Documentatie

Digital system development

Niels Aarts | 2VTITIOT

2023 - 2024

# Inhoudsopgave

[Inhoudsopgave 2](#_Toc154781780)

[Afkortingen & Termen 2](#_Toc154781781)

[Inleiding 3](#_Toc154781782)

[Hierarchy 3](#_Toc154781783)

[Board 4](#_Toc154781784)

[Basys 3 4](#_Toc154781785)

[Constrain file | Gebruikte hardware 4](#_Toc154781786)

[Developmentplan 5](#_Toc154781787)

[Test plan 6](#_Toc154781788)

[Doel 6](#_Toc154781789)

[Testdoelen 6](#_Toc154781790)

[Testomgeving 6](#_Toc154781791)

[Testprocedures 6](#_Toc154781792)

[TestBench 8](#_Toc154781793)

[Code 9](#_Toc154781794)

[VGA 9](#_Toc154781795)

[Clk-IP 13](#_Toc154781796)

[Score in 7-segment display 14](#_Toc154781797)

[Besturing 16](#_Toc154781798)

[Basys 3 16](#_Toc154781799)

[Keyboard 18](#_Toc154781800)

[Conclusie 20](#_Toc154781801)

[Bronnen 21](#_Toc154781802)

# Afkortingen & Termen

|  |  |  |
| --- | --- | --- |
| **Afkorting** | **Betekenis** | **Context** |
| FPGA | Field-Programmable Gate Array | Hardware bordje |
| VHDL | VHSIC Hardware Description Language | Programmeer taal |
| VGA | Video Graphics Array | Video poort aansluiting |

# Inleiding

Dit project omvat een spel dat specifiek is ontwikkeld voor de Basys 3, geïmplementeerd met behulp van de VHDL-taal. Het maakt gebruik van invoer van knoppen (voor spelbesturing) of een toetsenbord en schakelaar (reset) op de Basys 3, en toont de gegevens op zowel de 7-Segment Display als op een monitor die de VGA-standaard ondersteunt.

In dit spel heeft de gebruiker de controle over een bewegend object dat verticaal opereert, terwijl horizontaal bewegende balken moeten worden ontweken. De bediening van het object gebeurt via knoppen op de Basys 3 of via het toetsenbord. Zodra een van de knoppen wordt ingedrukt, beweegt het object met een constante snelheid omhoog of omlaag. Het object blijft in beweging, waardoor de gebruiker actief de controle moet behouden. Naarmate de tijd verstrijkt, stijgt de score van de gebruiker. Het spel eindigt wanneer het object een van de balken raakt of wanneer het de bovenste of onderste delen van het scherm bereikt. Op dat moment verschijnt er een melding op de monitor die het einde van het spel aankondigt. De score stopt ook met toenemen, zodat de gebruiker zijn eindscore kan zien.

Het spel is opgebouwd uit vijf modules: 1 voor de VGA-bediening, 1 voor de Button Debouncer, 1 voor de weergave met 7-segmenten op de Basys 3, een topmodule voor het toetsenbord, en 1 topmodule die de spelmechanica omvat.

