DSD: Project

Flappybird

Opleiding: Digital Systems Development

Academiejaar: 2023-2024

Gil Struyf

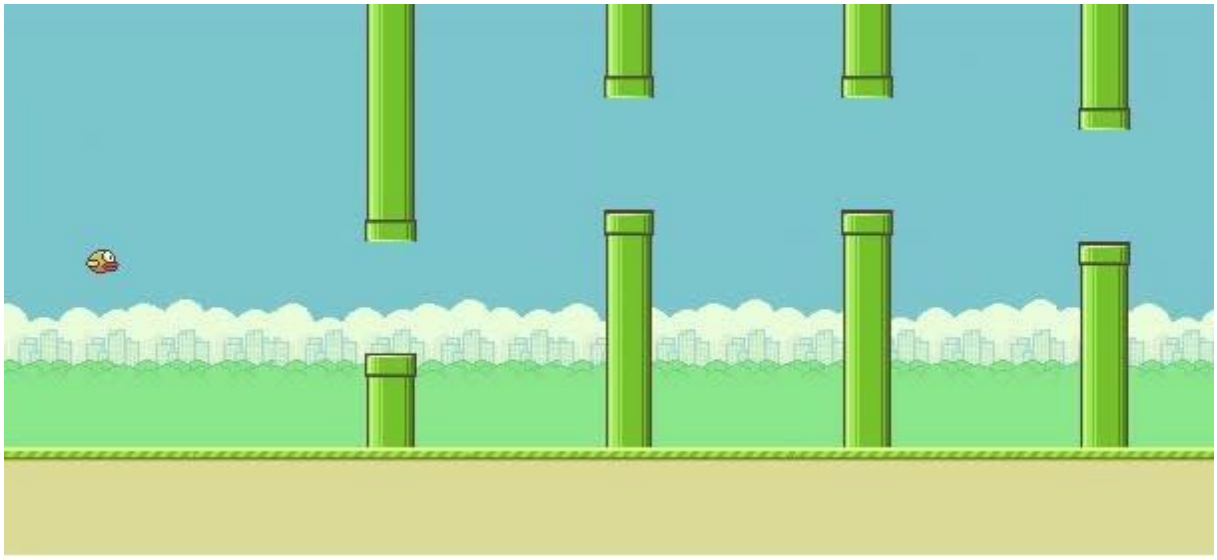
Inhoud

Inhoud

Flappy bird project:

1 Doel

Ik heb gekozen voor een “look a like” te bouwen van het zeer bekende mobiel spel flappy bird, hierbij is het de bedoeling om een vogeltje door verschillende pijpen (de obstakels) te laten vliegen. De bedoeling is eigenlijk om het zolang mogelijk vol te houden en het meeste punten te verzamelen, dit betekend dus dat dit tot het oneindig door zal gaan en geen mogelijkheid is om het dus uit te spelen.



Figuur 1 https://news.yahoo.com/die-another-day-flappy-bird-175619065.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAL4TltPCMbblzc7Qeh7SkFyoV28NQUUXDVzLgK2-mReC0lRLOBtGiC873Mu7rGT2hzP2ZYozu9QXsdG6F81bcRiPNj9z7gzj3ZZCr1kltXG

2 Functionaliteiten:

1. Mogelijkheid om score te bekijken op de 7segment display
2. Mogelijkheid om highscore te bekijken op de 7segment display (tussen beide kan je wisselen doormiddel van een schakelaar)
3. Mogelijkheid om een blokje (vogel) omhoog en naar beneden te bewegen op verschillende manieren
    - Via een button op een fpga
    - Via een keyboard klik om een aangesloten toestenbord
    - Via een geluidssensor

- Mogelijkheid om de pipe’s (obstakels) in het oneindigen te genereren.
- Mogelijkheid om detectie’s waar te nemen:
- Om de score op te tellen (als deze door de pipe is geweest)
- Om speler te laten dood gaan (als deze tegen de pipe of border is gebotst)
- Mogelijkheid als je de input trigger dat deze voor een bepaalde duur omhoog zal gaan.
- Manier om de pipe’s (obstakels) op random locaties te zetten (elk spel anders)
- Mogelijkheid om moeilijkheidsgraad in te stellen (afstand tussen pipes groter/kleiner maken)

3 Hardware

Nodige hardware om volledig project te kunnen uitvoeren.

basys3 artix-7 fpga board
VGA to VGA kabel
Monitor
Microfoonvolumesensor
Jumper kabels

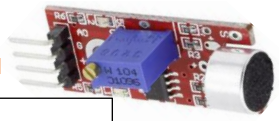
3.1 Aansluiting

Monitor:

VGA

Basys3 artix7 board:

Microfoonvolumesensor:

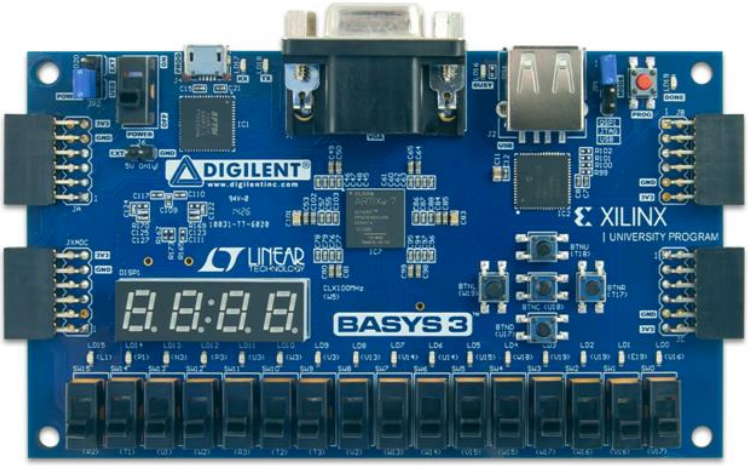


Jumper kabels

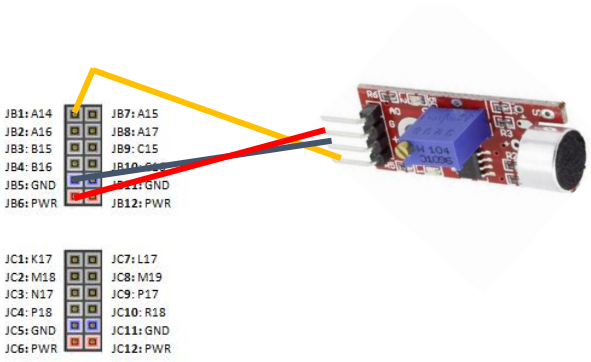
3.2 Aansluit diagram

Basys3 artix7 board:	Microfoonvolumesensor:
JB1	DO
JB5 of JB11	GND (-)
JB6 of JB12	5V (+)

