DSD Projectopdracht

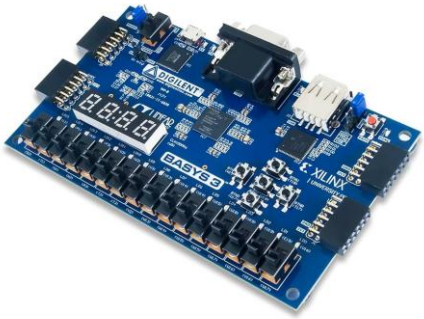
Daan Van der Weken | s150159 | 3ITIOT1

Opleiding: 25-26 IT professional

Academiejaar: 2025-2026

Student: Daan Van der Weken

Lector: D. Van Merode



AP HOGESCHOOL ANTWERPEN

AP.BE

1. Inhoud

2. INTRODUCTIE

In dit document ga ik mijn project voor het vak Digital Systems Development bespreken en becommentariëren. Hier ga ik mijn denkproces en mijn werkwijze uitleggen om het voor de docent en andere personen duidelijk te maken hoe ik tijdens deze les/labo weken aan de slag ben gegaan.

Binnen deze opgave is het de bedoeling dat we zelf in staat zijn om een zelf gekozen doel uit te werken en tot en zo goed mogelijk resultaat te komen. Hierbij moeten we gebruik maken van/rekening houden met onderstaande puntjes:

- Gebruik en bekijk het referentiemateriaal
- Vergaar de juiste datasheets
- Maak een ontwikkelingsplan
- Maak een testplan
- Gebruik een modulair project
- Leg het project uit aan je medestudenten en docent

3. LOGBOEK

Activiteit	Datum	Afgerond?
Uitleg van het project & Keuze onderwerp	17-09-2025	√
Start onderzoek & Opzet documentatie	17-09-2025	√
Vivado project opzetten & Eerste VGA tests	24-09-2025	√
Testplan & Development plan opstellen	01-10-2025	√
Basis grafisch: Achtergrond & Rasterlijnen	08-10-2025	√
Poging tekenen symbolen (X en O) in Top-file	15-10-2025	√
Testfase: Complexe logica in Top-file (Mislukt) *Geprobeerd alles in één bestand te zetten. Dit leverde 'multiple driver' errors op en onleesbare code. Geen vooruitgang geboekt, besloten om de structuur volledig om te gooien.*	22-10-2025 tot 29-10-2025	X
Herstructurering: Modulair ontwerp (Cell/Board) *Code herschreven naar losse componenten (modules). Dit was veel werk, maar loste de errors op.*	05-11-2025	√
Input implementatie (Switches & Buttons)	12-11-2025	√
Debugging Input: Smart Switch logica *Oplossing bedacht voor het feit dat switches fysiek aan bleven staan (masking).*	19-11-2025	√
Implementatie Game Logic (State Machine)	26-11-2025	√
Testfase: Timing errors (Race Conditions) *Onderzoek naar waarom de beurt wisselde tijdens het tekenen (glitch). Intensief getest met LEDs en opgelost met de 'UPDATE_BOARD' tussenstap.*	03-12-2025 tot 09-12-2025	√
Winstdetectie & 7-Segment display	10-12-2025	√

Tekstweergave & "Game Over" scherm	17-12-2025	√
Laatste bugs fixen (Reset denderen), Documentatie & Reflectie	23-12-2025	√
Finale Controle, Video, GitHub optimalisatie	31-12-2025	√

4. KEUZE PROJECT

Toen ik de eerste les moest nadenken over een geschikt onderwerp voor dit project, keek ik eerst naar wat ik zelf in mijn vrije tijd leuk vind. Thuis speel ik graag bordspellen met mijn familie, en dan specifiek strategische spellen. Dit had als gevolg dat ik na een tijd uitkwam bij 4-op-een-rij. Het idee om een eigen, digitale versie van zo'n spel te bouwen sprak me direct aan.

VAN 4-OP-EEN-RIJ NAAR 3-OP-EEN-RIJ

Tijdens de eerste brainstormfase besefte ik echter dat een volledige implementatie van 4-op-een-rij wellicht zeer complex zou zijn als instapproject. Bij dat spel heb je te maken met een groter speelveld (7x6).

Daarom heb ik gekozen voor de logische voorloper daarvan: 3-op-een-rij. Dit spel deelt hetzelfde DNA als 4-op-een-rij, maar is compacter:

- Het is beurt-gebaseerd (Speler A vs Speler B).
- Het draait om patroonherkenning (lijnen maken in een grid).
- Het heeft een duidelijk visueel raster.

Door te kiezen voor 3-op-een-rij kon ik de complexiteit van de spelregels iets verlagen, waardoor ik me volledig kon focussen op de technische uitdaging waar het in dit vak om draait, het beheren van de hardware-timing en het ontwikkelen van een foutloze spellogica in VHDL.

Mijn doel was om een volledig zelfstandig werkend systeem te bouwen, zonder gebruik te maken van een externe microcontroller, maar puur op basis van eigen FPGA-logica.

5. VOORBEREIDING

Voordat ik kon beginnen met het schrijven en beredeneren van code, heb ik eerst in kaart gebracht wat ik nodig had en mijn ontwikkelomgeving opgezet.

BENODIGDHEDEN

Om dit project te realiseren, heb ik gebruik gemaakt van de volgende componenten en tools:

HARDWARE

- **Digilent Basys 3 Artix-7 FPGA Board:** Het hart van het project.
- **VGA Monitor:** Een standaard beeldscherm (640x480 @ 60Hz).
- **VGA Kabel:** Voor de data-overdracht van bord naar scherm.
- **Micro-USB kabel:** Voor voeding en het uploaden van de bitstream.

SOFTWARE

- **AMD Xilinx Vivado (2023.2):** De IDE waarin ik de VHDL-code schrijf, simuleer en synthetiseer.

DOCUMENTATIE

- **Basys 3 Reference Manual***:* Voor het opzoeken van de pin-nummers (Constraints) van de schakelaars, LEDs en de VGA-poort.
- **VGA Timing Standards:** Tabellen met de exacte timing voor de front porch, back porch en sync pulse.

Ik ben begonnen met het installeren van Vivado en het aanmaken van een nieuw project. Hierbij was het belangrijk om de juiste FPGA-chip te selecteren, zodat de synthese correct verloopt voor het Basys 3 bord. Vervolgens heb ik de Constraints File (.xdc) gedownload en geïmporteerd, zodat ik de inputs en outputs kon koppelen aan mijn code.

6. VGA ONTDEKKEN

BASIS

Voor de basis ben ik eerst eens naar de theorieslides en componenten gaan kijken die u heeft aangehaald tijdens de theorielessen. Door naar deze blokken en structuren te gaan kijken begrijp ik al een beetje het begrip en heb ik al een zicht in de eventuele werking.

KLEUR SCHERM

Voor de eerste kennismaking met VGA en FPGA vond ik het kleuren van de schermachtergrond een goede opstart. Ik heb bij deze opzet ook meteen geopteerd voor gebruik te maken van een top file en daarin mijn andere VHDL-code te gaan plaatsen. Dit kon naar mijn mening later in dit project nog handig van pas komen om eventueel zaken te gaan koppelen aan elkaar. In deze eerste stap werk ik met een Top file en een file voor de vga die ik vga_sync noem.

Voor deze 1e opgave heb ik gemerkt dat ik een extra bibliotheek nodig heb om de vga_sync werkend te krijgen namelijk de volgende:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
```

Deze onderste bibliotheek heb ik nodig om gebruik te kunnen maken van “unsigned” binnen mijn code.

In deze vga_sync file ga ik gebruik maken van 2 invoer parameters en 6 uitvoer parameters.

```
5  entity vga_sync is
6      Port (
7          clk      : in  std_logic;
8          reset    : in  std_logic;
9          hsync    : out std_logic;
10         vsync    : out std_logic;
11         video_on : out std_logic;
12         p_tick   : out std_logic;
13         x        : out std_logic_vector(9 downto 0);
14         y        : out std_logic_vector(9 downto 0)
15     );
16 end vga_sync;
```

clk: De 100 MHz kloksignaal

reset: resetsignaal

hsync: Horizontaal synchronisatiesignaal

vsync: Verticaal synchronisatiesignaal

video_on: zichtbare gebied (1 = aan, 0 = uit)

p_tick: wanneer nieuwe pixel verwerken

x, y: De huidige pixelcoördinaten

De volgende stap waren mijn constanten deze heb ik geconfigureerd zoals hieronder weergegeven is in de afbeelding:

```
20      constant H_DISPLAY : integer := 640;
21      constant H_L_BORDER : integer := 48;
22      constant H_R_BORDER : integer := 16;
23      constant H_RETRACE : integer := 96;
24      constant H_MAX : integer := H_DISPLAY + H_L_BORDER + H_R_BORDER + H_RETRACE - 1;
25      constant START_H_RETRACE : integer := H_DISPLAY + H_R_BORDER;
26      constant END_H_RETRACE : integer := H_DISPLAY + H_R_BORDER + H_RETRACE - 1;
27
28      constant V_DISPLAY : integer := 480;
29      constant V_T_BORDER : integer := 10;
30      constant V_B_BORDER : integer := 33;
31      constant V_RETRACE : integer := 2;
32      constant V_MAX : integer := V_DISPLAY + V_T_BORDER + V_B_BORDER + V_RETRACE - 1;
33      constant START_V_RETRACE : integer := V_DISPLAY + V_B_BORDER;
34      constant END_V_RETRACE : integer := V_DISPLAY + V_B_BORDER + V_RETRACE - 1;
```

In onderstaande opsomming bespreek ik kort de keuzes van mijn constante deze bespreking is idem dito voor de verticale versie.

- H_DISPLAY: 640 pixels voor mijn zichtbare gebied.
- H_L_BORDER:
  H_R_BORDER: Niet-zichtbare gebieden aan de linker- en rechterkant.
- H_RETRACE: Duur van de horizontale synchronisatiepuls.
- H_MAX: Totaal aantal kloktikken per horizontale lijn (berekend = 799).
- START_H_RETRACE en END_H_RETRACE: Definiëren wanneer hsync laag is.

De volgende code die aanwezig is in mijn vga_sync zijn de signalen die ik gebruik.