# Hierarchy

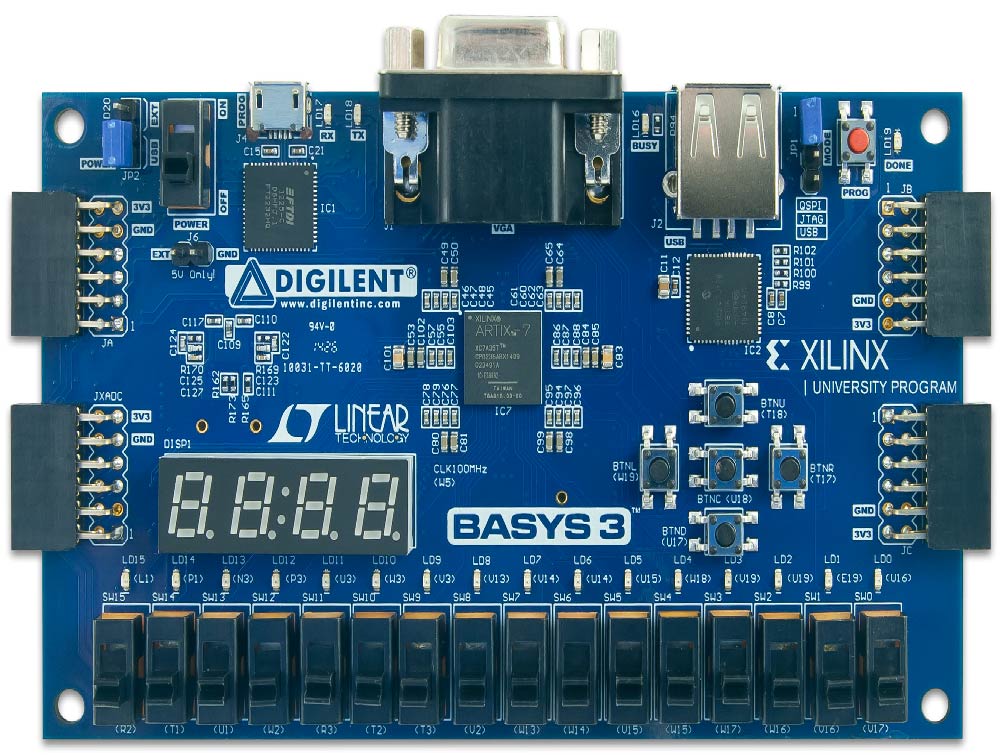
Afbeelding met tekst, elektronica, schermopname, software

Automatisch gegenereerde beschrijving

# Board

## Basys 3

* Family: Artix-7
* Package: cpg236
* Speed grade: -1
* Part number: xc7a35tcpg236-1
* Config flash part number: S25FL032



Figuur 1 Basys 3 board

## Constrain file | Gebruikte hardware

1. Klok - IP
2. 7-segmenten display
3. Switch 15 🡪 reset
4. ButtonU
5. ButtonD
6. VGA – Connector
7. USB - poort

# Developmentplan

Fase 1: Voorbereiding en Conceptualisatie

* Definieer het spelconcept, doelen en vereisten.
* Kies de Basys 3 FPGA en identificeer de benodigde hardware.
* Maak een initiële schets van het spel.

Fase 2: Hardware configuratie en Opzet

* Configureer de Basys 3 FPGA met de gewenste interfaces (USB, VGA, 7-segment display) en sluit het externe toetsenbord aan.
* Schrijf de VHDL-code voor de configuratie van de FPGA en interfaces.

Fase 3: Basisimplementatie

* Implementeer de basisfunctionaliteit: toon de speler, obstakels op het VGA-scherm en houd de score bij op het 7-segment display.
* Programmeer de toetsenbordinterface en gebruik een schakelaar als resetknop.

Fase 4: Uitbreiding van Gameplay

* Voeg complexere gameplay-elementen toe, zoals willekeurige obstakels en geavanceerde spellogica voor beweging en botsingsdetectie.

Fase 5: Testen en Debuggen

* Grondige test van het spel op de FPGA om fouten en bugs te identificeren.
* Optimaliseer code en prestaties voor een soepele werking.

Fase 6: Optimalisatie en Documentatie

* Optimaliseer de code voor betere prestaties en voeg uitgebreide documentatie toe voor toekomstige referentie.

Fase 7: Afronding en Presentatie

* Evalueer het voltooide spel volgens vooraf vastgestelde doelen.
* Bereid documentatie voor eventuele presentaties of rapporten over het project.

# Test plan

Voor het maken van een spel op een FPGA omvat het testen van verschillende componenten, zoals de knoppen op de FPGA, de VGA-poort, en mogelijk een toetsenbord. Daarnaast moet ook de werking van de 7-segmentdisplay voor het weergeven van de score worden gecontroleerd. Hier is mijn testplan voor dit project:

## Doel

Dit testplan heeft als doel om de functionaliteit van de FPGA-game te verifiëren en te valideren. Het omvat het testen van de knoppen op de FPGA, de VGA-poort, de toetsenbordinterface en de 7-segmentdisplay voor scoreweergave.