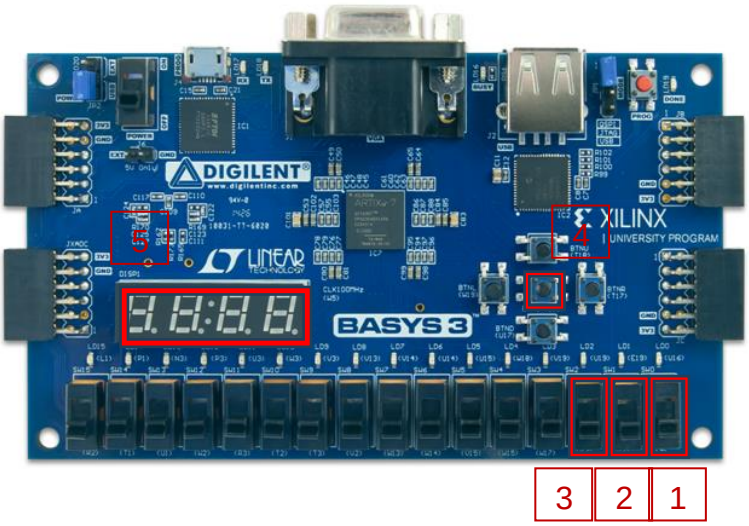
Basys3 artix7 board:



Microfoonvolumesensor:



3.3 Gebruikers instructies



1. Met deze switch stel je in welke input modus je zal gebruiken om het spel te spelen. keuze uit: knop (nr: 4) of sensor
2. Met deze switch kan scoreboard wissel tussen “high score” of “current score”
3. Met deze switch kan je de moeilijkheidsgraad aanpassen, je kan de afstand tussen de pipes vergroten of verkleiner, hierdoor kan je spel eenvoudiger maken.
4. Met deze button kan de speler doen springen (als deze is ingesteld met switch nr: 1)
5. Op de 3 laatste digits kan je de “high score” of “momentele score” aflezen (instelbaar met switch nr: 2).

4 Werking

4.1 Speler

De speler blijft altijd centraal op het scherm en is enkel beweegbaar op de Y-as, maar om toch het effect te geven dat deze vooruit gaat bewegen we de pipe's (obstakels) richting de speler.

4.1.1 input

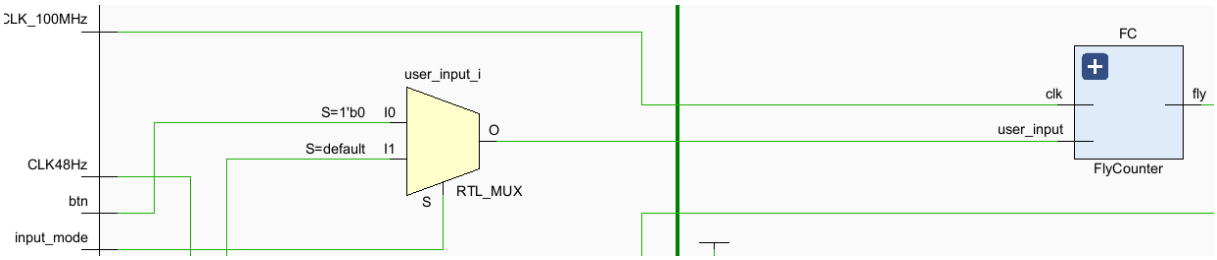
De speler bewegen kan op 2 manieren. Deze wordt gekozen via een switch op de fpga:

- Input_mode = 0 = PCB button
- Input_mode = 1 = Microfoonvolumesensor

Fly.vhd:

```
-- select the right input
process(input_mode)
begin
  if input_mode = '0' then
      user_input <= btn; -- Use sensor as input
  else
      user_input <= sensor; -- Use button as input
  end if;
end process;
```

Elaborated design:



4.1.2

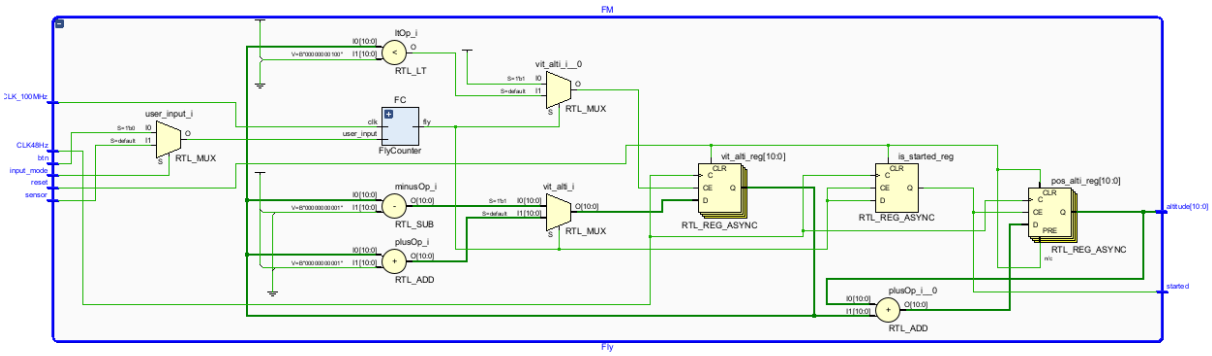
Beweging

De zwaartekracht en beweging van de speler wordt uitgevoerd met een kloksnelheid van 48Hz. En zal afhankelijk van de “fly counter” de speler omhoog of omlaag doen bewegen. Als het spel gestart is zal deze zwaartekracht simuleren en de speler laten vallen.

Fly.vhd:

```
 btnActive: process(CLK48Hz, reset)
  -- Beweeg speler
  btnActive: process(CLK48Hz, reset)
  begin
    if reset='1' then
      pos_alti <= pos_alti_def;
      vit_alti <= vit_alti_def;
      is_started <= '0';
    else
      if rising_edge(CLK48Hz) then
        -- Start beweging wanneer de game in gestart.
        if is_started='1' then
          pos_alti <= pos_alti + vit_alti;
        else
          pos_alti <= pos_alti;
        end if;

        -- als fly = 1 dan is de knop/sensor ingedrukt
        if fly='1' then
          vit_alti <= vit_alti - "00000000001"; -- vlieg omhoog
          is_started <= '1'; -- start de game
        else
          -- limiteer snelheid wanner deze te hoog is
          if vit_alti < "00000000100" then
            vit_alti <= vit_alti + "00000000001"; -- vlieg omlaag (zwaarte
kracht)
          else
            vit_alti <= vit_alti; -- limiteer snelheid
          end if;
          is_started <= is_started;
        end if;
      else
        pos_alti <= pos_alti;
        vit_alti <= vit_alti;
```

```
        is_started <= is_started;
      end if;
    end if;
  end process;

  altitude <= pos_alti;
  started <= is_started;
```



4.1.3 Fly counter

Aangezien ik standaard de geluids sensor niet had voorzien heb ik wel grootte aanpassingen moeten doen aan men jump code.