```
36      signal pixel_reg : unsigned(1 downto 0) := (others => '0');
37      signal pixel_tick : std_logic := '0';
38      signal h_count_reg, h_count_next : unsigned(9 downto 0) := (others => '0');
39      signal v_count_reg, v_count_next : unsigned(9 downto 0) := (others => '0');
40      signal hsync_reg, vsync_reg : std_logic := '0';
41      signal hsync_next, vsync_next : std_logic;
```

In onderstaande opsomming bespreek ik kort de keuzes van mijn signalen die ik ga gebruiken.

- pixel_reg: Een 2-bit teller om de 100 MHz klok te delen naar 25 MHz.
- pixel_tick: Een puls die elke 4 klokcycli (25 MHz) wordt gegenereerd.
- h_count_reg, h_count_next: Teller voor de horizontale pixelpositie (0 tot 799).
- v_count_reg, v_count_next: Teller voor de verticale lijnpositie (0 tot 524).
- hsync_reg, vsync_reg: Huidige status van synchronisatiesignalen.
- hsync_next, vsync_next: Volgende status van synchronisatiesignalen.

de volgende zaken die aanwezig zijn in mijn code zijn 2 processen deze hebben elk hun eigen functionaliteit maar nemen wel dezelfde input parameters namelijk clk en reset.

Onderstaande vind u mijn 1e proces, dit proces heeft als hoofd functie om een pixel_tick te gaan genereren wat bedoel ik hiermee, de pixel_tick ga ik gebruiken als trigger om mijn horizontale en mijn verticale counter te gaan triggeren en dus bij gevolg de aankomende state gaan bepalen, deze implementatie volgt later nog.

```
44      -- Pixel tick (clk / 4 = 25MHz)
45      process(clk, reset)
46      begin
47          if reset = '1' then
48              pixel_reg <= (others => '0');
49          elsif rising_edge(clk) then
50              pixel_reg <= pixel_reg + 1;
51          end if;
52      end process;
53      pixel_tick <= '1' when pixel_reg = "00" else '0';
54      p_tick <= pixel_tick;
```

Het 2e proces dat ik gebruik in mijn code is het process dat ervoor gaat zorgen dat ik ga opschuiven op mijn beeldscherm, dit gaat tellen waar ik al ben geweest en waar niet. Bij een reset vallen we terug op “0”.

```
56      -- Counters
57      process(clk, reset)
58      begin
59          if reset = '1' then
60              h_count_reg <= (others => '0');
61              v_count_reg <= (others => '0');
62              hsync_reg <= '0';
63              vsync_reg <= '0';
64          elsif rising_edge(clk) then
65              h_count_reg <= h_count_next;
66              v_count_reg <= v_count_next;
67              hsync_reg <= hsync_next;
68              vsync_reg <= vsync_next;
69          end if;
70      end process;
```

In onderstaande afbeelding gaan we onze waardes die we nemen/berekenen gaan toewijzen aan onze signalen en uitgangen ik zal deze even kort aanhalen.

```
72      -- Next-state
73      h_count_next <= (others => '0') when pixel_tick = '1' and h_count_reg = to_unsigned(H_MAX, 10) else
74                      h_count_reg + 1 when pixel_tick = '1' else h_count_reg;
75      v_count_next <= (others => '0') when pixel_tick = '1' and h_count_reg = to_unsigned(H_MAX, 10) and v_count_reg = to_unsigned(V_MAX, 10) else
76                      v_count_reg + 1 when pixel_tick = '1' and h_count_reg = to_unsigned(H_MAX, 10) else v_count_reg;
77
78      -- Sync signals (active low)
79      hsync_next <= '0' when h_count_reg >= to_unsigned(START_H_RETRACE, 10) and h_count_reg <= to_unsigned(END_H_RETRACE, 10) else '1';
80      vsync_next <= '0' when v_count_reg >= to_unsigned(START_V_RETRACE, 10) and v_count_reg <= to_unsigned(END_V_RETRACE, 10) else '1';
81
82      -- Outputs
83      video_on <= '1' when (h_count_reg < to_unsigned(H_DISPLAY, 10)) and (v_count_reg < to_unsigned(V_DISPLAY, 10)) else '0';
84      hsync <= hsync_reg;
85      vsync <= vsync_reg;
86      x <= std_logic_vector(h_count_reg);
87      y <= std_logic_vector(v_count_reg);
```

- Next state
  Bij deze 2 lijnen gaan we de volgende state van de horizontale en verticale counter zetten. Dit doen we aan de hand van de eerder aangehaalde pixel_tick en de vorige waarde van het bijhorende count register. We bekijken hiervan de momentele toestand en aan de hand daarvan gaan we de volgende toestand bepalen.

- Sync signals
  Deze regels genereren de sync-pulsen: hsync wordt 96 klokcycli laag na elke lijn, vsync wordt 2 lijnen laag na elk frame, precies wat een VGA-monitor nodig heeft om te weten wanneer een nieuwe lijn of nieuw scherm begint.

- Outputs

  Hierbij verwijs ik simpel weg mijn signalen door naar de juiste uitgangen.

Tussenresultaat:

SCHERMFORMAAT

Het doel van deze tussenstap is het actieve tekengebied beperken tot 440x440 px met een grijze rand en zwart speelveld.

Voor deze tussenstap was er niet heel veel verandering nodig, uitsluitend in onze Top file moeten we een aantal zaken gaan aanpassen die ik hier voor jullie ga bespreken.

Om te beginnen hebben we een extra bibliotheek nodig om dit tot een goed eind te laten komen.

```
use IEEE.NUMERIC STD.ALL;
```

Deze bibliotheek heb ik nodig om de 'to_integer' methode te kunnen gebruiken.

Vervolgens declareren we de constanten die we zelf het liefst willen voor onze borders. Hierbij heb ik gekozen voor 100 en 20 om een mooi vierkant te bekomen.

```
29  constant hBorder : integer := 100;   -- 100 px links/rechts
30  constant vBorder : integer := 20;    -- 20 px boven/onder
```

Berekening vierkant:

$$640\text{Hpx} - (2 * 100\text{Hpx}) * 480Vpx - (2 * 20Vpx) = 440Hpx * 440Vpx$$

Vierkant 440 x 440px

Als we deze waardes gaan gebruiken in ons gewijzigd 'video_on' process dan komen we tot de gewenste uitkomst.

```
41  process(video_on, x, y)
42  begin
43      if video_on = '1' then
44          -- Binnen actief speelveld?
45          if to_integer(unsigned(x)) > hBorder and
46             to_integer(unsigned(x)) < 640 - hBorder and
47             to_integer(unsigned(y)) > vBorder and
48             to_integer(unsigned(y)) < 480 - vBorder then
49              rgb <= "000000000000";   -- ZWART = speelveld (achtergrond)
50          else
51              rgb <= "011101110111";   -- GRIJS = border (50% helderheid)
52          end if;
53      else
54          rgb <= (others => '0');      -- ZWART buiten video_on
55      end if;
56  end process;
57  Behavioral;
```

Hierbij gaan we de bovenstaande berekening gaan implementeren in de alreeds bestaande functie 'video_on' bekomen we bovenstaande code. We gaan in deze code gaan kijken naar de waardes van onze variabelen x en van y en deze bepalen of de pixel zwart of grijs word getoond daarom is het van cruciaal belang deze ook mee te geven aan het process.

(ik heb hierbij mijn rode kleur veranderd door grijs en zwart omdat het anders minder duidelijk weer te geven was.)

Tussenresultaat:

LIJN TONEN

Om 3 op één rij te kunnen spelen hebben we een oppervlakte nodig waar we kunnen gaan spelen. Om dit te gaan realiseren was ik aan het denken aan een raster gemaakt door 4 lijnen over ons eerder gemaakt zwart vierkant.

Om deze stap wat kleiner te maken ga ik eerst één lijn proberen maken die ik dan eenvoudig ga kunnen aanpassen om later mijn raster te maken.

Om dit te gaan realiseren maak ik gebruik van 2 extra constante variabelen.

```
33      constant lineWeight : integer := 2;
34      constant hLinePos1  : integer := vBorder + 147;
```

LineWeight gaat aangeven hoe breed we onze lijn willen hierbij heb ik 2 gekozen dit wordt dan 4 → 2 boven, 2onder.

hLinePos1 ga ik gebruiken om te gaan bepalen waar ik deze lijn wil gaan positioneren. Mijn redenering voor de positie is 'vBorder + 147' om deze wat te verduidelijken kunnen we deze berekening ook schrijven als '$vBorder + \frac{1}{3}\ van\ het\ zelf\ getekende\ vierkant\ (440)$' zo kom ik aan de waarde 147.

Om deze lijn dan ook effectief te gaan tekenen moeten we wederom ons 'video_on' process gaan aanpassen.

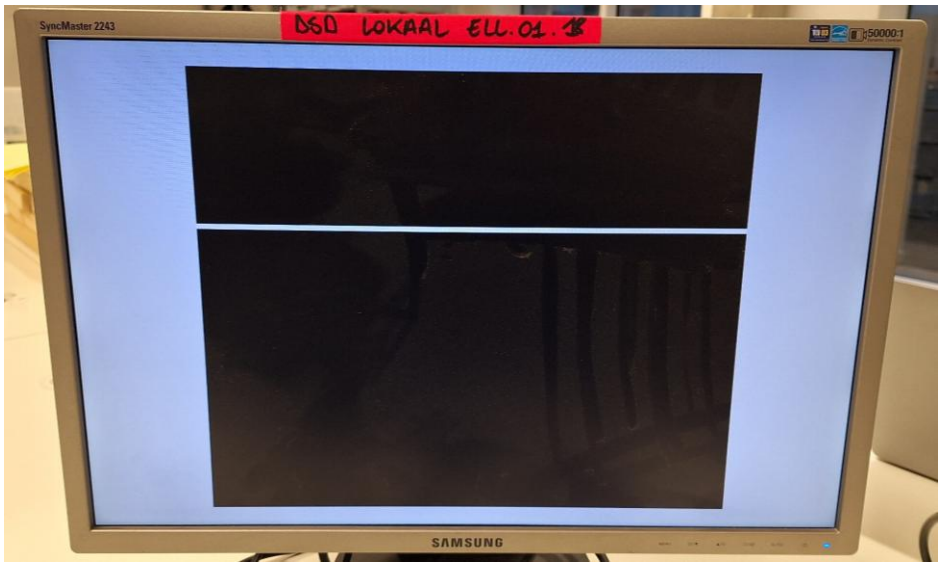
```
45      process(video_on, x, y)
46          variable y_int : integer;
47          variable x_int : integer;
48      begin
49          y_int := to_integer(unsigned(y));
50          x_int := to_integer(unsigned(x));
51
52          if video_on = '1' then
53              -- Binnen speelveld?
54              if x_int > hBorder and x_int < 640 - hBorder and
55                 y_int > vBorder and y_int < 480 - vBorder then
56
57                  -- HORZONTALE LIJN TONEN
58                  if y_int >= hLinePos1 - lineWeight and
59                     y_int <= hLinePos1 + lineWeight then
60                      rgb <= "111111111111";  -- WIT = lijn
61                  else
62                      rgb <= "000000000000";  -- ZWART = achtergrond
63                  end if;
64
65              else
66                  rgb <= "011101110111";  -- GRIJS = border
67              end if;
68          else
69              rgb <= (others => '0');
70          end if;
71      end process;
```

Hierbij gaan we kijken of we ons binnen het zwarte vierkant bevinden en dan gaan we kijken of we binnen de dikte en de positie van de lijn zijn , indien dit allemaal het geval is kleuren we wit. Al de andere pixels blijven hetzelfde als voorheen.

Ter verduidelijking een leesbaar inzicht met getallen:

```
if in_speelveld then
   if y >= 165 and y <= 169 then
      rgb <= wit;     -- LIJN
   else
      rgb <= zwart;   -- ACHTERGROND
   end if;
else
   rgb <= grijs;
end if;
```

Tussenresultaat:



RASTER MAKEN

Nu dat ik wist hoe ik één lijn kon maken was ik ook instaat om een raster te gaan ontwikkelen. Dit was door mijn tussenstap niet zo moeilijk, ik moest gaan bepalen waar ik mijn lijnen wou gaan positioneren in onderstaande tabel kan je de waardes vinden die ik heb gebruikt deze heb ik berekend* zoals ik bij mijn 1e lijn alreeds heb aangehaald.

(*)440 px / 3 = 146.67 → afronding op 147 px per cel

Lijn	Positie	Berekening
hLinePos1	**y = 167**	20 + 147
hLinePos2	**y = 314**	20 + 294
vLinePos1	**x = 247**	100 + 147
vLinePos2	**x = 394**	100 + 294

Deze waardes gaan we dan ook in een constante gaan steken zodat ik deze kan gebruiken.

```
34      constant hLinePos1  : integer := vBorder + 147;
35      constant hLinePos2  : integer := vBorder + 294;
36      constant vLinePos1  : integer := hBorder + 147;
37      constant vLinePos2  : integer := hBorder + 294;
```

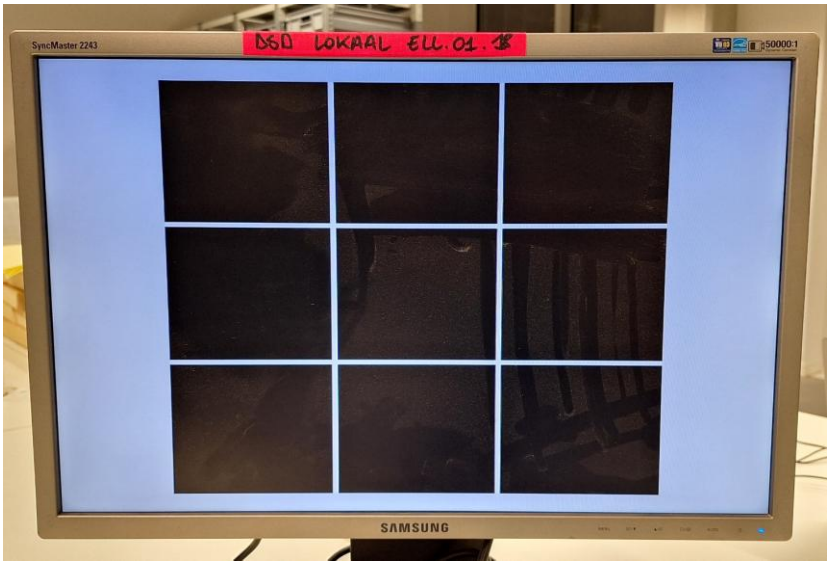
Om deze te gaan tekenen heb ik voor mijzelf weer een eenvoudige logica gemaakt.

```
if y in hLinePos1 or hLinePos2 then
   rgb <= wit;
elsif x in vLinePos1 or vLinePos2 then
   rgb <= wit;  -- Verticale lijnen winnen
else
   rgb <= zwart;
end if;
```

Om deze dan ook weer effectief te gaan weergeven moeten we zoals hierboven in de vorige stap al besproken deze gaan tekenen. Dit gaan we doen op dezelfde manier als ik alreeds besproken heb namelijk door gebruik te maken van ‘if’ en ‘else’ statements.

```
48      process(video_on, x, y)
49          variable y_int : integer;
50          variable x_int : integer;
51          variable draw_line : boolean := false;
52      begin
53          y_int := to_integer(unsigned(y));
54          x_int := to_integer(unsigned(x));
55
56          draw_line := false;
57
58          if video_on = '1' then
59              -- Binnen speelveld?
60              if x_int > hBorder and x_int < 640 - hBorder and
61                 y_int > vBorder and y_int < 480 - vBorder then
62
63                  -- HORZONTALE LIJNEN
64                  if (y_int >= hLinePos1 - lineWeight and y_int <= hLinePos1 + lineWeight) or
65                     (y_int >= hLinePos2 - lineWeight and y_int <= hLinePos2 + lineWeight) then
66                      draw_line := true;
67
68                  -- VERTICALE LIJNEN (heeft prioriteit)
69                  elsif (x_int >= vLinePos1 - lineWeight and x_int <= vLinePos1 + lineWeight) or
70                        (x_int >= vLinePos2 - lineWeight and x_int <= vLinePos2 + lineWeight) then
71                      draw_line := true;
72                  end if;
73
74                  -- Teken lijn of achtergrond
75                  if draw_line then
76                      rgb <= "111111111111";  -- WIT = rasterlijn
77                  else
78                      rgb <= "000000000000";  -- ZWART = achtergrond
79                  end if;
80
81              else
82                  rgb <= "011101110111";  -- GRIJS = border
83              end if;
84          else
85              rgb <= (others => '0');
86          end if;
87      end process;
```

Tussenresultaat:



VIERKANT MAKEN

De volgende stap die ik nu ga implementeren is mijn eerste tekening van een vierkantje binnen in een cel zodanig dat 1 van de spelers een eigensoort ‘stempel’ heeft om in een vakje te plaatsen.

Een belangrijk aspect binnen deze stap is dat we moeten weten waar een cel begint en eindigt zodat we ons vierkant daar aan kunnen aanpassen. Hiervoor heb ik een nieuwe constante gedefinieerd namelijk:

```
38      constant sqBorder       : integer := 30;
```

Waarom sqBorder = 30?

Ik heb een aantal waardes voor sqBorder gebruikt bij te grote waardes raakte het vierkant de rasterlijnen en werd het overschreven door wit. Bij te kleine waardes was het niet echt goed zichtbaar. Door de waarde te verhogen naar 30 komt er een duidelijke zwarte rand rondom het vierkant.

Coördinaten van het rode vierkant (cel 1)

Links : x > 100 + 30 = 130

Rechts : x < 247 – 30 = 217

Boven : y > 20 + 30 = 50

Onder : y < 167 – 30 = 137

Om tot slot mijn vierkant te tonen heb ik een extra variabele aan gemaakt die ik kan gaan gebruiken om te bepalen of ik rood moet kleuren of niet.

```
60          variable draw_square  : boolean := false;
```

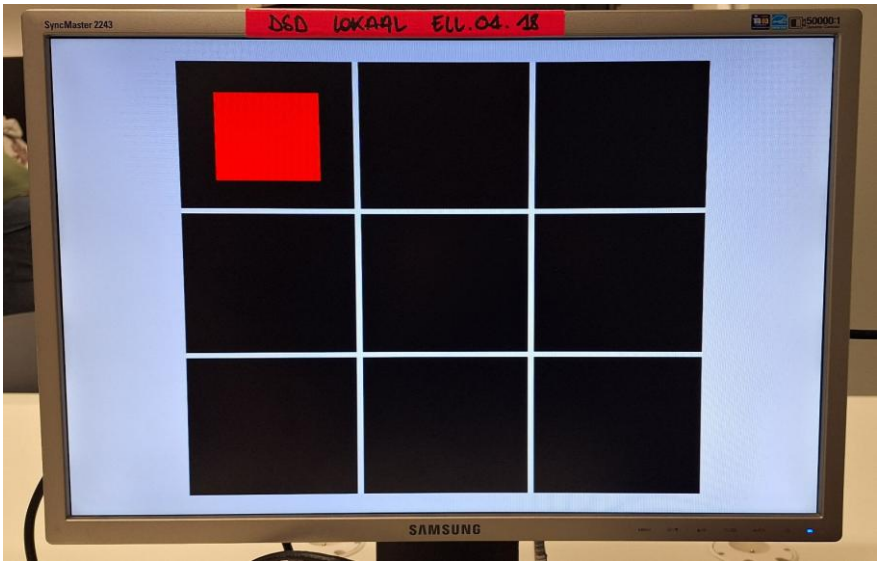
We passen wederom onze if-statement aan in ons video_on process om ons rood vierkant te gaan implementeren dit doen we als volgt.