## Testdoelen

1. Testen van de knopfunctionaliteit op de FPGA.

2. Valideren van de VGA-uitvoer om ervoor te zorgen dat deze correct wordt gegenereerd.

3. Testen van de toetsenbordinterface.

4. Controleren of de 7-segmentdisplay de score correct weergeeft.

## Testomgeving

- FPGA-ontwikkelingsbord - Computer - een toetsenbord - VGA-monitor - Testprogramma voor de FPGA-game

## Testprocedures

1. Knoppen op de FPGA testen

a. Start het testprogramma op de FPGA.

b. Druk op elke knop op de FPGA.

c. Controleer of het spel reageert op de knoppen zoals verwacht.

d. Documenteer eventuele problemen of onjuiste reacties.

2. VGA-poort testen

a. Start het testprogramma op de FPGA.

b. Sluit de VGA-monitor aan op de FPGA.

c. Controleer of het VGA-uitvoersignaal op de monitor wordt weergegeven.

d. Controleer of het beeld correct is en geen fouten bevat.

e. Documenteer eventuele problemen of fouten.

3. Toetsenbordinterface testen

a. Sluit een toetsenbord aan op de FPGA.

b. Start het testprogramma op de FPGA.

c. Gebruik aangewezen toetsen op het toetsenbord.

d. Controleer of het spel reageert op de toetsen zoals verwacht.

e. Documenteer eventuele problemen of onjuiste reacties.

4. 7-segmentdisplay voor scoreweergave testen

a. Start het testprogramma op de FPGA.

b. Speel het spel en behaal een score.

c. Controleer of de 7-segmentdisplay de score correct weergeeft.

d. Documenteer eventuele problemen of onjuiste scoreweergave.

**Acceptatiecriteria**

- De knoppen op de FPGA moeten correct werken en de game moet reageren op knopinvoer.

- De VGA-uitvoer moet correct worden gegenereerd en op de monitor worden weergegeven zonder artefacten.

- Als een toetsenbord wordt gebruikt, moet het correct functioneren, en het spel moet reageren op toetsenbordinvoer.

- De 7-segmentdisplay moet de score correct weergeven.

**Testrapportage**

- Ik documenteer de resultaten van elke teststap, inclusief eventuele problemen of fouten.

- Geef aan of de acceptatiecriteria zijn voldaan.

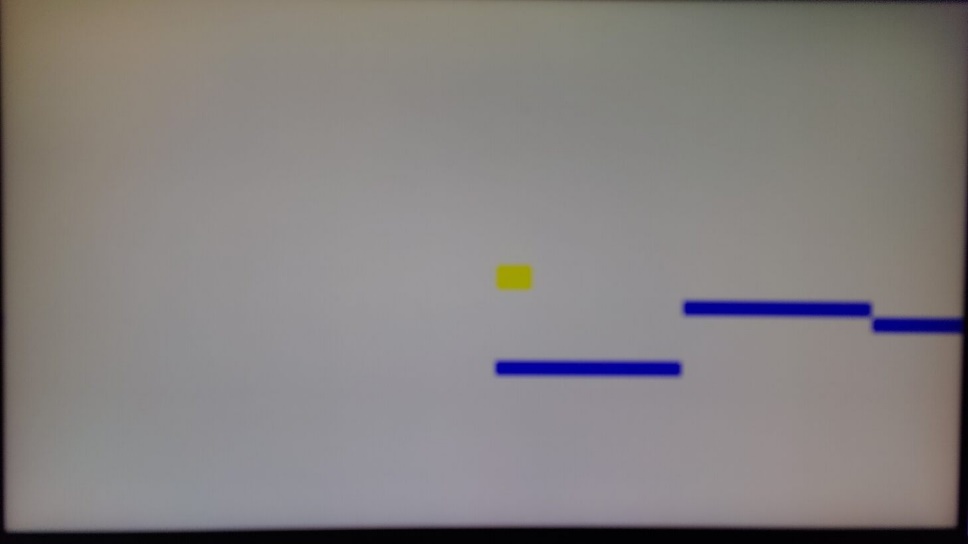
- Indien problemen worden geconstateerd, beschrijf ik deze en geef mogelijke oplossingen aan.

## TestBench

Afbeelding met schermopname, lijn

Automatisch gegenereerde beschrijving

Ik ga enkel de toetsenbord testen om te kunnen laten zien dat ik toetsen kan inlezen in de fpga.

