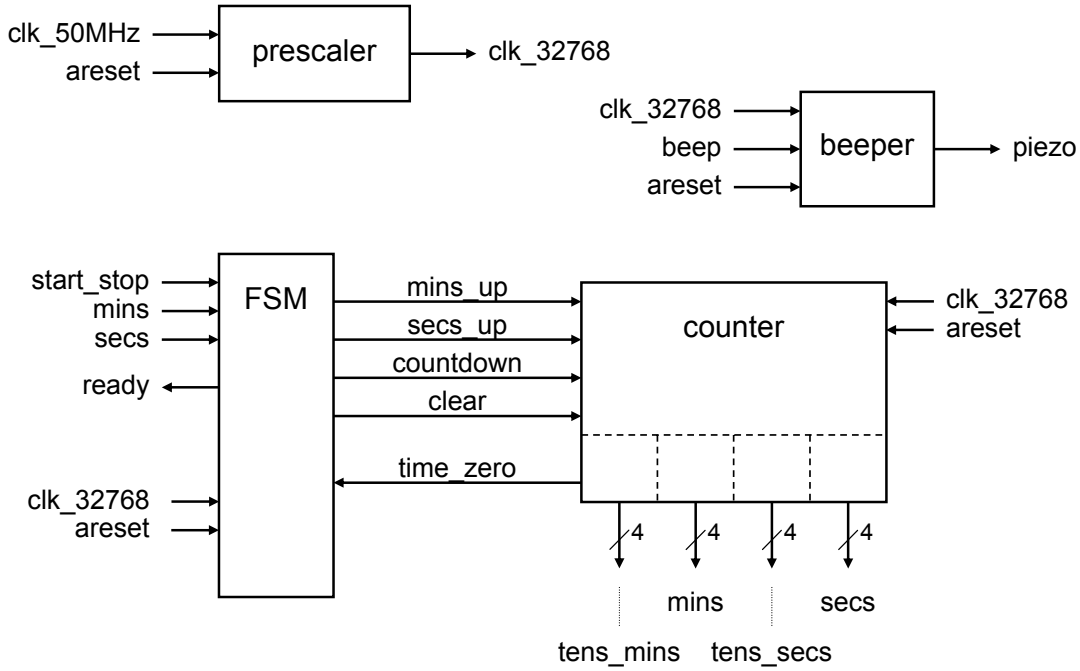
Opdracht week 3 en 4 – kookwekker

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 1.

Figuur 1: *Twee voorbeelden van een kookwekker*

Deze opdracht bestaat uit het ontwerpen, implementeren en beproeven van een kookwekker. In figuur 2 is het blokschema gegeven.



Figuur 2: *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!

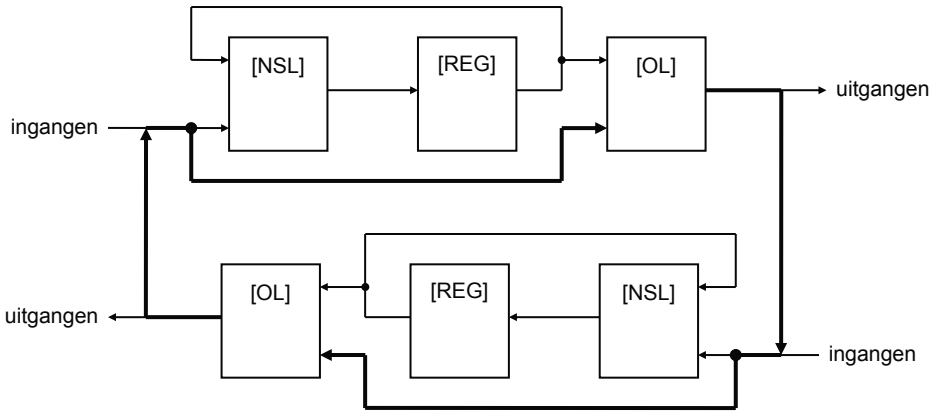
Opmerkingen

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.

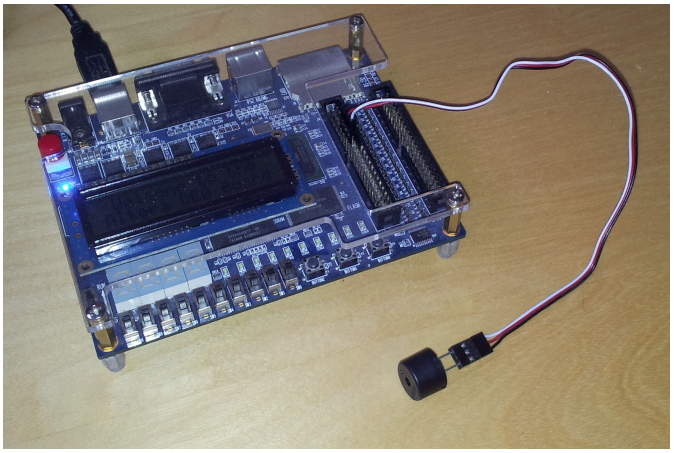
Verder zijn er nog wat punten om mee te nemen in het ontwerp:

- 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 2.
- Let op dat de FSM en de teller twee *rondgekoppelde* toestandsmachines zijn die last kunnen hebben van asynchrone lusvorming. Zie figuur 3.

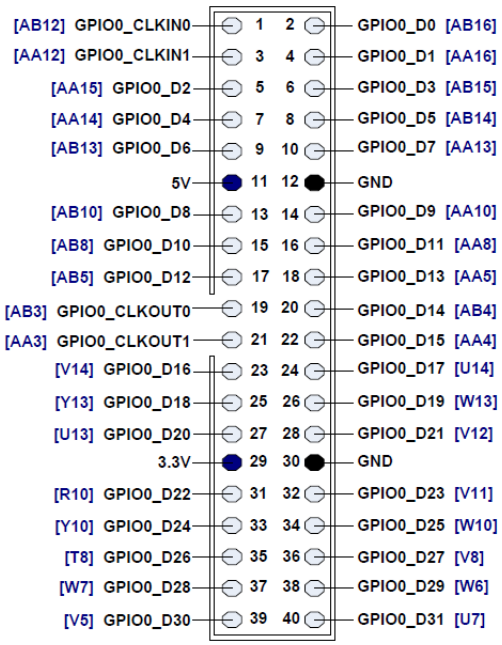


Figuur 3: *Rondgekoppelde Mealy-machines met mogelijk een combinatorische lus.*

Tussen de uitgang `piezo` en de ground wordt een piezo-buzzer geplaatst zodat een pieptoon van 2 kHz te horen is. Zie figuur 4. 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 5.



Figuur 4: *Aansluiting van de piezo-buzzer op een DE0-bordje.*



Figuur 5: *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*