```
88          if draw_line then
89              rgb <= "111111111111";          -- WIT = rasterlijnen (winnen altijd)
90          elsif draw_square then
91              rgb <= "111100000000";          -- ROOD = vierkant cel 1
92          else
93              rgb <= "000000000000";          -- ZWART = achtergrond
94          end if;
```

Het rode vierkant verschijnt duidelijk in de linkerboven-cel met zwarte ruimte rondom.

De witte rasterlijnen blijven onaangetast en lopen er netjes omheen. Dit bevestigt dat de coördinatenberekeningen aanvaardbaar zijn

Tussenresultaat:



KRUISJE MAKEN

Het doel van deze stap is een realistisch, groot en vet rood kruisje (X) tekenen in cel 1 dat onze 2e 'stempel' voorstelt. Dit is voorlopig de laatste stap van de grafische opbouw voordat we naar de echte modulaire architectuur gaan.

Voor deze stap te implementeren moeten er zoals altijd een aantal aanpassingen worden gedaan aan de code.

Nieuwe constanten toegevoegd (parameters voor het kruisje)

```
38      constant sqBorder       : integer := 30;
39      constant crossThickness : integer := 22;
```

Het kruisje wordt volledig binnen cel 1 getekend met behulp van twee diagonale vergelijkingen:

- Diagonaal 1: van linksboven naar rechtsonder
- Diagonaal 2: van rechtsboven naar linksonder

Ik maak gebruik van de abs-functie voor een zeer leesbare en efficiënte detectie van beide diagonalen:

```
72                  -- 2. KRUISJE in cel 1
73                  elsif x_int > hBorder + sqBorder and x_int < vLinePos1 - sqBorder and
74                        y_int > vBorder + sqBorder and y_int < hLinePos1 - sqBorder then
75
76                      -- Diagonaal 1: linksboven → rechtsonder
77                      if abs( (x_int - (hBorder + sqBorder)) - (y_int - (vBorder + sqBorder)) ) < crossThickness then
78                          draw_x := true;
79                      -- Diagonaal 2: rechtsboven → linksonder
80                      elsif abs( (x_int - (hBorder + sqBorder)) + (y_int - (vBorder + sqBorder))
81                                 - (hLinePos1 - sqBorder - (vBorder + sqBorder)) ) < crossThickness then
82                          draw_x := true;
83                      end if;
84                  end if;
```

Deze methode om het kruis te bepalen was een zeer pittige opgave ik ga deze op de volgende pagina proberen eenvoudig te bespreken.

Stap-voor-stap uitleg

We definiëren eerst de linkerbovenhoek van de cel (dus binnen de zwarte rand):

```
vhdlleft := hBorder + sqBorder;    -- bijv. 100 + 30 = 130
top  := vBorder + sqBorder;    -- bijv. 20 + 30  = 50
cel_hoogte := hLinePos1 - sqBorder - (vBorder + sqBorder);  -- 167 - 30 - 50 = 87
```

Eerste diagonaal: linksboven → rechtsonder

Op een perfecte \ diagonaal geldt:

```
text(x − left) = (y − top)
Dus:
text(x − left) − (y − top) = 0
```

→ Als een pixel dicht bij die lijn ligt, is dat verschil klein.

We maken hem dik door te zeggen:

```
vhdlabs( (x_int - left) - (y_int - top) ) < crossThickness
```

Voorbeeld met crossThickness = 22:

Als het verschil tussen -22 en +22 ligt → pixel hoort bij de dikke lijn

Resultaat: een 44 pixels dikke schuine streep van linksboven naar rechtsonder

Tweede diagonaal: rechtsboven → linksonder

Op een perfecte / diagonaal geldt:

```
text(x − left) + (y − top) = cel_hoogte
```

Want: rechtsboven = (grote x, kleine y) → som is groot

linksonder = (kleine x, grote y) → som is ook groot

Dus:

```
text(x − left) + (y − top) − cel_hoogte = 0
```

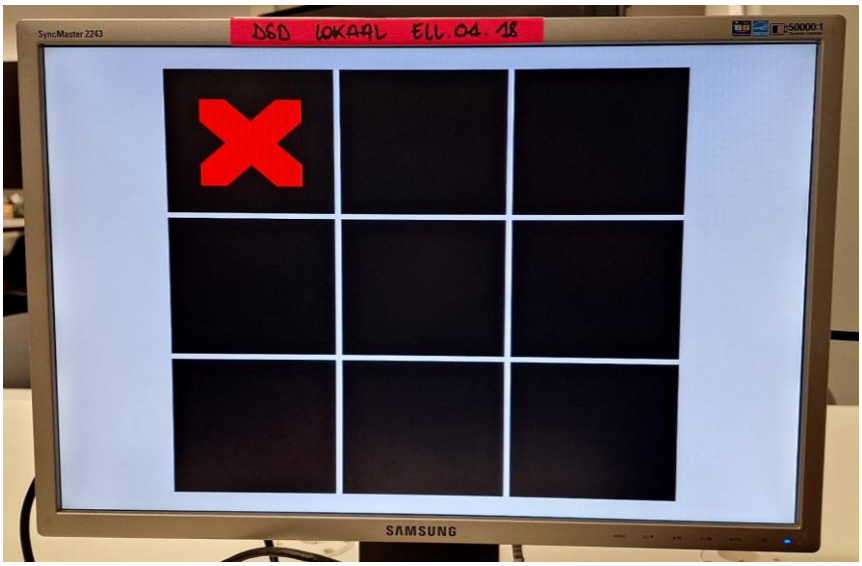
Weer: als het verschil klein is → pixel ligt op de lijn.

```
vhdlabs( (x_int - left) + (y_int - top) - cel_hoogte ) < crossThickness
```

→ Dit geeft een dikke / streep

Diagonaal	Voorwaarde (perfecte lijn)	Dikke lijn (abs < dikte)
\	(x−left) = (y−top)	abs((x−left) − (y−top)) < 22
/	(x−left) + (y−top) = cel_hoogte	abs((x−left) + (y−top) − cel_hoogte) < 22

Tussenresultaat:



7. DE ARCHITECTUUR OPSTELLEN/BEDENKEN

In het begin had ik alles in één grote top.vhd: raster, vierkant, kruisje, coördinaten, alles. Dat werkte prima voor de eerste stappen, maar al snel merkte ik de nadelen:

- De code werd onoverzichtelijk en steeds langer
- Als ik iets wilde veranderen (bijv. dikte van X), moest ik door honderden regels scrollen
- Ik zou later 9 kruisjes/bolletjes moeten tekenen → dan werd het pure copy-paste-hel
- Bij debuggen was het bijna onmogelijk om te zien waar een fout precies zat

Daarom heb ik mezelf de volgende vragen gesteld (die ik ook echt op papier heb gezet tijdens de les):

1. Wat doet precies hetzelfde voor alle 9 cellen?

- Het tekenen van X en vierkant, de state machine (leeg → X → O)
- Conclusie: dit moet in een aparte, herbruikbare cel-module!

2. Wat is uniek per cel?

- Alleen de positie op het scherm (x-offset en y-offset)
- Dus de cel-module krijgt zijn positie mee als generic of signaal

3. Wie bepaalt welke cel geselecteerd is en wie aan de beurt is?

- Dat is pure spel-logica (switches, knoppen, beurt, win-detectie)
- Dit hoort absoluut niet in de tekenmodule → aparte game_logic.vhd

4. Wie zorgt voor de VGA-timing?

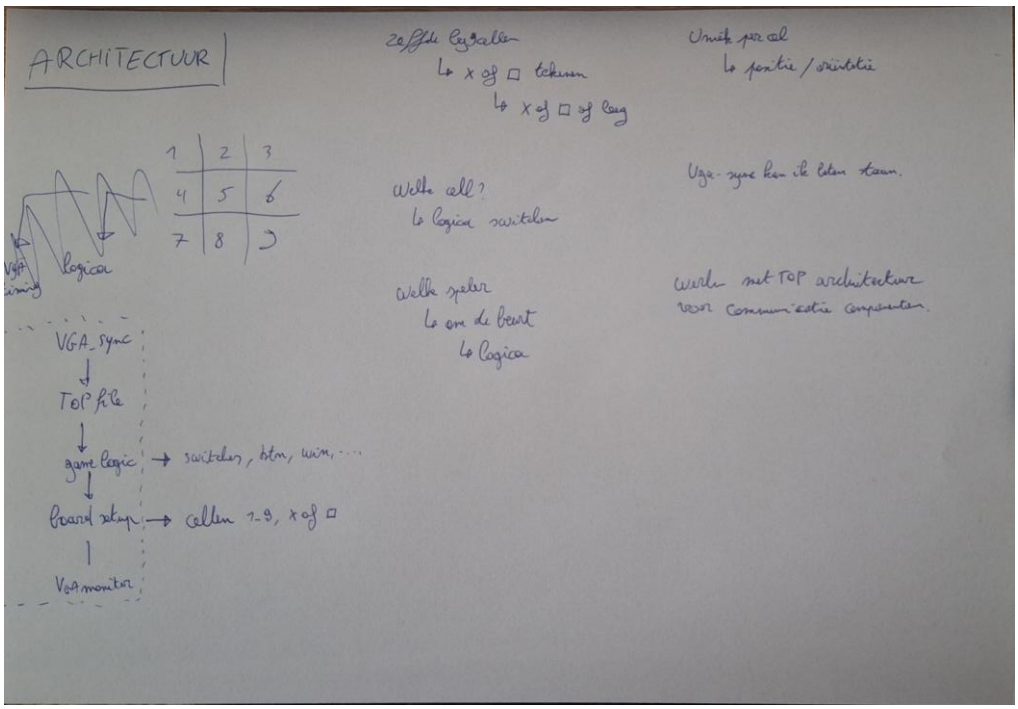
- vga_sync.vhd werkt al perfect en heeft niets met het spel te maken, laten zoals het is

5. Hoe breng ik al die modules samen zonder chaos?

- Een nieuwe, schone top.vhd die alleen maar draden verbindt
- Geen tekenlogica meer in top → alleen instantiëring en klok/reset

Schets op papier

Ik heb tijdens de les eerst een ruw blokdiagram getekend op een blaadje en gebruikt als basis voor de definitieve architectuur.



Waarom deze architectuur mijn keuze is:

- Zeer makkelijk debuggen: als cel 5 niet werkt, kijk ik alleen in cell.vhd
- Ligt in de lijn van de structuur die ik door de voorbije jaren heb geleerd
- Perfect te presenteren en uit te leggen

Dankzij deze denkoefening zie ik de tussenstappen die ik nodig heb voor een vlotte overgang.

8. DE MODULES

CELMODULE

Na de grafische opbouw in één grote top.vhd (hoofdstuk 6) was het tijd om over te schakelen naar een professionele, modulaire architectuur.

De eerste stap hierin was het uit elkaar halen van mijn code: alle tekenlogica voor symbolen in één cel zowel mijn rode vierkant (stap 6.6) als mijn grote rode kruisje (stap 6.7) – moest uit top.vhd verdwijnen en in een aparte, herbruikbare module komen: cell.vhd.

Tijdens het werken met één grote top.vhd merkte ik al snel de beperkingen:

- De file werd steeds langer en onoverzichtelijk
- Elke aanpassing aan het uiterlijk van het kruisje (dikte, kleur, border) moest in één grote process
- Bij het later toevoegen van 9 cellen zou ik 9 keer dezelfde code moeten kopiëren → pure copy-paste-hel
- Debuggen werd steeds moeilijker: waar zit de fout precies?

Daarom heb ik besloten om alle tekenlogica van één enkele cel (mijn grote rode kruisje en vierkant) volledig uit top.vhd te halen en in een aparte module te plaatsen.

1. Nieuwe source toevoegen
    a. Add Sources → Create new file → VHDL Module → naam: `cell`

2. Tekenlogica
    a. Beide symbolen overgenomen uit mijn oude top.vhd:
        i. Het rode vierkant met (uit stap 6.6)
        ii. Het grote rode kruisje (uit stap 6.7)

3. Generics POS_X en POS_Y toegevoegd
    a. Dit is cruciaal: elke cel krijgt zijn eigen startpositie mee → de module is volledig herbruikbaar.

4. State machine geïmplementeerd (nog niet volledig gebruikt, maar wel al aanwezig)
    a. Aanwezigheid vereist om later te gaan bepalen wat en wanneer er mag worden getekend

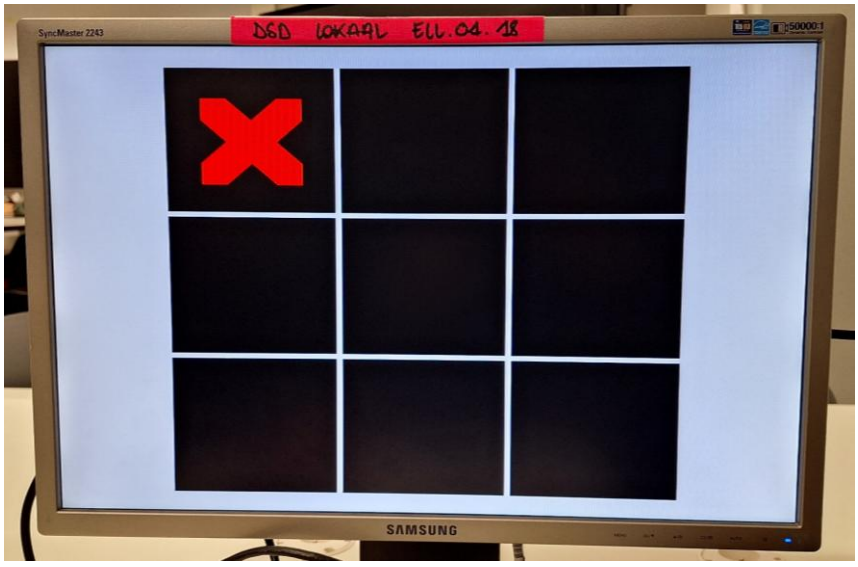
5. Tijdelijke top.vhd gebruikt om slechts één cel te testen op positie
    a. Dit was voor mij persoonlijk een duidelijke tussenstap

6. Prioriteit correct ingesteld: rasterlijnen (in top.vhd) winnen van cel-symbolen
    a. Dit is nodig om een duidelijke fout te kunnen zien tijdens het ontwikkelen

In deze tussenstap heb ik bewust slechts één cel op het scherm getoond (cel 1, linksboven). Voor de input heb ik tijdelijk de center knop (btnC) gebruikt als bevestigingsknop in plaats van de 9 switches.

Reden: zo kon ik 100 % zeker zijn dat de state machine en tekenlogica werken, zonder verwarring door meerdere switches tegelijk te gebruiken. De bovenste knop (btnC) is rechtstreeks verbonden met de `confirm`-ingang van de cel. Selectie van de cel gebeurde via een vaste ‘1’ (hardcoded) later worden de switches gebruikt.

Met de succesvolle creatie en validatie van cell.vhd is de basis gelegd voor een professionele, modulaire architectuur. De module bevat al mijn tekenlogica (vierkant én kruisje), is volledig herbruikbaar en 100 % gevalideerd. Deze stap is essentieel voor de schaalbaarheid van het project.



9. 9 CELLEN + INPUT

Na de succesvolle validatie van één enkele cel in hoofdstuk 8 was de volgende logische stap het instantiëren van alle 9 cellen tegelijk, zodat het volledige bord zichtbaar en bespeelbaar werd maar nu volledig modulair opgebouwd.

IMPLEMENTATIE IN VIVADO 9 CELLEN

1. Nieuwe module aangemaakt: board.vhd
2. Gebruik van een generate-loop om 9 identieke instanties van cell.vhd te creëren