 Afbeelding met tekst, elektronica, Elektronische engineering, schermopname

Automatisch gegenereerde beschrijving

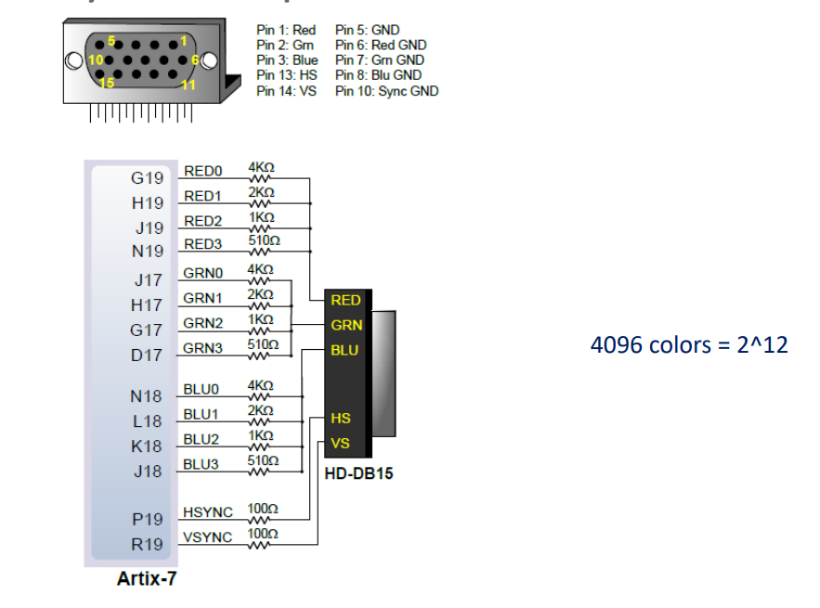
Afbeelding met tekst, schermopname, waas

Automatisch gegenereerde beschrijving

# Code

## VGA

Ik heb de voorbeeldcode van Digitap als referentie gebruikt bij het werken met een FPGA en een camera. Hieronder een korte samenvatting voor wat voor mij belangrijk was.

RGB-signalen: De VGA-connector gebruikt drie analoge signalen voor de kleuren rood (R), groen (G) en blauw (B). Deze signalen worden gegenereerd door de FPGA om verschillende kleuren te produceren. Elke kleur heeft zijn eigen signaal pad.

Horizontale en verticale synchronisatiesignalen: Naast de RGB-signalen, zijn er synchronisatiesignalen voor de horizontale (HSYNC) en verticale (VSYNC) synchronisatie. Deze signalen worden gebruikt om aan te geven waar elke nieuwe rij (VSYNC) en elke nieuwe pixel (HSYNC) op het scherm begint.

Timing: VGA vereist specifieke timing voor de signalen. De juiste timing is cruciaal voor het correct weergeven van het beeld op het scherm. Dit omvat de frequentie van de signalen en de timing tussen de synchronisatie- en kleursignalen.

D-sub connector: De Basys 3 gebruikt meestal een VGA D-sub 15-pins connector om de verschillende signalen door te geven. Deze connector is gestandaardiseerd voor VGA-uitvoer.

In de top code (game.vhd) is er een deel *“drawing the video”* hierin staat wat er weergegeven wordt via de VGA connectie. Hier kan je dus ook de kleuren, vormen en positie aanpassen.

|  |
| --- |
| *--Guy*  *if( Y > posY and Y < posY + sizeOfGuy and X > posX and X < posX + sizeOfGuy ) then*  *if( gameOver = '1') then*  *Red <= 15;*  *Green <= 15;*  *Blue <= 0;*  *else*  *Red <= 15;*  *Green <= 0;*  *Blue <= 0;*  *end if;* |

In boven staande code staat hoe de speler wordt weergegeven. Door de waardes aan te passen bij (Red, Green of Blue) kan je de kleuren aanpassen van de speler.