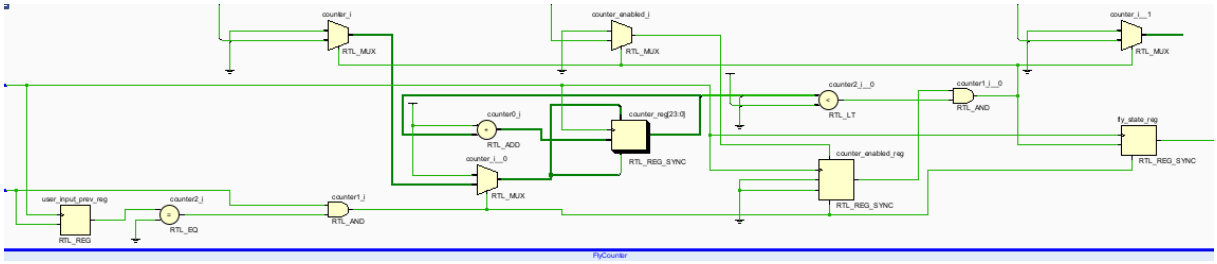
Hiervoor moest je de knop inhouden om omhoog te gaan (wat probleem is). Aangezien de sensor maar een puls geeft wanneer deze wordt getriggerd is dit een probleem (ten zij je 48Hz snel kan klappen met je handen ;)).

Hierdoor was dus nood aan een counter die ons voor bepaalde tijd (1MHz / 150k = 150ms) zal laten stijgen. Dit gebeurt door de fly counter.

fly_counter.vhd:

```
entity FlyCounter is
    Port ( clk : in STD_LOGIC;
           user_input : in STD_LOGIC;
           fly : out STD_LOGIC);
end FlyCounter;

architecture Behavioral of FlyCounter is
    Constant counter_threshold : integer := 15000000; -- 15000000 * (1 /
100MHz) = 0.15 seconden
    signal counter : integer range 0 to counter_threshold;
    signal fly_state : STD_LOGIC := '0';
```

```
    signal user_input_prev : STD_LOGIC := '0';
    signal counter_enabled : BOOLEAN := FALSE;

begin
    process(clk)
    begin
        if rising_edge(clk) then
            if user_input = '1' and user_input_prev = '0' then -- Als er een
user input is en de counter nog niet gestart is
                counter <= 1; -- reset de counter
                fly_state <= '1';  -- laat de speler vliegen
                counter_enabled <= TRUE;
            elsif counter_enabled and counter < counter_threshold then -- als
de counter is gestart en niet is afgelopen
                counter <= counter + 1;
                fly_state <= '1';  -- laat de speler vliegen
            else -- in rust stand
                counter <= 0;
                fly_state <= '0';
                counter_enabled <= FALSE;
            end if;

            user_input_prev <= user_input;
        end if;
    end process;

    fly <= fly_state; -- map deze fly state aan de uitgang van de component

end Behavioral;
```



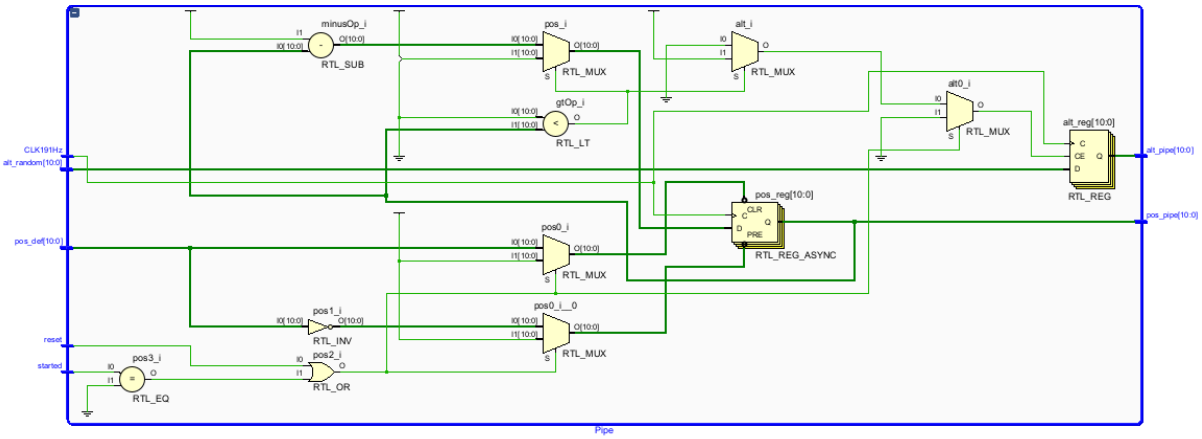
4.2 Pipe (objecten)

4.2.1 Beweging pipe:

De pipe is niet stationair, maar beweegt dus richting te speler. Dit gebeurt met een kloksnelheid van 191Hz. Om het effect te geven dat het spel oneindig is gaan we gebruiken maken van 2 pipes. Wanneer pipe1 tot aan het einde van het scherm komt (linker kant) gaan we pipe2 teleporteren naar begin van het scherm (rechter kant). En wanneer deze tot aan het einde komt (linker kant) zal deze pipe1 teleporteren naar het begin (rechter kant). Hierdoor wissel je heel te tijd af tussen beide pipes, en lijk het spel oneindig, maar zijn het eigenlijk maar 2 pipes die geteleporteert worden.

Pipe.vhd:

```
clockActive: process(CLK191Hz, reset, started)
begin
if reset='1' or started='0' then
  pos <= pos_def; -- reset naar default positie
else
  if rising_edge(CLK191Hz) then
    if pos > "00000000000" and started='1' then
      pos <= pos - 1; -- beweeg de pipe naar links
    else
      alt <= alt_random; -- geef de pipe een random hoogte
      pos <= pos_bordure; -- reset naar rechter border
    end if;
  else
    pos <= pos; -- blijf op dezelfde positie
  end if;
end if;
end process;
```



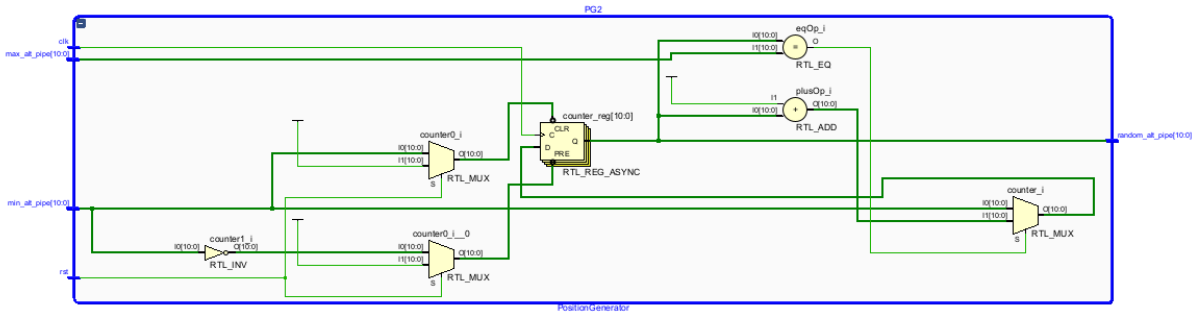
4.2.2 Random positie pipe

De pipes worden altijd op een random locatie gespawned. Dit randomness creëren we doormiddel van een counter die heel de spel lang blijft draaien met een clock frequentie niet gelijk zijn met de frequentie in het spel gebruikt worden. De counter telt van de minimum pipe hoogte tot de maximum toegelaten pipe hoogten.