```
    gen_cells: for i in 0 to 8 generate
        constant row : integer := i / 3;
        constant col : integer := i mod 3;
        constant pos_x : integer := hBorder + col * CELL_SIZE;
        constant pos_y : integer := vBorder + row * CELL_SIZE;
    begin
        cell_i: entity work.cell
            generic map (POS_X => pos_x, POS_Y => pos_y)
            port map (
                clk       => clk,
                reset     => reset,
                sel       => sw(i),
                turn      => '0',
                confirm   => btnC,
                pixel_x   => pixel_x,
                pixel_y   => pixel_y,
                video_on  => video_on,
                rgb_out   => cell_rgb(i),    -- elke cel heeft zijn eigen draad!
                state_out => open
            );
    end generate;
```

3. Posities berekend met:
    a. hBorder = 100, vBorder = 20
    b. CELL_SIZE = 147
    c. Rij en kolom bepaald via i / 3 en i mod 3
4. Rasterlijnen overgenomen uit mijn oude top.vhd en in board.vhd geplaatst (met hoogste prioriteit)

```
    -- 2. RASTERLIJNEN
    process(video_on, pixel_x, pixel_y)
        variable x_int, y_int : integer;
    begin
        grid_rgb <= "000000000000";
        x_int := to_integer(unsigned(pixel_x));
        y_int := to_integer(unsigned(pixel_y));

        if video_on = '1' then
            if (y_int >= vBorder + 147 - lineWeight and y_int <= vBorder + 147 + lineWeight) or
               (y_int >= vBorder + 294 - lineWeight and y_int <= vBorder + 294 + lineWeight) or
               (x_int >= hBorder + 147 - lineWeight and x_int <= hBorder + 147 + lineWeight) or
               (x_int >= hBorder + 294 - lineWeight and x_int <= hBorder + 294 + lineWeight) then
                grid_rgb <= "111111111111";
            end if;
        end if;
    end process;
```

5. Achtergrond (grijze rand + zwart speelveld) opnieuw geïmplementeerd in board.vhd om volledige controle te behouden

```
        -- Achtergrond: grijze rand + zwart speelveld
        if video_on = '1' then
            if x_int > hBorder and x_int < 640-hBorder and
               y_int > vBorder and y_int < 480-vBorder then
                temp_rgb := "000000000000";  -- zwart
            else
                temp_rgb := "111111111111";  -- grijs
            end if;
        else
            temp_rgb := "000000000000";
        end if;
```

Belangrijke ontwerpkeuzes

1. Elke cel krijgt zijn eigen rgb-signaal
    a. tijdens mijn ontwerp heb ik te maken gekregen met multiple driver fouten

2. Prioriteitvolgorde: rasterlijnen > cel-symbolen > achtergrond

3. Switches (sw[8:0]) direct verbonden met de sel-ingangen van de cellen

4. Bevestiging via center knop (btnC) – gemeenschappelijk voor alle cellen

INPUT-IMPLEMENTATIE: SWITCHES EN BEVESTIGINGSKNOP

Vanaf deze stap (9 cellen) wordt de input van de speler volledig geïmplementeerd met de hardware-knoppen van de Basys 3:

- De 9 switches (sw[0] t/m sw[8]) zijn rechtstreeks verbonden met de `sel`-ingang van de bijbehorende cel in de generate-loop van board.vhd.
- Switch 0 selecteert cel 1 (linksboven)
- Switch 1 selecteert cel 2
- ...
- Switch 8 selecteert cel 9 (rechtsonder)

- De center knop (btnC) is verbonden met de `confirm`-ingang van alle 9 cellen tegelijk.

Werking: de speler zet een switch aan (cel kiezen), drukt daarna op de center knop deze dient als bevestiging het programma zet dan de rode X in de gekozen cel.

Tussenresultaat

Na synthese en programmering verscheen het volledige 3x3 speelbord weer op het scherm met:

- 9 cellen, perfect uitgelijnd

- Exact dezelfde stijl als in hoofdstuk 6: grijze rand, zwarte achtergrond, witte rasterlijnen

- Mijn grote rode kruisje (sqBorder = 30, crossThickness = 22) verschijnt in de geselecteerde cel bij gebruik van de bijbehorende switch + center knop

10. DE SPELLOGICA

Na het succesvol implementeren van de grafische weergave (het bord) en de basis-input, kwam ik bij het meest complexe deel van het project: de game_logic.vhd. Dit is het "brein" van de FPGA. Waar de board module alleen maar pixels kleurt, bepaalt de game_logic de regels, de beurten en de geldigheid van zetten.

Voordat de logica kan werken, is er een constante communicatie nodig tussen de modules. Dit heb ik gerealiseerd via een feedback-loop: de board-module houdt de visuele data bij en stuurt dit als een 18-bits vector (cells_state) naar de game_logic. De game_logic analyseert dit en stuurt commando's (confirm, turn) terug.

Ik heb hier drie specifieke uitdagingen moeten oplossen die cruciaal zijn voor een soepele spelervaring.

KNOPPEN (INPUT_SW) & SMART SWITCH VALIDATIE

Een fysieke schakelaar op het Basys 3 bordje heeft een mechanisch geheugen. Als ik schakelaar 0 omhoog zet voor de eerste zet, blijft die fysiek omhoog staan.

Het probleem:
Speler 1 zet schakelaar 0 omhoog (voor vakje 1) en bevestigt. Er verschijnt een X. Speler 2 is aan de beurt en zet schakelaar 1 omhoog. Op dat moment ziet de FPGA dat zowel schakelaar 0 als schakelaar 1 aan staan (logische '11'). De FPGA kan niet weten welke van de twee de "nieuwe" zet is.

De Oplossing:
Software Masking Om dit op te lossen heb ik een slim filter geprogrammeerd. Ik kijk niet alleen naar de schakelaars (sw), maar ik vergelijk deze real-time met de status van het bord (cells_state).

De logica die ik heb geschreven werkt als volgt: "Een schakelaar is alleen geldig als hij AAN staat ÉN het bijbehorende vakje op het bord nog LEEG ('00') is."