|  |
| --- |
| entity vga is  Port (  CLK\_I : in STD\_LOGIC;  VGA\_HS\_O : out STD\_LOGIC;  VGA\_VS\_O : out STD\_LOGIC;  VGA\_RED\_O : out STD\_LOGIC\_VECTOR (3 downto 0);  VGA\_BLUE\_O : out STD\_LOGIC\_VECTOR (3 downto 0);  VGA\_GREEN\_O : out STD\_LOGIC\_VECTOR (3 downto 0);  Y : out natural range 0 to 1280;  X : out natural range 0 to 1024;  R : in natural range 0 to 15;  G : in natural range 0 to 15;  B : in natural range 0 to 15  --reset : in STD\_LOGIC  );  end vga; |

|  |
| --- |
| architecture Behavioral of vga is    component clk\_wiz\_0  port  (  clk\_in1 : in std\_logic;  clk\_out1 : out std\_logic  --reset : in std\_logic  );  end component;  --\*\*\*1280x1024@60Hz\*\*\*--  constant FRAME\_WIDTH : natural := 1280;  constant FRAME\_HEIGHT : natural := 1024;    constant H\_FP : natural := 48; --H front porch width (pixels)  constant H\_PW : natural := 112; --H sync pulse width (pixels)  constant H\_MAX : natural := 1688; --H total period (pixels)    constant V\_FP : natural := 1; --V front porch width (lines)  constant V\_PW : natural := 3; --V sync pulse width (lines)  constant V\_MAX : natural := 1066; --V total period (lines)    constant H\_POL : std\_logic := '1';  constant V\_POL : std\_logic := '1';    -- Pixel clock, in this case 108 MHz  signal pxl\_clk : std\_logic;  -- The active signal is used to signal the active region of the screen (when not blank)  signal active : std\_logic;    -- Horizontal and Vertical counters  signal h\_cntr\_reg : std\_logic\_vector(11 downto 0) := (others =>'0');  signal v\_cntr\_reg : std\_logic\_vector(11 downto 0) := (others =>'0');    -- Pipe Horizontal and Vertical Counters  -- signal h\_cntr\_reg\_dly : std\_logic\_vector(11 downto 0) := (others => '0');  -- signal v\_cntr\_reg\_dly : std\_logic\_vector(11 downto 0) := (others => '0');    -- Horizontal and Vertical Sync  signal h\_sync\_reg : std\_logic := not(H\_POL);  signal v\_sync\_reg : std\_logic := not(V\_POL);  -- Pipe Horizontal and Vertical Sync  -- signal h\_sync\_reg\_dly : std\_logic := not(H\_POL);  -- signal v\_sync\_reg\_dly : std\_logic := not(V\_POL);    signal vga\_red\_reg : std\_logic\_vector(3 downto 0) := (others =>'0');  signal vga\_green\_reg : std\_logic\_vector(3 downto 0) := (others =>'0');  signal vga\_blue\_reg : std\_logic\_vector(3 downto 0) := (others =>'0');    begin  pixel\_clock : clk\_wiz\_0 port map (clk\_in1 => CLK\_I, clk\_out1 => pxl\_clk);--, reset => reset);    -- Horizontal counter  process (pxl\_clk)  begin  if (rising\_edge(pxl\_clk)) then  if (h\_cntr\_reg = (H\_MAX - 1)) then  h\_cntr\_reg <= (others =>'0');  else  h\_cntr\_reg <= h\_cntr\_reg + 1;  end if;  end if;  end process;  -- Vertical counter  process (pxl\_clk)  begin  if (rising\_edge(pxl\_clk)) then  if ((h\_cntr\_reg = (H\_MAX - 1)) and (v\_cntr\_reg = (V\_MAX - 1))) then  v\_cntr\_reg <= (others =>'0');  elsif (h\_cntr\_reg = (H\_MAX - 1)) then  v\_cntr\_reg <= v\_cntr\_reg + 1;  end if;  end if;  end process;  -- Horizontal sync  process (pxl\_clk)  begin  if (rising\_edge(pxl\_clk)) then  if (h\_cntr\_reg >= (H\_FP + FRAME\_WIDTH - 1)) and (h\_cntr\_reg < (H\_FP + FRAME\_WIDTH + H\_PW - 1)) then  h\_sync\_reg <= H\_POL;  else  h\_sync\_reg <= not(H\_POL);  end if;  end if;  end process;  -- Vertical sync  process (pxl\_clk)  begin  if (rising\_edge(pxl\_clk)) then  if (v\_cntr\_reg >= (V\_FP + FRAME\_HEIGHT - 1)) and (v\_cntr\_reg < (V\_FP + FRAME\_HEIGHT + V\_PW - 1)) then  v\_sync\_reg <= V\_POL;  else  v\_sync\_reg <= not(V\_POL);  end if;  end if;  end process;    process (pxl\_clk)  begin  if (rising\_edge(pxl\_clk)) then    if (h\_cntr\_reg < FRAME\_WIDTH and v\_cntr\_reg < FRAME\_HEIGHT) then  X <= conv\_integer(h\_cntr\_reg);  Y <= conv\_integer(v\_cntr\_reg);  vga\_red\_reg <= conv\_std\_logic\_vector(R, 4);  vga\_green\_reg <= conv\_std\_logic\_vector(G, 4);  vga\_blue\_reg <= conv\_std\_logic\_vector(B, 4);  else  vga\_red\_reg <= "0000";  vga\_green\_reg <= "0000";  vga\_blue\_reg <= "0000";  end if;    end if;  end process;    VGA\_HS\_O <= h\_sync\_reg;  VGA\_VS\_O <= v\_sync\_reg;  VGA\_RED\_O <= vga\_red\_reg;  VGA\_GREEN\_O <= vga\_green\_reg;  VGA\_BLUE\_O <= vga\_blue\_reg; |

