdracht key2pulselength

In deze opdracht wordt de mapping van de relevante toetsen van het orgel naar de pulsduur geregeld. Bij elke toets van figuur 5.2 hoort een constante waarde die op de uitgang van het systeem komt.
Nu wordt het dus tijd om de theorie over de toonladder in de praktijk te brengen!
Als een ongeldige toets wordt ingedrukt, die dus niet op het orgeltje voorkomt, komt er de integerwaarde 0 op de uitgang. Wanneer de uitgang van `key` van `key2pulselength` 0 is zal het er op aangesloten systeem geen toon genereren.

N.B.: In paragraaf 5.1.4 is aangegeven dat `key` ook direct is verbonden met het nog te ontwerpen systeem `mul_power_of_2`. In `mul_power_of_2` wordt ingang `key` gebruikt voor de octaafverhoging/octaafverlaging. In het nu te ontwerpen systeem, `key2pulselength`, is uitgang `pulse_length` 0 wanneer ‘A’ of ‘Z’ is ingedrukt.

De entity beschrijving is gegeven:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY key2pulselength IS
  GENERIC (max_length : integer := 20000);
  PORT (key    : IN std_logic_vector(7 DOWNTO 0);
        pulse_length : OUT INTEGER RANGE 0 TO max_length
       );
END key2pulselength;
```

Opdracht 1: Geef een architecture beschrijving van `key2pulselength`.
Zorg er voor dat het leesbaar blijft! Declareer constanten, bijvoorbeeld
CONSTANT key_Tab : std_logic_vector := X"0D";

Opdracht 2: Test deze schakeling. Dit is op zich geen complex ontwerp. Een echte testomgeving is hier dan ook niet voor gemaakt. Er kan volstaan worden met enkele commando’s in het transcript window. Uiteindelijk verwachten we hier een scriptfile met daarin de te geven commando’s.
Neem ook een waveform op in het journaal.

Opdracht 3: Synthetiseer je ontwerp. Dit onderdeel wordt nog niet gerealiseerd op het experimenteerbordje. Toch is dit wel handig om na te gaan of dit deelsysteem door synthese komt.

5.4.6 Deelopdracht vermenigvuldig met een macht van twee

In paragraaf 5.1.4 is globaal de toongeneratie beschreven. Met de toetsen 'A' en 'Z' wordt de octaafverandering gerealiseerd. Een octaaf hoger/lager betekent dat de frequentie wordt verdubbeld respectievelijk gehalveerd.

N.B. 1: Bij de reset kies je ergens een frequentie in het midden van de mogelijke frequenties.

N.B. 2: Denk er aan dat de keycode gedurende het indrukken als constante input op de ingang `key` staat. Dus als je geen bijzondere maatregelen treft zal bij één keer indrukken van de 'A' met een 50MHz klok dit systeem heel vaak een 'A' op de ingang zien staan en dus niet één maar vele octaven omhoog zal gaan.

De entity beschrijving is gegeven:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY mul_power_of_2 IS
  GENERIC (max_length : integer := 20000);
  PORT ( clk                    : IN std_logic;
         reset                  : IN std_logic;
         key                    : IN std_logic_vector(7 DOWNTO 0);
         pulse_length           : IN integer RANGE 0 TO max_length;
         pulse_length_multiplied : OUT integer RANGE 0 TO max_length*32
             );
END mul_power_of_2;
```