PositionGenerated.vhd:

```
entity PositionGenerator is
    Port ( min_alt_pipe : in STD_LOGIC_VECTOR (10 downto 0);
           max_alt_pipe : in STD_LOGIC_VECTOR (10 downto 0);
           random_alt_pipe : out STD_LOGIC_VECTOR (10 downto 0);
           clk : in STD_LOGIC;
           rst : in STD_LOGIC);
end PositionGenerator;

architecture Behavioral of PositionGenerator is

signal counter : STD_LOGIC_VECTOR (10 downto 0) := min_alt_pipe;
begin
    process(clk, rst)
    begin
        if rst = '1' then
            counter <= min_alt_pipe; -- reset counter naar minimale hoogte
        elsif rising_edge(clk) then
            if (counter = max_alt_pipe) then
                counter <= min_alt_pipe;  -- reset counter naar minimale
hoogte als max hoogte is bereikt
            else
                counter <= counter + 1; -- verhoog de counter met 1
            end if;
```

```
        end if;
    end process;

    random_alt_pipe <= counter;
end Behavioral;
```



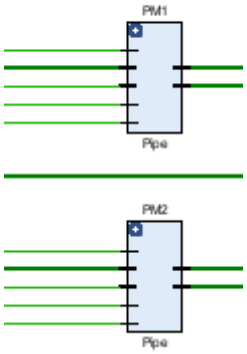
4.2.3 Meerde instancies van component

Ik maak gebruik van 2 pipes zodat er op bepaalde momenten 2 pipes in beeld zitten, dit help met het oneindig effect creëren en continuïteit te verhogen.

Game.vhl:

```
  -- Pipe 1
  PM1 : Pipe
  port map (
    pos_def => "01011000000", -- (640+64)
    alt_random => random_alt_pipe1,
    CLK191Hz => CLK191Hz,
    reset => reset,
    started => started,
    pos_pipe => pos_pipe1,
    alt_pipe => alt_pipe1
  );

  -- Pipe 2
  PM2 : Pipe
  port map (
    pos_def => "00101100000", -- (640+64)/2
    alt_random => random_alt_pipe2,
    CLK191Hz => CLK191Hz,
    reset => reset,
    started => started,
    pos_pipe => pos_pipe2,
    alt_pipe => alt_pipe2
```

```
);
```

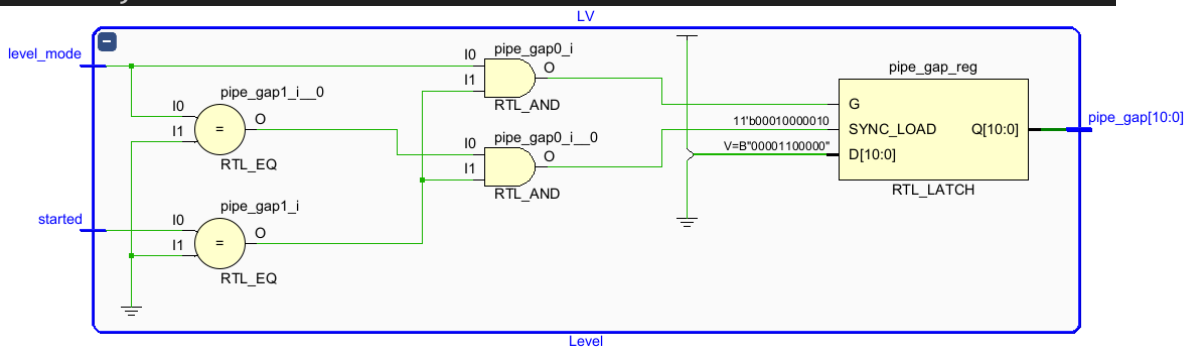

PM1
Pipe
PM2
Pipe

4.3 Level kiezen

Aangezien dat ik bij het speler merkten dat het spel soms moeilijk en uitdagend kon zijn heb ik gekozen om het ook mogelijk te maken dat je het spel makkelijker kan maken. Dit door een switch op de juiste positie te zetten.

Level.vhd:

```
entity Level is
    Port (
        level_mode : in STD_LOGIC;
        pipe_gap : out std_logic_vector(10 downto 0);
        started : in STD_LOGIC

    );
end Level;

architecture Behavioral of Level is

Signal level1: std_logic_vector(10 downto 0):= "00010000010";
Signal level2: std_logic_vector(10 downto 0):= "00001100000";

begin

process(level_mode) -- check het na welk skill level de user wilt gebruiken
begin
    if (level_mode = '0' and started = '0') then -- vergroot de pipe gap
        pipe_gap <= level1;
     elsif (level_mode = '1' and started = '0') then -- verklein de pipe gap
        pipe_gap <= level2;
    end if;
end process;

end Behavioral;
```



4.4 Collision detecteren

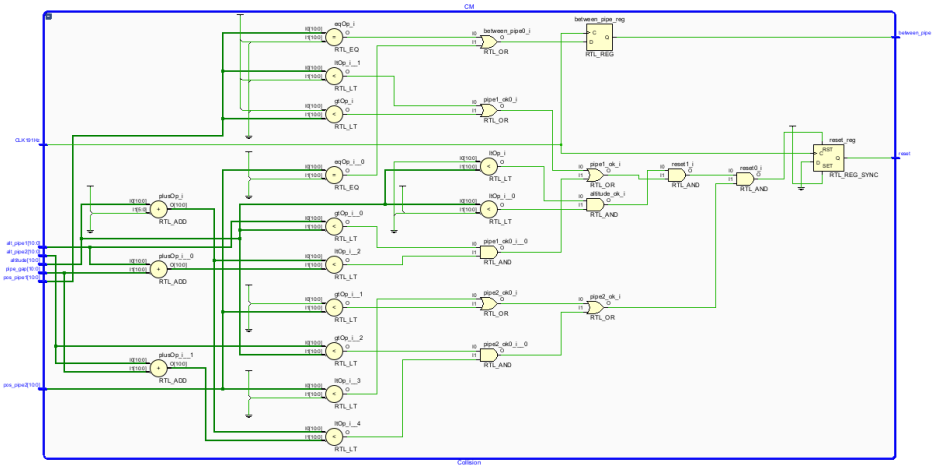
Collisions detecteren zijn zeer belangrijk voor de volgende scenario's

- Checken of speler zich binnen het speelveld bevind
- Checken of speler een pipe heeft geraakt.
- Checken of een speler zich tussen 2 pipes bevind (scoreboard)

Deze checks worden uitgevoerd aan een kloksnelheid van 191Hz (zelfde snelheid waarmee pipes bewegen). We gaan detecties detecteren door constant de positie van de speler te vergelijken met deze van de pipes of speelveld.