### Clk-IP

Afbeelding met tekst, schermopname, software, nummer

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, nummer, Lettertype

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, nummer, software

Automatisch gegenereerde beschrijving

## Score in 7-segment display

Inspiratie van de stopwatch komt voort uit een mechanisme waarbij een score wordt opgeteld op basis van klokpulsen. Deze score stopt wanneer je de rand van het blok raakt en moet worden gereset met behulp van een schakelaar.

|  |
| --- |
| *-- Timer voor score*  *process(clk)*  *begin*  *-- Scoreteller verhogen bij elke klokpuls*  *if (rising\_edge(clk)) then*  *score\_cnt <= score\_cnt + 1;*  *-- Resetten van scoreteller en score bij resetsignaal*  *if (reset = '1') then*  *score\_cnt <= 0;*  *score <= 0;*  *-- Score verhogen na een bepaald aantal pulsjes en als het spel niet voorbij is*  *elsif (score\_cnt >= 10000000) then*  *score\_cnt <= 0;*  *if (gameOver = '0') then*  *score <= score + 1;*  *end if;*  *end if;*  *end if;*  *end process;*  *-- Weergave van score op 7-segment display*  *process(score)*  *variable integer0: natural range 0 to 9;*  *variable integer1: natural range 0 to 9;*  *variable integer1mod: natural range 0 to 99;*  *variable integer2: natural range 0 to 9;*  *variable integer2mod: natural range 0 to 999;*  *variable integer3: natural range 0 to 9;*  *begin*  *-- Eenheden*  *integer0 := score mod 10;*  *in0 <= conv\_std\_logic\_vector(integer0, 4);*    *-- Tientallen*  *integer1mod := score mod 100;*  *integer1 := integer1Mod / 10;*  *in1 <= conv\_std\_logic\_vector(integer1, 4);*    *-- Honderdtallen*  *integer2mod := score mod 1000;*  *integer2 := integer2mod / 100;*  *in2 <= conv\_std\_logic\_vector(integer2, 4);*    *-- Duizendtallen*  *integer3 := score / 1000;*  *in3 <= conv\_std\_logic\_vector(integer3, 4);*  *end process;* |

Hier zie je de logica voor het versnellen/vermoeilijken van het spel aan de hand van de score. Je ziet dat de Bars en de (guy = speler) versnellen wanneer de score een bepaalde mijlpaal behaald.

|  |
| --- |
| entity SevSeg\_4digit is  Port ( clk : in STD\_LOGIC;  in0, in1, in2, in3 : in STD\_LOGIC\_VECTOR(3 downto 0);  a, b, c, d, e, f, g, dp : out STD\_LOGIC;  an : out STD\_LOGIC\_VECTOR(3 downto 0) );  end entity SevSeg\_4digit; |