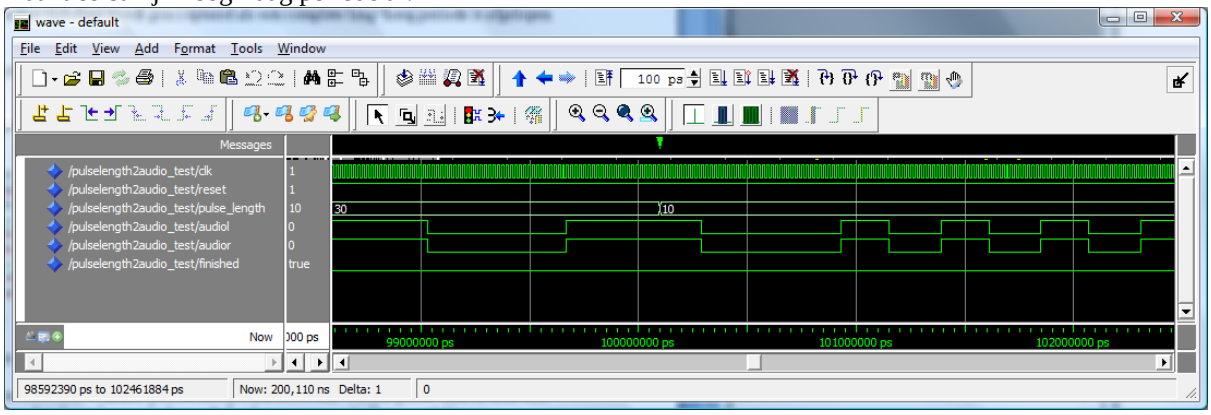
Opdracht 1: Geef een architecture beschrijving van de `mul_power_of_2`.

Opdracht 2: Maak een testbench waarin de combinatie van de `key2pulselength` schakeling met `mul_power_of_2` schakeling wordt getest.

Opdracht 3: Synthetiseer je ontwerp. Dit onderdeel wordt nog niet gerealiseerd op het experimenteerbordje. Toch is dit wel handig om na te gaan of dit deelsysteem door synthese komt.

5.4.7 Deelopdracht pulselength2audio

Deze schakeling krijgt een constante `pulse_length` en zal de beide audio uitgangen periodiek gedurende `pulse_length` × clk_periode hoog maken en dezelfde tijdsduur laag maken.
Zorg er voor dat pas overgegaan wordt op een andere tijdsduur als de complete hoog+laag periode is voltooid. Dit is ook zichtbaar in figuur 5.12: Pulse_length verandert van 30 naar 10 maar het gegenereerde audio signaal maakt eerst zijn hoog+laag periode af.



Figuur 5.12: simulatierun van `pulselenth2audio`

N.B.: Als op de ingang `pulselength` de waarde 0 staat zal er geen toon worden gegenereerd; de uitgang is dan constant 0.

De entity beschrijving is gegeven:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY pulselength2audio IS
  GENERIC (max_length : integer := 20000);
  PORT  (clk    : IN std_logic;
         reset  : IN std_logic;
         pulse_length : IN INTEGER RANGE 0 TO max_length;
         audiol : OUT std_logic;
         audior : OUT std_logic
  );
END pulselength2audio;
```

Opdracht 1: Geef een architecture beschrijving van de `pulselength2audio`.

Opdracht 2: Maak een testbench waarmee `pulselength2audio` wordt getest. Laat in deze test ook zien dat de complete hoog+laag periode wordt doorlopen wanneer input `pulse_length` van waarde verandert (neem een relevant deel van de waveform op in je journaal).

Opdracht 3: Synthetiseer je ontwerp. Dit onderdeel wordt nog niet gerealiseerd op het experimenteerbordje. Toch is dit wel handig om na te gaan of dit deelsysteem door synthese komt.

5.4.8 Deelopdracht tone generation

In paragraaf 5.1.4 is de toongeneratie behandeld. De entity beschrijving is gegeven:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY tone_generation IS
  PORT (clk    : IN std_logic;
        reset  : IN std_logic;
        key    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        audiol : OUT std_logic;
        audior : OUT std_logic);
END ENTITY tone_generation;
```

Opdracht 1: Geef een architecture beschrijving van de `tone_generation`.

Opdracht 2: Maak een testbench voor `tone_generation`. Merk op dat een simulatie (erg) lang kan duren dus maak een niet te omvangrijke test. Neem ook een waveform op in het journaal.