Collision.vhd:

```
  altitude_ok <= (min_altitude < altitude) and (altitude < max_altitude); --
als speler zich binnen de spelomgeving begint
  pipe1_ok <= (pos_pipe1 < bird_X) or
              (pos_pipe1 > bird_X + bird_size + pipe_width) or
              ((altitude > alt_pipe1) and
               (altitude + bird_size < alt_pipe1 + pipe_gap)); -- als speler
postitie niet overlapt met pipe1
  pipe2_ok <= (pos_pipe2 < bird_X) or
              (pos_pipe2 > bird_X + bird_size + pipe_width) or
              ((altitude > alt_pipe2) and
               (altitude + bird_size < alt_pipe2 + pipe_gap)); -- als speler
postitie niet overlapt met pipe2

  do_reset <= '0' when altitude_ok and pipe1_ok and pipe2_ok else '1'; -- als
alle collision oke zijn, doe geen reset anders wel
  reset <= do_reset when rising_edge(CLK191Hz);  -- syncroniseer de reset

  -- Check of de speler zich tussen een pipe bevind (gebruikt door scoreboard)
  process(CLK191Hz)
  begin
    if rising_edge(CLK191Hz) then
        between_pipe <= (pos_pipe1 = bird_X) or (pos_pipe2 = bird_X);
    end if;
  end process;
```



4.5 Scoreboard

Deze counter kan in theorie tot 999 tellen. Maar is praktisch natuurlijk niet mogelijk om te halen. Scoreboard heeft 2 soorten scoreboards waar je kan tussen wisselen. “Current score” → deze geeft de score weer die je momenteel in het spel aan het halen bent. “high score” → deze geeft de hoogste score weer die de speler heeft gehaald (dit reset uiteraard als de fpga wordt uigetrokken).

4.5.1 current_score & high_score

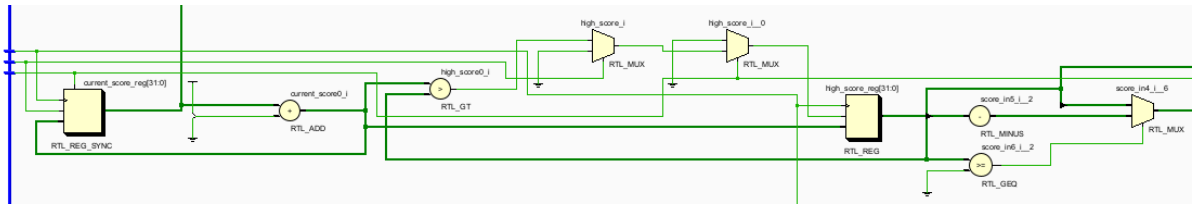
in deze code gaan we elke puls van 191 Hz klok (de zelfde als de collision en pipes). checken of de speler zich tussen de pipes bevind, indien ja verhoog de current_counter

Indien de current_score is groter is dan de high_score zal de current_score gelijk gezet worden met de high_score. Indien het spel reset, reset de current_score.

Scoreboard.vhd:

```
    -- Score optellen en resetten
    process(CLK191Hz)
    begin
    if rising_edge(CLK191Hz) then
      if reset = '1' then
```

```
        current_score <= 0; -- reset
      else
        if between_pipe then -- Als speler zich tussen de pipe bevind -->
verhoog momentele score
          current_score <= current_score + 1;
          if current_score + 1 > high_score then -- Als momentele score hoger
is dan de current_score --> maak current_score de high_score
            high_score <= current_score + 1;
          end if;
        end if;
      end if;
    end if;
    end process;
```

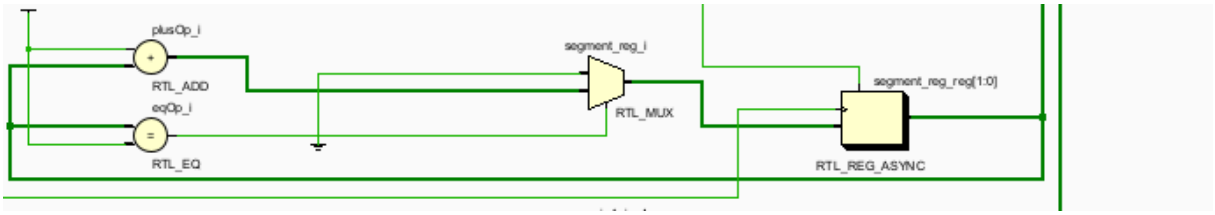


4.5.2 7 segment display

We hebben een counter nodig die individueel de verschillende digits van de 4digit 7segment display aanstuurt: Eerst AN0 dan AN1 dan AN2 dan AN3 …

Scoreboard.vhd:

```
process(CLK191Hz, reset) -- 0 tot 3 counter --> stuurt AN inputs aan van de
7segment display
begin
  if reset = '1' then
      segment_reg <= "00";  -- Reset
  elsif rising_edge(CLK191Hz) then
      if segment_reg = "11" then
          segment_reg <= "00";  -- Reset
      else
          segment_reg <= segment_reg + 1;  -- verhoog te counter
      end if;
  end if;
end process;
```

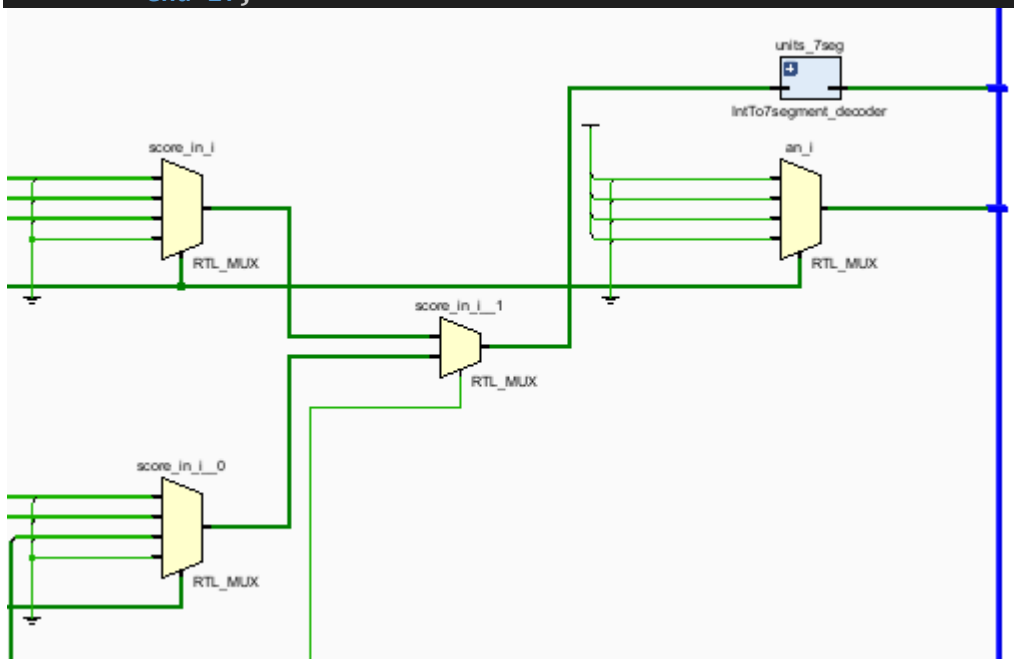


Nu gaan we afhankelijk van de keuze van de score (high score of current score). De 4 digits tonen op de display. We gaan de int splitsen in *eenheden*, *tientallen* en *honderdtallen*. Hierbij kunnen *maximum score van 999* tonen.