|  |
| --- |
| architecture Behavioral of SevSeg\_4digit is  constant N : integer := 18;  signal count : STD\_LOGIC\_VECTOR(N-1 downto 0) := (others => '0');  signal digit\_val : STD\_LOGIC\_VECTOR(3 downto 0);  signal digit\_en : STD\_LOGIC\_VECTOR(3 downto 0);  signal sseg\_LEDs : STD\_LOGIC\_VECTOR(6 downto 0);  begin  process(clk)  begin  if rising\_edge(clk) then  count <= count + "1";  end if;  end process;  process(count, in0, in1, in2, in3)  begin  digit\_en <= "1111"; -- default  digit\_val <= in0; -- default  case count(N-1 downto N-2) is  when "00" => -- select first 7Seg.  digit\_val <= in0;  digit\_en <= "1110";  when "01" => -- select second 7Seg.  digit\_val <= in1;  digit\_en <= "1101";  when "10" => -- select third 7Seg.  digit\_val <= in2;  digit\_en <= "1011";  when others => -- select fourth 7Seg.  digit\_val <= in3;  digit\_en <= "0111";  end case;  end process;  process(digit\_val)  begin  sseg\_LEDs <= "1111111"; -- default  case digit\_val is  when "0000" => sseg\_LEDs <= "1000000"; -- to display 0  when "0001" => sseg\_LEDs <= "1111001"; -- to display 1  when "0010" => sseg\_LEDs <= "0100100"; -- to display 2  when "0011" => sseg\_LEDs <= "0110000"; -- to display 3  when "0100" => sseg\_LEDs <= "0011001"; -- to display 4  when "0101" => sseg\_LEDs <= "0010010"; -- to display 5  when "0110" => sseg\_LEDs <= "0000010"; -- to display 6  when "0111" => sseg\_LEDs <= "1111000"; -- to display 7  when "1000" => sseg\_LEDs <= "0000000"; -- to display 8  when "1001" => sseg\_LEDs <= "0010000"; -- to display 9  when others => sseg\_LEDs <= "0111111"; -- dash  end case;  end process; |

## Besturing

### Basys 3

Als referentie heb ik de educational demo gebruikt op digitap.

|  |
| --- |
| *--Setting the motion up or down*  *process( clk, reset, btnU\_DB, btnD\_DB)*  *begin*  *if( rising\_edge(clk)) then*  *btn\_cnt <= btn\_cnt + 1;*  *if( reset = '1' ) then --Resetting the game*  *moveUp <= '0';*  *moveDown <= '0';*  *elsif( btn\_cnt >= 500000) then*  *btn\_cnt <= 0;*  *--Direction*  *if( btnU\_DB = '1') then*  *moveUp <= '1';*  *moveDown <= '0';*  *elsif( btnD\_DB = '1') then*  *moveUp <= '0';*  *moveDown <= '1';*  *end if;*  *end if;*  *end if;*  *end process;* |

Hierboven staat hoe de richting wordt verandert bij het indrukken van een knop op de FPGA.

**UP bounce**

|  |
| --- |
| ENTITY debounce IS  GENERIC(  counter\_size : INTEGER := 20);  PORT(  clk : IN STD\_LOGIC;  button : IN STD\_LOGIC;  result : OUT STD\_LOGIC);  END debounce;  ARCHITECTURE logic OF debounce IS  SIGNAL flipflops : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL counter\_set : STD\_LOGIC;  SIGNAL counter\_out : STD\_LOGIC\_VECTOR(counter\_size DOWNTO 0) := (OTHERS => '0');  BEGIN  counter\_set <= flipflops(0) xor flipflops(1); --determine when to start/reset counter    PROCESS(clk)  BEGIN  IF(rising\_edge(clk)) THEN  flipflops(0) <= button;  flipflops(1) <= flipflops(0);  If(counter\_set = '1') THEN --reset counter because input is changing  counter\_out <= (OTHERS => '0');  ELSIF(counter\_out(counter\_size) = '0') THEN --stable input time is not yet met  counter\_out <= counter\_out + 1;  ELSE --stable input time is met  result <= flipflops(1);  END IF;  END IF;  END PROCESS;  END logic; |