Opdracht 3: Synthetiseer je ontwerp. Dit onderdeel wordt nog niet gerealiseerd op het experimenteerbordje. Toch is dit wel handig om na te gaan of dit deelsysteem door synthese komt.

5.4.9 Deelopdracht orgel

Het orgel bestaat uit de subsystemen `readkey` en `tone_generation`; zie ook paragraaf 5.1.2.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY organ IS
  PORT (clk  : IN std_logic;
        reset  : IN std_logic;
        kbdata   : IN STD_LOGIC; -- low freq. clk (~ 20 kHz) serial data from the keyboard
        kbclock  : IN STD_LOGIC; -- clock from the keyboard
        audiol : OUT std_logic;
        audior : OUT std_logic;
               -- debug outputs
        dig0, dig1: OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- show key pressed on display
        dig2, dig3: OUT STD_LOGIC_VECTOR(7 DOWNTO 0) -- show key pressed on display
);
END ENTITY organ;
```

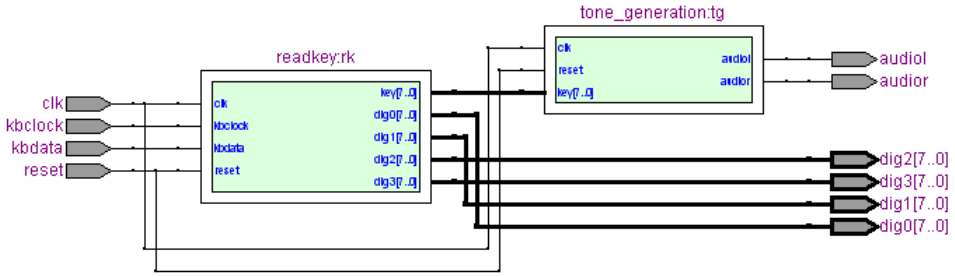
Opdracht 1: Beschrijf de architecture behorend bij entity `organ`

Opdracht 2: Synthetiseer het ontwerp en realiseer dit op het experimenteerbordje.

Opdracht 3: Toon je muzikale kwaliteiten!

Laat de correcte werking zien en aftekenen door de assistent.

6 Eindopdracht: orgel met opname mogelijkheid



Figuur 5.13: Opdeling van het orgel in 2 subsystemen

In de tussenopdracht is een orgel gemaakt bestaande uit twee subsystemen `readkey` en `tone_generation` (fig. 5.13). Bij het tot stand komen van het orgel waren de verschillende deelproblemen gegeven. Bij de eindopdracht wordt dat niet meer gedaan.

In de eindopdracht moet een opname/weergave mogelijkheid aan het orgel worden toegevoegd met de volgende randvoorwaarden:

1. De bestaande subsystemen `readkey` en `tone_generation` mogen niet worden gewijzigd.
2. Voor de opslag van de ingedrukte toetsen en de tijdsduur van het indrukken moet gebruik worden gemaakt van het SRAM geheugen dat aanwezig is op het livedesignbord.
3. De opname/weegave eenheid heeft een drietal toetsen (gebruik hiervoor de knoppen op het livedesignbord)
    a. Stop; opname of weergave beeindigen.
    b. Rec; opnemen van wat op dat moment wordt gespeeld (dat moet ook te horen zijn tijdens de opname).
    c. Play; afspelen van wat is opgenomen

In de eindopdracht komen in ieder geval de volgende onderdelen voor:

- Uw eigen specificatie waaraan deze opname/weergave module moet voldoen (in ieder geval rekening houdend met bovenstaande randvoorwaarden). Motiveer hierin ook uw keuze voor de resolutie m.b.t. het bemonsteren van `readkey`. Verder is gegeven de organist vele uren onafgebroken kan spelen.
- De opname/weergave module moet afzonderlijk getest worden m.b.v. een VHDL testbench.
    - Voor dit onderdeel moet er ook een simulatiemodel van een SRAM zijn. De datasheet van de SRAM is gegeven. Tevens is een verre van ideaal VHDL model beschikbaar op Blackboard.