Scoreboard.vhd:

```
process(segment_reg) -- Multiplexing
    begin
       if score_mode = '0' then -- als score_mode gelijk is aan momentele
score
           case (segment_reg) is
              when "00" =>
                 an <= "1110";
                 score_in <= current_score mod 10; -- stuur de eenheden van de
current score naar laatste 7segement display
              when "01" =>
                 an <= "1101";
                 score_in <= (current_score / 10) mod 10; -- stuur de
tientallen van de current score naar meeste voorlaaste 7segement display
              when "10" =>
                 an <= "1011";
                 score_in <= (current_score / 100) mod 10; -- stuur de
hondertallen van de current score naar meeste 2de 7segement display
              when "11" =>
                 an <= "0111";
                 score_in <= 0; -- stuur altijd een 0 naar de eerste 7segement
display
              when others =>
                 an <= "1111"; -- stop alle 7segement displays als de register
een ander waarde heeft dan 0 1 2 3 (binaire)
           end case;
         else -- als score_mode gelijk is aan hoogste score
            case (segment_reg) is
              when "00" =>
                 an <= "1110";
                 score_in <= high_score mod 10; -- stuur de eenheden van de
high score naar laatste 7segement display
              when "01" =>
                 an <= "1101";
```

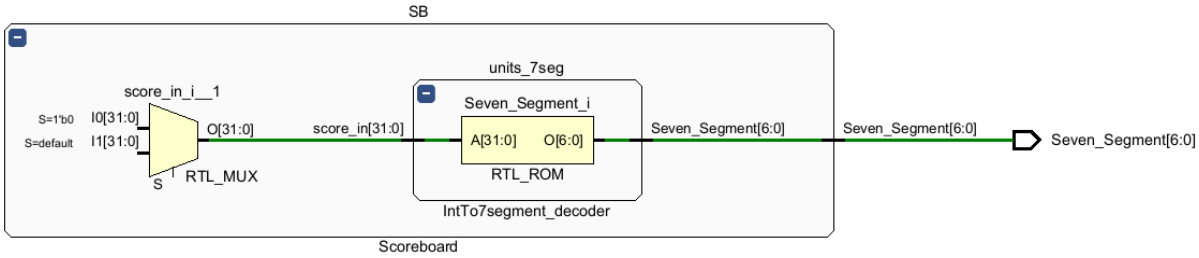
```
                score_in <= (high_score / 10) mod 10; -- stuur de tientallen
van de high score naar meeste voorlaaste 7segement display
            when "10" =>
                an <= "1011";
                score_in <= (high_score / 100) mod 10; -- stuur de
hondertallen van de high score naar meeste 2de 7segement display
            when "11" =>
                an <= "0111";
                score_in <= 0; -- stuur altijd een 0 naar de eerste 7segement
display
            when others =>
                an <= "1111"; -- stop alle 7segement displays als de register
een ander waarde heeft dan 0 1 2 3 (binaire)
          end case;
        end if;
```



nu gaan we de integer naar de int-to-segment-decoder sturen. Die werkelijk de integer op de 7-segment display zal tonen.

IntTo7Segment_decoder.vhd:

```
entity IntTo7segment_decoder is Port (
    score_in : in INTEGER;
    Seven_Segment : out STD_LOGIC_VECTOR (6 downto 0);
    an : out STD_LOGIC_VECTOR (3 downto 0)
);
end IntTo7segment_decoder;

architecture Behavioral of IntTo7segment_decoder is

begin

process(score_in)
begin
    case score_in is -- multiplixer
        when 0 =>
            Seven_Segment <= "1000000"; ---0
        when 1 =>
            Seven_Segment <= "1111001"; ---1
        when 2 =>
            Seven_Segment <= "0100100"; ---2
        when 3 =>
            Seven_Segment <= "0110000"; ---3
        when 4 =>
            Seven_Segment <= "0011001"; ---4
        when 5 =>
            Seven_Segment <= "0010010"; ---5
        when 6 =>
            Seven_Segment <= "0000010"; ---6
        when 7 =>
            Seven_Segment <= "1111000"; ---7
        when 8 =>
            Seven_Segment <= "0000000"; ---8
        when 9 =>
            Seven_Segment <= "0010000"; ---9
        when others =>
            Seven_Segment <= "1000000"; ---0
    end case;
end process;

end Behavioral;
```



4.6 Display

Hierbij gaan we de figuren genereren en inkleuren met de juiste kleur.

Display.vhd:

```
entity Display is
  Port (
    CLK_BG    : in   STD_LOGIC_VECTOR (6 downto 0); -- clock voor actergrond
effecten

    -- VGA controller
    blank     : in   STD_LOGIC;
    hcount    : in   STD_LOGIC_VECTOR (10 downto 0);
    vcount    : in   STD_LOGIC_VECTOR (10 downto 0);

    -- Game state
    altitude  : in   STD_LOGIC_VECTOR (10 downto 0);
    pos_pipe1 : in   STD_LOGIC_VECTOR (10 downto 0);
    alt_pipe1 : in   STD_LOGIC_VECTOR (10 downto 0);
    pos_pipe2 : in   STD_LOGIC_VECTOR (10 downto 0);
    alt_pipe2 : in   STD_LOGIC_VECTOR (10 downto 0);
    pipe_gap : in STD_LOGIC_VECTOR (10 downto 0);


    -- Output
    color     : out  STD_LOGIC_VECTOR (7 downto 0)
  );
end Display;

architecture Behavioral of Display is

  Signal is_bird:  BOOLEAN; -- True als gebied de speler bevat
  Signal is_grass: BOOLEAN; -- True als gebied de grass bevat
  Signal is_bar:   BOOLEAN; -- True als gebied de bar (boven op gras) bevat
  Signal is_pipe1: BOOLEAN; -- True als gebied de pipe1 bevat
  Signal is_pipe2: BOOLEAN; -- True als gebied de pipe2 bevat

  Constant bird_X : STD_LOGIC_VECTOR (10 downto 0) := "00001000000"; -- X
positie of de speler
```

```
  Constant bird_size : STD_LOGIC_VECTOR (10 downto 0) := "00000100000"; --
groote van de speler
  Constant sky_height : STD_LOGIC_VECTOR (10 downto 0) := "00110110000"; --
hoogte van de lucht
  Constant bar_height : STD_LOGIC_VECTOR (10 downto 0) := "00000000100"; --
hoogte van de bar
  Constant pipe_width : STD_LOGIC_VECTOR (10 downto 0) := "00001000000"; --
breedte van de pipe

  -- Colors RRRGGGBB
  -- Generate with :
bin(int(0x71/255*2**3)),bin(int(0xc5/255*2**3)),bin(int(0xcf/255*2**2))
  Constant bird_color : STD_LOGIC_VECTOR (7 downto 0) := "11000100";
  Constant grass_color : STD_LOGIC_VECTOR (7 downto 0) := "11011010";
  Constant bar_color : STD_LOGIC_VECTOR (7 downto 0) := "01111000";
  Signal animated_pipe_color : STD_LOGIC_VECTOR (4 downto 0);
  Signal pipe_color : STD_LOGIC_VECTOR (7 downto 0) := "00010000";
  Signal animated_bg_color : STD_LOGIC_VECTOR (6 downto 0);
  Signal background_color : STD_LOGIC_VECTOR (7 downto 0) := "01101111";

begin

  -- Test areas
  is_bird <= (hcount > bird_X) and
             (hcount < bird_X + bird_size) and
             (vcount > altitude) and
             (vcount < altitude + bird_size);
  is_pipe1 <= (hcount + pipe_width > pos_pipe1) and
              (hcount + pipe_width < pos_pipe1 + pipe_width) and
              ((vcount < alt_pipe1) or
              (vcount > alt_pipe1 + pipe_gap));
  is_pipe2 <= (hcount + pipe_width > pos_pipe2) and
              (hcount + pipe_width < pos_pipe2 + pipe_width) and
              ((vcount < alt_pipe2) or
              (vcount > alt_pipe2 + pipe_gap));
  is_grass <= vcount > sky_height + bar_height;
  is_bar <= vcount > sky_height;

  -- Animated background
  animated_bg_color <= vcount(6 downto 0) - hcount(6 downto 0) - CLK_BG;
  background_color <= "01101111" when animated_bg_color(6)='1' else
"01001011";

  -- Animated pipes
  animated_pipe_color <= vcount(4 downto 0) + hcount(4 downto 0) - pos_pipe1(4
downto 0);
```

```
  pipe_color <= "00010000" when animated_pipe_color(4)='1' else "00001100";

  -- Define current pixel color
  color <= (others=>'0') when blank='1'     else
           bird_color   when is_bird=true  else
           grass_color  when is_grass=true else
           bar_color    when is_bar=true   else
           pipe_color   when (is_pipe1=true or is_pipe2=true) else
           background_color;

end Behavioral;
```