|  |
| --- |
| ENTITY debounce IS  GENERIC(  counter\_size : INTEGER := 20);  PORT(  clk : IN STD\_LOGIC;  button : IN STD\_LOGIC;  result : OUT STD\_LOGIC);  END debounce;  ARCHITECTURE logic OF debounce IS  SIGNAL flipflops : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL counter\_set : STD\_LOGIC;  SIGNAL counter\_out : STD\_LOGIC\_VECTOR(counter\_size DOWNTO 0) := (OTHERS => '0');  BEGIN  counter\_set <= flipflops(0) xor flipflops(1); --determine when to start/reset counter    PROCESS(clk)  BEGIN  IF(rising\_edge(clk)) THEN  flipflops(0) <= button;  flipflops(1) <= flipflops(0);  If(counter\_set = '1') THEN --reset counter because input is changing  counter\_out <= (OTHERS => '0');  ELSIF(counter\_out(counter\_size) = '0') THEN --stable input time is not yet met  counter\_out <= counter\_out + 1;  ELSE --stable input time is met  result <= flipflops(1);  END IF;  END IF;  END PROCESS;  END logic; |

**Down-bounce**

### Keyboard

Het is mogelijk om een USB-toetsenbord op de Basys 3 aan te sluiten en de FPGA te programmeren om de USB-invoer om te zetten naar een formaat dat compatibel is met de PS/2-standaard, waardoor het mogelijk wordt om met het toetsenbord te communiceren alsof het direct op een PS/2-poort is aangesloten.

Dit onderdeel bestaat uit twee subonderdelen. Het eerste onderdeel, U1: PS2, zet de PS2-klok en PS2-gegevens om naar de scan-code-uitvoer, ps2\_out, en geeft een vlag aan als de gegevens geldig zijn, ps2\_out\_new. PS2\_CR zet de scan-code van het toetsenbord om naar een waarde voor kolom en rij voor sprite 1 (C1 & R1).

Afbeelding met tekst, keyboard, Lettertype, nummer

Automatisch gegenereerde beschrijving

|  |
| --- |
| *-- Check if a key is pressed on the keyboard*  *if key\_ready = '1' then*  *case key\_data is*  *when "00110001" =>*  *moveUp <= '1'; -- Set moveUp based on a specific key*  *moveDown <= '0';*  *when "01011010" =>*  *moveUp <= '0';*  *moveDown <= '1'; -- Set moveDown based on a specific key*  *when others =>*  *moveUp <= '0';*  *moveDown <= '0';*  *end case;*  *end if;* |

|  |
| --- |
| *begin*  *process (clk, rst) begin*  *if rst = '1' then*  *flag <= '0';*  *Up\_int <= '0';*  *Down\_int <= '0';*  *elsif rising\_edge(clk) then*  *prev\_valid <= valid;*  *if (prev\_valid = '1' and valid = '0' and flag = '0') then*  *-- Push key the first time, check data at falling edge*  *case data is*  *when X"****1C****" => Up\_int <= '1'; -- A-Key*  *when X"****15****" => Down\_int <= '1'; -- Q-Key*  *when others => -- Do nothing for other keys*  *end case;*  *flag <= '1';*  *elsif (prev\_valid = '1' and valid = '0' and flag = '1') then*  *-- If flag is set, ignore the next button press*  *case data is*  *when others => -- Do nothing for other keys*  *end case;*  *flag <= '0';*  *end if;*  *end if;*  *end process;* |

In bovenstaande code zie je de controller hier declareer ik welke knoppen er gebruikt gaan worden zoals de eerste foto van boven.

# Conclusie

Bij de eerste implementatie van het toetsenbord functioneerde er aanvankelijk niets. Nadat ik een ander toetsenbord had gebruikt, kreeg ik wel een reactie, maar nog niet helemaal zoals gewenst. Maar dit kon ik snel aanpassen in de spellogica.

Zodra dat alles werkten was het makkelijk om alles af te werken in de top game module. Hier heb ik alles dynamisch gemaakt voor snel aanpassingen te kunnen doen.

Zoals de kleuren van objecten en de guy (speler), snelheid van de guy (speler) en de scoreboard.

# Bronnen

* **Gitingore:** ar61232.txt "Vivado .gitignore example file"

Source: <https://www.xilinx.com/support/answers/61232.html>

* **Info:** Board, fpga

Source: <https://github.com/dries007/Basys3/tree/master/docs>

* **VGA:**

Source: <https://www.fpga4student.com/2018/08/basys-3-fpga-ov7670-camera.html>

* **Toetsenbord:** Code, aansluiting

Source: Digitap: <https://learning.ap.be/course/view.php?id=58878>

* **Github:**

Source: <https://github.com/AartsNiels/geometry-dash>