```
54      process(sw, cells_state)
55          variable count : integer;
56      begin
57          count := 0;
58          for i in 0 to 8 loop
59              if cells_state(2*i+1 downto 2*i) = "00" then
60                  -- Vakje is leeg, switch telt mee
61                  sw_valid(i) <= sw(i);
62                  if sw(i) = '1' then count := count + 1; end if;
63              else
64                  -- Vakje is bezet, switch negeren
65                  sw_valid(i) <= '0';
66              end if;
67          end loop;
68          sw_count <= count;
69      end process;
```

In de code heb ik dit geïmplementeerd door een lus (for-loop) te maken die over alle 9 schakelaars gaat. Als cells_state voor een positie al gevuld is, forceer ik de interne waarde van die schakelaar naar '0'. Hierdoor hoeft de speler de oude schakelaars niet terug naar beneden te doen; het systeem negeert ze automatisch.

SELECTIE EN BEVEILIGING

Een gebruiker mag het spel niet kunnen laten vastlopen of valsspelen door verkeerde combinaties in te drukken.

Ik heb een strenge beveiliging ingebouwd die voorkomt dat een beurt wordt verwerkt als er geen duidelijke keuze is gemaakt. Dit werkt op basis van een AND-poort logica. Een zet wordt pas geaccepteerd als aan twee voorwaarden tegelijk is voldaan:

1. De Bevestigingsknop (btnC) is ingedrukt (btn_pressed = '1').
2. Het aantal geldige schakelaars (sw_count) is precies gelijk aan 1.

Ik heb een teller (sw_count) gemaakt die telt hoeveel van de gemaskeerde "Smart Switches" op '1' staan.

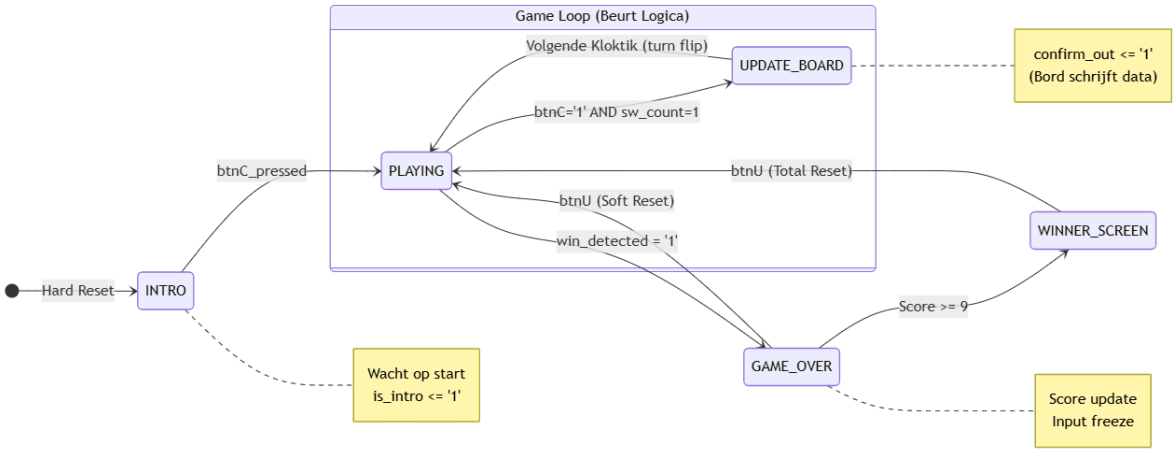
- Als sw_count = 0 (geen keuze): De knopdruk wordt genegeerd.
- Als sw_count > 1 (meerdere vakjes tegelijk): De knopdruk wordt genegeerd en het error-signaal wordt geactiveerd.

```
152             if btn_pressed = '1' and sw_count = 1 then
153                 confirm_out <= '1'; -- Schrijf naar bord
154                 current_state <= UPDATE_BOARD; -- Ga naar tussenstap
155             end if;
```

BEURTBEHEER (TURN SIGNAAL & STATE MACHINE)

Het simpelweg omwisselen van de beurt (turn <= not turn) zorgde in de eerste versies voor een grafische fout (een "glitch"). De FPGA rekent zo snel (in nanoseconden) dat hij de beurt wisselde *terwijl* hij nog bezig was met het schrijven van het symbool naar het bord. Hierdoor verscheen soms het symbool van de *volgende* speler in het vakje van de *huidige* speler.

Om dit op te lossen heb ik een Finite State Machine (FSM) ontworpen.



Deze machine regelt niet alleen de glitches, maar de volledige flow van het spel. De statussen zijn:

1. INTRO: Het spel start hier. Het wacht op een startcommando voordat het speelveld actief wordt.
2. PLAYING: Het spel wacht op input (Switch + Button).
3. UPDATE_BOARD : In deze stap houd ik het signaal confirm hoog. Het bord ziet dit en tekent het symbool van de huidige speler. De beurt wisselt hier nog NIET.
4. PLAYING (Terugkeer): Pas nadat de cel veilig gevuld is, wisselen we de beurt en gaan we terug.
5. WINNER_SCREEN: Zodra een winnaar is gedetecteerd, wordt de input bevroren en tonen we het eindscherm.

Dit zorgt voor een stabiel en deterministisch systeem.

11. DE GAME STATUS

Naast het regelen van de invoer, moet het systeem continu controleren wat de status van het spel is: heeft iemand gewonnen?

WINNENDE COMBINATIES

Omdat een FPGA alles parallel verwerkt, controleer ik alle 8 mogelijke win-lijnen tegelijkertijd in één klokcyclus.

Om dit programmeerbaar te maken, moest ik een data-conversie toepassen. De input is een lineaire 18-bits vector (cells_state). In de code zet ik deze om naar een interne Array (Grid 0 tot 8). Hierdoor kan ik in de code logisch verwijzen naar vakjes.

Ik heb de status van het bord omgezet naar een leesbare matrix-structuur.

De logica werkt met combinatorische vergelijkingen: Winst op Rij 1 = (Vakje 0 == Vakje 1) EN (Vakje 1 == Vakje 2) EN (Vakje 0 is NIET leeg)

Zodra één van deze vergelijkingen 'WAAR' is, wordt het signaal win_detected hoog en wordt de winnaar ("01" voor X of "10" voor O) opgeslagen.

CELLEN CONTROLE

Wanneer de status GAME_OVER bereikt wordt, stuur ik een signaal naar de text_display module. Omdat een FPGA geen ingebouwde lettertypes heeft zoals een computer, heb ik deze zelf moeten "tekenen" op bit-niveau in VHDL.

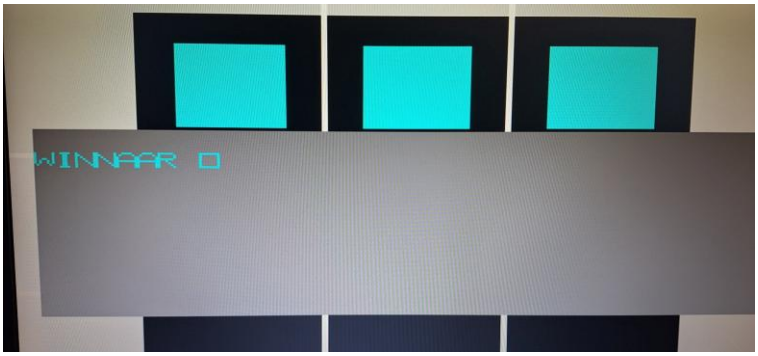
Ik heb een bitmap gemaakt waarbij elke letter een rooster van 8x8 bits is. Voor de winnaar "vierkant" heb ik een speciaal karakter ontworpen: F_Sq (Een vierkantje), omdat mijn speler 2 met vierkantjes speelt.

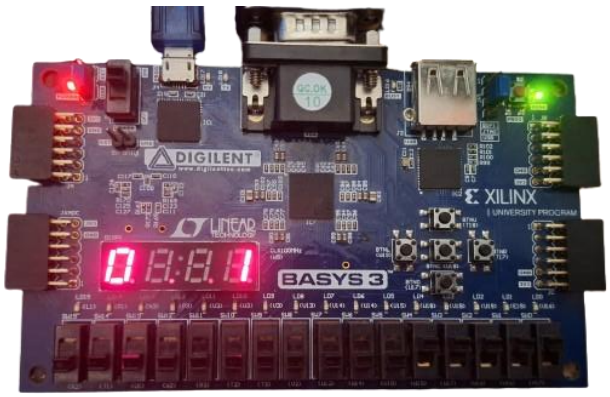
```
34      constant F_A  : font_row := ("00111100","01000010","10000001","10000001","11111111","10000001","10000001","10000001");
35      constant F_B  : font_row := ("11111100","10000010","10000010","11111100","10000010","10000010","10000010","11111100");
36      constant F_C  : font_row := ("00111100","01000010","10000000","10000000","10000000","10000000","01000010","00111100");