4.7 VGA controller

-- Author : Ulrich Zoltán

http://www.epanorama.net/documents/pc/vga_timing.html

Vga_controller_640_60.vhdl:

```
begin

  -- output horizontal and vertical counters
  hcount <= hcounter;
  vcount <= vcounter;

  -- blank is active when outside screen visible area
  -- color output should be blacked (put on 0) when blank in active
  -- blank is delayed one pixel clock period from the video_enable
  -- signal to account for the pixel pipeline delay.
  blank <= not video_enable when rising_edge(pixel_clk);

  -- increment horizontal counter at pixel_clk rate
  -- until HMAX is reached, then reset and keep counting
  h_count: process(pixel_clk)
  begin
     if(rising_edge(pixel_clk)) then
        if(rst = '1') then
           hcounter <= (others => '0');
        elsif(hcounter = HMAX) then
           hcounter <= (others => '0');
        else
           hcounter <= hcounter + 1;
        end if;
     end if;
  end process h_count;

  -- increment vertical counter when one line is finished
  -- (horizontal counter reached HMAX)
  -- until VMAX is reached, then reset and keep counting
  v_count: process(pixel_clk)
  begin
     if(rising_edge(pixel_clk)) then
        if(rst = '1') then
           vcounter <= (others => '0');
        elsif(hcounter = HMAX) then
           if(vcounter = VMAX) then
              vcounter <= (others => '0');
           else
```

```
            vcounter <= vcounter + 1;
         end if;
      end if;
   end if;
end process v_count;

-- generate horizontal synch pulse
-- when horizontal counter is between where the
-- front porch ends and the synch pulse ends.
-- The HS is active (with polarity SPP) for a total of 96 pixels.
do_hs: process(pixel_clk)
begin
   if(rising_edge(pixel_clk)) then
      if(hcounter >= HFP and hcounter < HSP) then
         HS <= SPP;
      else
         HS <= not SPP;
      end if;
   end if;
end process do_hs;

-- generate vertical synch pulse
-- when vertical counter is between where the
-- front porch ends and the synch pulse ends.
-- The VS is active (with polarity SPP) for a total of 2 video lines
-- = 2*HMAX = 1600 pixels.
do_vs: process(pixel_clk)
begin
   if(rising_edge(pixel_clk)) then
      if(vcounter >= VFP and vcounter < VSP) then
         VS <= SPP;
      else
         VS <= not SPP;
      end if;
   end if;
end process do_vs;

-- enable video output when pixel is in visible area
video_enable <= '1' when (hcounter < HLINES and vcounter < VLINES) else '0';

end Behavioral;
```

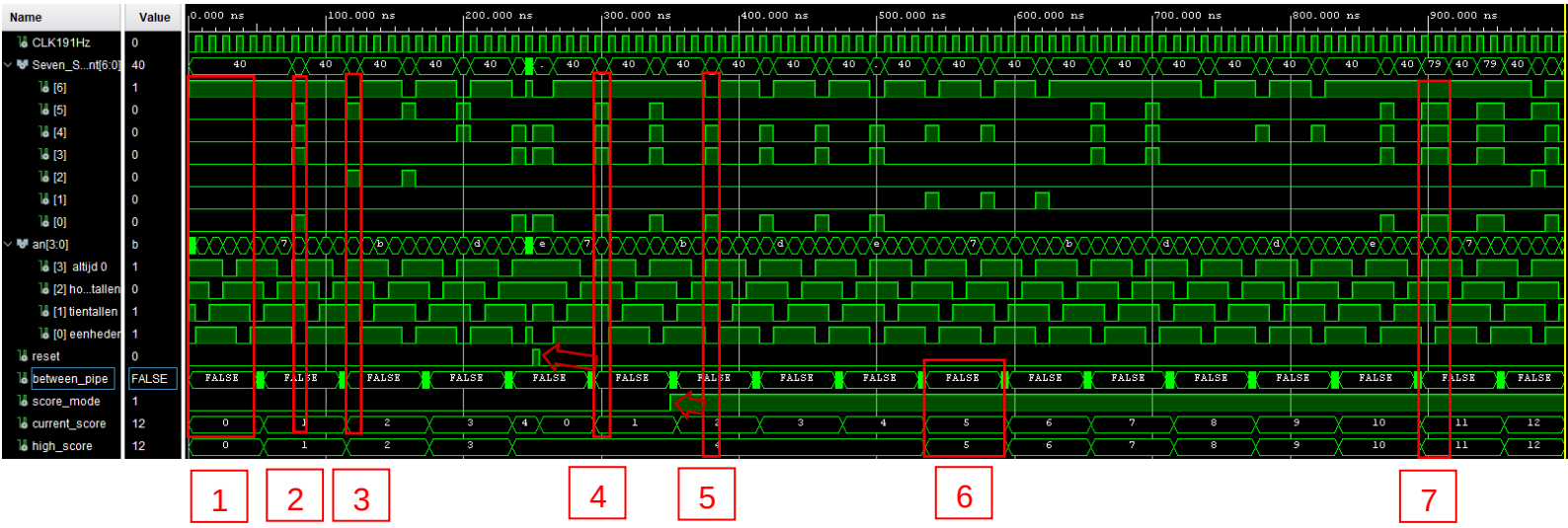
5 Simulatie

Aangezien een volledige game zeer moeilijk te simuleren valt, aangezien het vaak eenvoudiger is om je resultaat rechtstreeks op de vga display te bekijken, daarom heb ik er voor gekozen op het spel op te delen uit enkele kleinere simulaties en deze beknopt uit te leggen.