        - Een simulatie kan erg lang duren immers de klokfrequentie is 50 MHz en het indrukken van een toets duurt bijvoorbeeld 1 sec. In de testbench de toets dus niet te lang indrukken. (Wellicht kan het principe ook sneller worden gesimuleerd door de ‘bemonsteringstijd’ te schalen.)
- Realiseer uw ontwerp met het livedesignbord. De entity `organ_with_recorder.vhd` en constraintfile `organ_with_recorder.qsf` zijn gegeven (Blackboard).
    - Indien de realisatie niet werkt dan voor het hele systeem een VHDL testbench maken (hierbij weer het SRAM VHDL model gebruiken). Voor de VHDL testbench kun je je laten inspireren door `readkey_simple_test`.

Laat de correcte werking zien en aftekenen door de assistent.

7 Afronding practicum.

Laat de studentassistent de correcte werking van het orgel zien.
Lever ook het journaal in, vermeld op het voorblad:

- naam (met voorletters)
- adres
- studentnummer

Bekende tool problemen

7.1 Modelsim

7.1.1 Tijdens simulatie al (enkele) op enkele voorwaarden voor synthese controleren?

Een veel voorkomende fout is dat bij het beschrijven van een combinatorische schakeling d.m.v. een process niet alle signalen die worden gelezen in dat proces in de sensitivitylist staan (lijst achter keyword process). In ModelSim kun je hier al op controleren. Ga naar “compiler Options” en vink de optie “check for synthesis” aan.

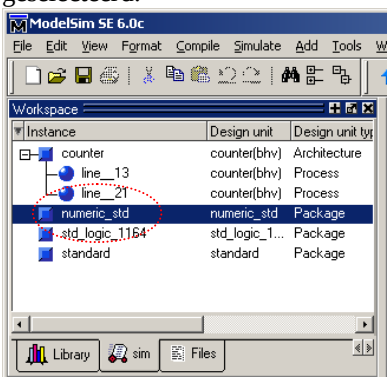
QuestaSim/ModelSim gebruikt de modelsim.ini file om bovenstaande instelling op te slaan. Deze file mag niet door de student worden overschreven. Wil je wel de instellingen onthouden maak dan een kopie van de file modelsim.ini (C:/Program Files/questasim_6.5b) en plaats deze in de directory waarin ook je ontwerp staat. QuestaSim zal dan deze locale modelsim.ini gebruiken.

Een alternatief is om bij compileren de optie –check_synthesis mee te geven:

```
vcom -check_synthesis <filenaam>
```

7.1.2 Geen signalen in de waveform?

Na het uitvoeren van het commando “add wave *” worden de signalen niet in het wave window getoond. Waarschijnlijk is in het linker window niet het ontwerp (in dit voorbeeld counter) maar een package geselecteerd.



7.2 Quartus II

Tijdens het synthetiseren van een ontwerp worden een groot aantal files gegenereerd. Deze files komen in dezelfde directory waar ook het ontwerp in staat. Let er daarbij op de constraintfile (*.qsf) wordt aangepast. Als je bijvoorbeeld een verkeerd device hebt geselecteerd komt dat device ook in de constraintfile. Bewaar een kopie van deze file op een andere plek.

7.3 Programmer

7.3.1 Byteblaster

Wellicht moet je de eerste keer nog vertellen dat er gebruik wordt gemaakt van de “byteblasterMV (LPT1)” met JTAG interface. Default heeft de programmer (Tools=>Programmer “No hardware”).
Klik dan op “Hardware setup” en selecteer ByteblasterMV.

7.3.2 JTAG

Soms geeft de programmer een foutmelding m.b.t. JTAG. Dit is eenvoudig op te lossen door de spanning even van het livedesignbordje te halen.