5.1 Scoreboard

```
UUT: Scoreboard
port map(CLK191Hz => CLK191Hz, reset=>reset, between_pipe=>between_pipe,
Seven_Segment=>Seven_Segment, an=>an, score_mode=>score_mode);

CLK191Hz <= not CLK191Hz after 5 ns; -- clock simulatie



process -- simuleer wanneer er een punt gescoord word
begin
 wait for 50 ns;
 between_pipe <= true;
 wait for 5 ns;
 between_pipe <= false;
 wait for 5 ns;
end process;


process -- simuleer dat speler het spel faalt
begin
   wait for 250 ns;
   reset <= '1';
   wait for 5 ns;
   reset <= '0';
   wait for 1000 ns;
end process;

process -- simuleer dat speler score modus veranderd van momentele score naar
highscore
begin
   wait for 350 ns;
   score_mode <= '1'; -- verandere score modus naar high score
   wait for 1000 ns;
end process;
```

```
process(CLK191Hz) -- de score in INT formaat bekijken
begin
if rising_edge(CLK191Hz) then
 if reset = '1' then
   current_score <= 0; -- reset
 else
   if between_pipe then -- Als speler zich tussen de pipe bevind --> verhoog
momentele score
     current_score <= current_score + 1;
     if current_score + 1 > high_score then -- Als momentele score hoger is
dan de current_score --> maak current_score de high_score
       high_score <= current_score + 1;
     end if;
   end if;
 end if;
end if;
end process;
```

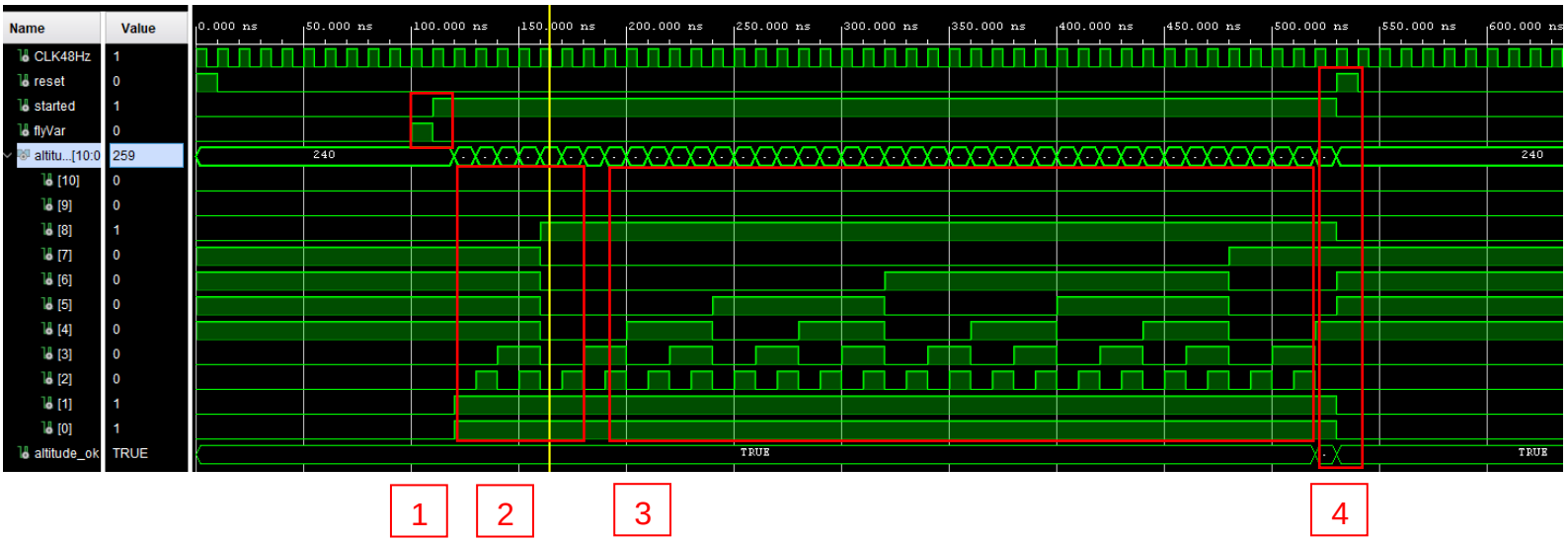


1. Er is nog geen punt gescoord (between pipe is nog niet ‘true’ geweest). Eenheden tientallen en hondertallen geven allemaal nog 7 segment 0 aan.
2. Er is een punt gescoord! De 7 segment van an0 (eenheden) geeft een 1 aan → “1111001” (score_mode = current_score)
3. Er is weer een punt gescoord! De 7 segment van an0 (eenheden) geeft een 2 aan → “0100100”
4. Er is weer een punt gescoord! Enkel is de speler dood gegaan net voor het nieuwe punt (reset was 1). Hierbij geeft het 7 segment van an0 (eenheden) weer een 1 aan. → “1111001”

5. Er is weer een punt gescoord! Enkel heeft de speler de score_mode aangepast naar “high_score” dit betekend dat op de 7segment display niet de momentele score meer toont maar de hoogste score die de speler heeft gehaald. In dit geval is dit voor 4 op an0 (eenheden) → 0011001
6. De speler is namelijk bezig aan een goede score! Hij heeft namelijk de highscore verbroken (merk op nu loopt de high score en momentele score terug gelijk)
7. De score van de speler bedraagt nu 11, dit betekend dat niet alleen de eenheden (an0) een waarde 1 krijgt maar ook de tientallen (an1).

5.2 Fly

Ook heb ik het vliegen gesimuleerd, om te checken of deze omhoog en naar beneden wilt bewegen. En dat het spel zijn eigen automatisch reset nadat hij de vloer of plafond raakt.



1. De speler heeft de knop ingedrukt voor 10 ns, het spel is dus gestart
2. De speler zal omhoog gaan voor een bepaalde periode
3. Aangezien de speler niet terug heeft gedrukt zal de speler naar beneden vallen (in theorie wordt het getal hoger)
4. Speler heeft het laagste mogelijke punt bereikt. De speler wordt terug op begin hoogte geteleporteerd! En “started” wordt terug op 0 gezet

6 Project

Mijn project kan je terug vinden op mijn github repository →
https://github.com/Gehug/flappy-bird-fpga

Demo video kan je hier terug vinden →
https://ap.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=99896ceb-e507-404e-98c9-b0f601405ae0