64      constant F_Sq : font_row := ("11111111","10000001","10000001","10000001","10000001","10000001","10000001","11111111");
```

OUTPUT VAN WIN_STATE EN KLEUR

Zodra er een winnaar is:

1. Input Freeze: De knoppen worden genegeerd.
2. Tekst: Afhankelijk van wie gewonnen heeft, wordt de tekstkleur aangepast: Cyaan voor "X WINT" en Rood voor "VIERKANT WINT".
3. Score: De scoreteller op het 7-segment display wordt opgehoogd.



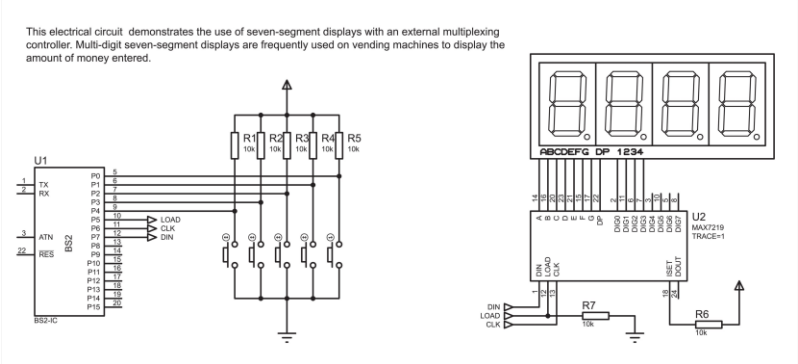


DE SCORE WEERGAVE (7-SEGMENT MULTIPLEXING)

Om de stand (Score X vs Score O) bij te houden, wilde ik gebruik maken van de 7-segment displays op het Basys 3 bord. Dit leek in eerste instantie simpel: zet een getal op de uitgang en klaar. Echter, de hardware-architectuur van deze displays vereist een complexe aansturing die Time Multiplexing heet.

Het Hardware Dilemma Het bordje heeft 4 displays (digits), maar slechts één set van 7 pinnen voor de segmenten . Dit betekent dat alle 4 de displays fysiek aan elkaar geknoopt zijn.

- Als ik een "1" naar de segmenten stuur, en ik zet alle displays aan, dan staat er op alle vier de displays een "1" (1111).
- Het is fysiek onmogelijk om tegelijkertijd een "3" op het linker display en een "5" op het rechter display te zetten.



De Oplossing:
Time Multiplexing Om toch verschillende scores te tonen (bijvoorbeeld "3" voor X links en "5" voor O rechts), maak ik gebruik van de traagheid van het menselijk oog (Persistence of Vision). Ik heb een proces geschreven dat razendsnel wisselt tussen de displays:

1. Zet segmenten klaar voor getal "3" -> Zet alleen linker display aan -> Wacht kort.
2. Zet segmenten klaar voor getal "5" -> Zet alleen rechter display aan -> Wacht kort.
3. Herhaal dit oneindig.

Als dit sneller gebeurt dan 50 keer per seconde, zien onze ogen dit niet als knipperen, maar als twee constant brandende getallen.

De Technische Implementatie In seg7_display.vhd heb ik dit gerealiseerd in drie stappen:

1. De Refresh Counter:
   De FPGA klok (100 MHz) is veel te snel voor multiplexing. Als ik elke kloktik wissel, krijgen de LEDs niet genoeg energie om op te lichten (ghosting). Ik heb een teller (refresh_counter) gemaakt van 20 bits. Ik gebruik alleen de bovenste 2 bits hiervan om te bepalen welk display aan de beurt is. Dit vertraagt de wissel-snelheid naar ongeveer 1 kHz (optimaal voor het oog).

```
22      -- Vertraging voor het switchen van displays (multiplexing)
23      process(clk, reset)
24      begin
25          if reset = '1' then
26              refresh_counter <= (others => '0');
27          elsif rising_edge(clk) then
28              refresh_counter <= refresh_counter + 1;
29          end if;
30      end process;
```

2. De Anode Selector (AN):
   Met de bovenste 2 bits van de teller kies ik welk display actief is (Active Low logic: '0' is aan).
   - "00" -> Activeer display rechts (Score O)
   - "11" -> Activeer display links (Score X)
   - De middelste twee displays zet ik uit of gebruik ik voor een scheidingsteken.

3. De Cathode Decoder (SEG): Afhankelijk van welk display actief is, kies ik welke score getoond moet worden. Deze 4-bit score (0-9) wordt via een case-statement vertaald naar de juiste 7-bit code voor de segmenten a t/m g.

Bijv: 0000 (Cijfer 0) wordt 1000000 (alles aan behalve het middelste streepje g).

Hierdoor fungeert de seg7_display module als een volledig zelfstandige driver die continu op de achtergrond draait, zonder dat de hoofd-gamelogica zich druk hoeft te maken over timing of verversing.

DE VISUELE PRIORITEIT

Een cruciaal onderdeel van de weergave zit in de top.vhd. De FPGA krijgt tegelijkertijd kleur-informatie van het spelbord (raster/kruisjes) én van de tekst-module. Ik heb hier een prioriteits-multiplexer geschreven:

1. Heeft de tekst-module pixels (niet zwart)? -> Toon Tekst.
2. Is de Intro actief? -> Toon Zwart.
3. Anders -> Toon Spelbord. Dit zorgt ervoor dat de tekst ("WINNER" of "FOUT") altijd bovenop het speelveld wordt getekend en niet erdoorheen.

12. DEBUGGING EN TESTING

LED-TESTPROCES

Omdat ik niet altijd direct kon zien wat er misging op het VGA-scherm (soms bleef het zwart door timingfouten), heb ik de LEDs op het Basys 3 bordje gebruikt als debug-tools.

- Toepassing: Ik koppelde interne signalen aan de LEDs. LED(0) gaf de reset aan, LED(1) de beurtwissel.
- Resultaat: Hierdoor zag ik dat mijn turn signaal te snel wisselde door het denderen van de knop, wat leidde tot de implementatie van de State Machine.

SWITCH TEST

In het begin leek het alsof ik vakje 6 (rechtsboven) niet kon selecteren. Ik twijfelde aan mijn VHDL-code. Om dit uit te sluiten heb ik een simpele "hardware bypass" geschreven: led(6) <= sw(6). Toen ik de schakelaar omhoog deed, ging de LED niet aan. Hierdoor wist ik dat het probleem niet in mijn spellogica zat, maar in de fysieke mapping (constraints file). Het bleek een typfout in de .xdc file te zijn.

HET "DENDER" PROBLEEM BIJ RESETTEN

Tijdens het testen merkte ik een vreemd fenomeen. Wanneer ik op de bovenste knop (btnU) drukte om het spel te resetten, werkte dit soms wel, maar soms raakte het spel direct weer in de war. Soms leek het alsof de beurtwissel niet goed gereset werd.

Analyse:
Na onderzoek bleek dit te liggen aan het denderen (bouncing) van de mechanische knop. Wanneer je een knop indrukt, maakt het metaal binnenin even contact, laat los, en maakt weer contact. Dit gebeurt in milliseconden. Omdat de FPGA op 100 MHz werkt (100 miljoen keer per seconde), zag mijn logica de reset-knop niet als één druk, maar als een snelle reeks van aan-uit-aan signalen. Hierdoor werd het spel gereset, maar direct daarna in dezelfde fractie van een seconde opnieuw beïnvloed door zwevende signalen.

Oplossing:
Edge Detection Ik heb dit opgelost door flankdetectie (Edge Detection) toe te voegen aan de reset-knop, net zoals ik eerder bij de bevestigingsknop (btnC) deed. Ik kijk nu niet meer of de knop "hoog" is (if btnU = '1'), maar ik kijk naar het exacte moment dat hij van laag naar hoog gaat:

```
btnU_pressed <= '0';
if reset_soft = '1' and btnU_prev = '0' then
      btnU_pressed <= '1'; -- Puls van precies 1 kloktik lang
end if;
btnU_prev <= reset_soft;
```

SIMULATIE

Tijdens de synthese kreeg ik de foutmelding: [DRC INBB-3] Black Box Instances: Cell 'vga' has undefined contents.

- Oorzaak: Vivado wist dat ik een VGA-module wilde gebruiken, maar kon de inhoudelijke code niet vinden.
- Oplossing: Ik was vergeten het bestand vga_sync.vhd correct toe te voegen aan de project-hiërarchie. Na toevoeging werkte de synthese direct.

DE NOODZAAK VAN SIMULATIE TIJDENS HET ONTWIKKELPROCES

Hoewel de testbench in de volgende paragraaf dient als verificatie van het eindproduct, wil ik benadrukken dat simulatie voor mij geen eenmalige eindcontrole was, maar een continue en essentiële tool gedurende het hele project.

Tijdens de eerdere tussenstappen (zoals het opzetten van de VGA-timing en de eerste state machines) liep ik vaak tegen onverwacht gedrag aan. Op de fysieke FPGA resulteerde dit simpelweg in een zwart scherm, waardoor debuggen onmogelijk was. Door op die momenten terug te grijpen naar de simulatie in Vivado, kon ik signaalconflicten op nanoseconde-niveau opsporen die met een multimeter of blote oog onzichtbaar zijn. Hoewel ik deze iteratieve cyclus van foutopsporing veel keren heb doorlopen, zou het ons te ver brengen om elke afzonderlijke casus hier te bespreken. Deze processen hebben mij doen inzien dat simulatie bij hardware-ontwerp geen optie is, maar een noodzaak; zonder deze tool was het oplossen van timing-problemen gokwerk geweest.

SIMULATIE LOGICA (TESTBENCH)

Om de correcte werking van de spellogica en de win-detectie te verifiëren zonder afhankelijk te zijn van externe hardwarefactoren (zoals denderende knoppen of VGA-timing), heb ik een behavioral simulation uitgevoerd.

Ik heb hiervoor een testbench geschreven (tb_game_logic.vhd) waarin de module game_logic als *Unit Under Test* (UUT) wordt geplaatst. In deze simulatie wordt een volledig spelverloop nagebootst via code.

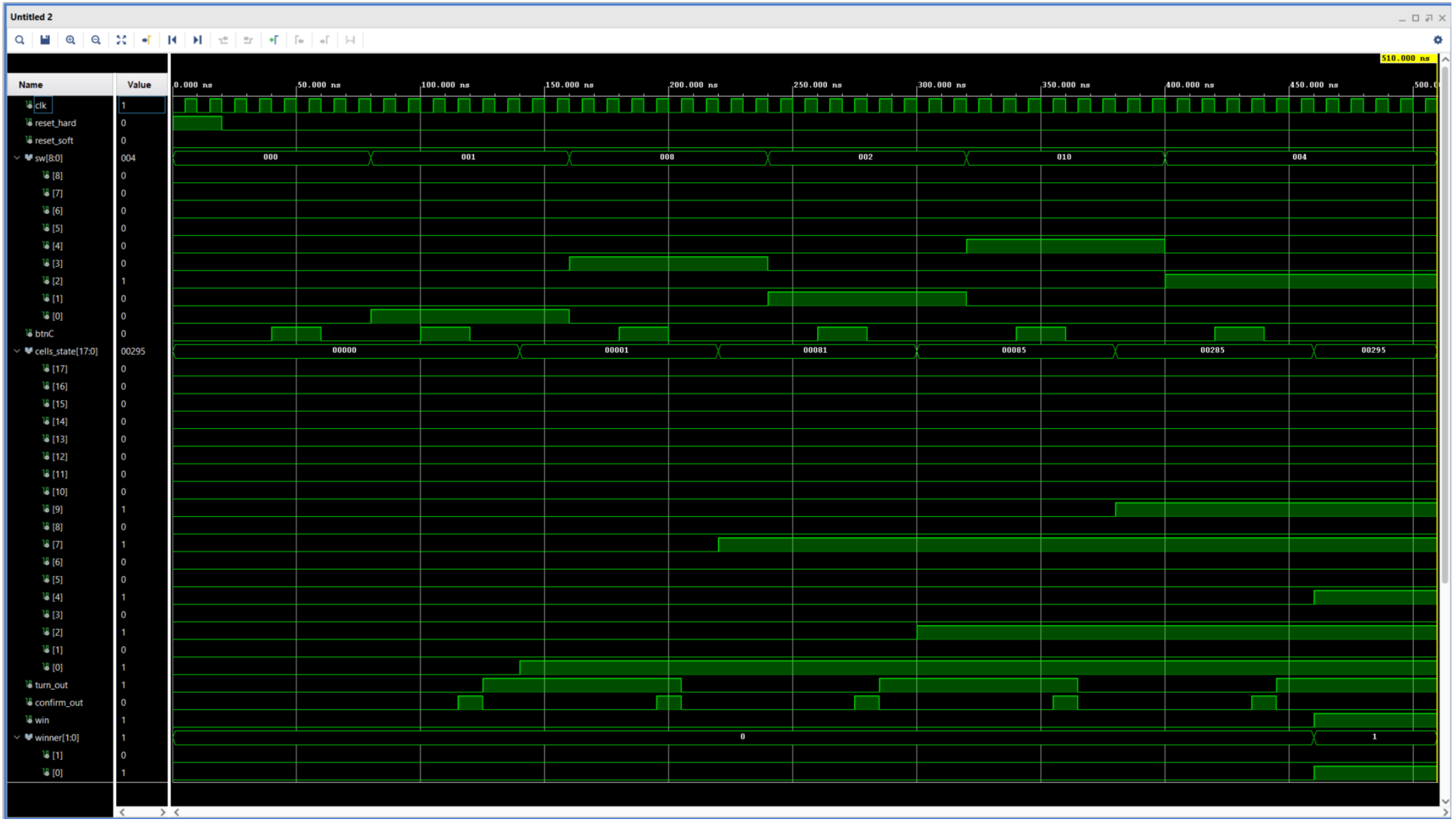
Het Test Scenario: De testbench voert de volgende stappen automatisch uit:

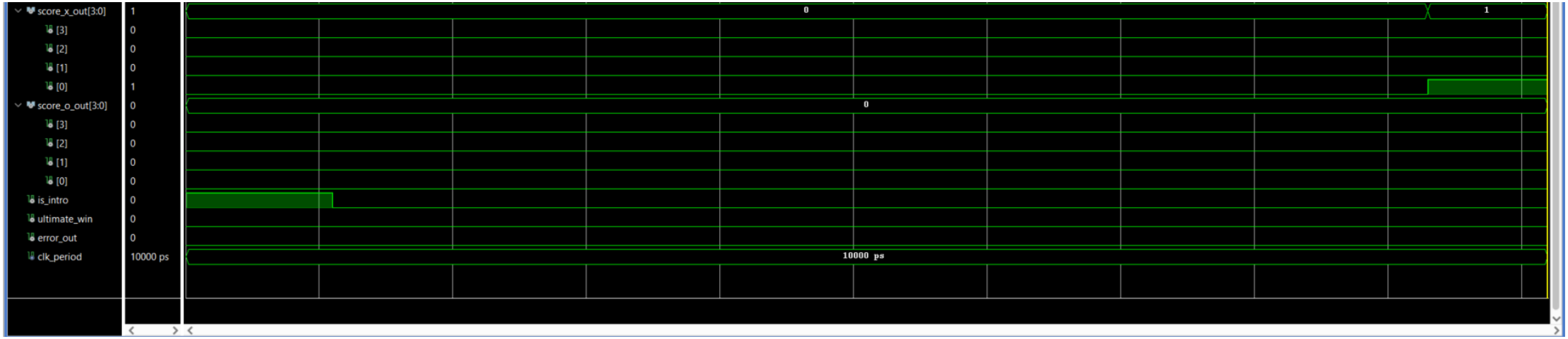
1. Initialisatie: Het systeem krijgt een harde reset en het intro-scherm wordt weggeklikt (startknop simulatie).
2. Spelverloop: De testbench simuleert inputs van de switches en de center-knop om beurtelings zetten te doen voor Speler X en Speler O.
3. Feedback Simulatie: Omdat de grafische board-module in deze test ontbreekt, manipuleert de testbench na elke zet handmatig het signaal cells_state. Hiermee wordt gesimuleerd dat het bord succesvol is bijgewerkt.

Resultaat: In het gesimuleerde scenario vult Speler X de bovenste rij (vakjes 0, 1 en 2). Zoals te zien is in de onderstaande waveform, reageert de logica correct na de beslissende zet:

- Het signaal win springt naar hoog ('1').
- Het signaal winner geeft de waarde "01" aan (wat correspondeert met Speler X).

Dit bevestigt dat de State Machine, de beurtwisseling en het parallelle winst-algoritme foutloos functioneren.


Untitled 2
Name
Value
clk
reset_hard
reset_soft
sw[8:0]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
btnC
cells_state[17:0]
[17]
[16]
[15]
[14]
[13]
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
turn_out
confirm_out
win
winner[1:0]
[1]
[0]
0.000 ns
50.000 ns
100.000 ns
150.000 ns
200.000 ns
250.000 ns
300.000 ns
350.000 ns
400.000 ns
450.000 ns
510.000 ns
000
001
008
002
010
004
00000
00001
00081
00085
00285
00295
0
1


score_x_out[3:0]
[3]
[2]
[1]
[0]
score_o_out[3:0]
[3]
[2]
[1]
[0]
is_intro
ultimate_win
error_out
clk_period
1
0
0
0
1
0
0
0
0
0
0
0
0
10000 ps
0
1
0
10000 ps

13. RESULTAAT

Na een semester van ontwerpen, implementeren en iteratief testen, is het eindresultaat een volledig zelfstandig functionerend digitaal systeem op de Basys 3 FPGA. Het project voldoet aan alle gestelde eisen en bevat extra functionaliteiten zoals tekstweergave en scorebijhouding.

STABIELE VISUELE WEERGAVE (VGA)

Het systeem genereert een trillingsvrij 640x480 beeld op 60Hz. De timing voldoet aan de industriestandaarden. Het speelveld is nauwkeurig gecentreerd (440x440 px) en opgebouwd uit een esthetische combinatie van een grijze border, witte rasterlijnen en een diepzwarte achtergrond.

ROBUUSTE SPELBESTURING

Het spel is volledig speelbaar als "Tic-Tac-Toe". De besturing via de 9 switches en bevestigingsknop voelt direct en betrouwbaar aan. Dankzij de geïmplementeerde **Smart Switch validatie** en **Edge Detection** is het systeem "fool-proof": valsspelen of het spel laten vastlopen door willekeurige input is onmogelijk gemaakt.

COMPLETE SPELERVARING & FEEDBACK

De FPGA detecteert parallel alle 8 mogelijke win-condities binnen één klokcyclus. Bij winst schakelt het systeem direct over naar een feedback-modus:

- Visueel: De winnaar wordt op het scherm getoond met custom bitmapped tekst ("X WINT" in cyaan of "VIERKANT WINT" in rood).
- Score: De stand wordt real-time bijgehouden en weergegeven op het 7-segment display via Time Multiplexing.

TRACEERBAARHEID VIA GITHUB

Het volledige project en de broncode zijn gepubliceerd op GitHub. De opbouw van deze repository is zo ingericht dat deze zo goed als een één-op-één overeenkomt met de titels en hoofdstukken van dit document. Hierdoor is de code niet alleen als eindproduct beschikbaar, maar is de volledige ontwikkelingsgeschiedenis (van eerste VGA-pixel tot eindspel) stap voor stap terug te vinden aan de hand van deze documentatie.

Repository: https://github.com/Daanvdw2005/FPGA_Project_2025

DEMONSTRATIE PANOPTO

In volgende video geef ik een uitleg (geluid) en een demonstratie van mijn finale versie:

https://ap.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=823933cd-92d1-4ab3-8138-b3c3015037ae

14. REFLECTIE

De onderschatting van de complexiteit

Eerlijk gezegd heb ik de omvang van dit project zwaar onderschat. Aanvankelijk dacht ik: *"Het is maar 3-op-een-rij, dat wordt een makkie"*. Niets was minder waar. Ik zag dat veel medestudenten kozen voor een project rondom één bewegend object of attribuut (bijvoorbeeld een blokje dat over het scherm beweegt en verdwijnt als het iets raakt). Dat is vooral een kwestie van coördinaten updaten. Bij mijn project kwam er echter veel meer fundamenteel denkwerk en logica kijken. Het is geen kwestie van één object dat beweegt, maar een systeem dat negen verschillende statussen moet onthouden, beurten moet wisselen, valsspelen moet voorkomen en continu acht verschillende win-lijnen moet controleren. De logische architectuur achter dit statische bord bleek vele malen complexer dan ik vooraf had kunnen voorspellen.

Wat heb ik geleerd?

Het belangrijkste leerpunt van dit project was het fundamentele verschil tussen software en hardware (VHDL). In software gebeuren dingen sequentieel (na elkaar), maar in een FPGA gebeurt alles parallel (tegelijkertijd). Dit dwong me om heel goed na te denken over toestanden (States) en kloksignalen. Een simpele "if"-lus werkt anders in hardware dan in klassiek programmeren. Daarnaast heb ik de kracht van modulair denken ontdekt. Door de cell, board en game_logic strikt te scheiden, bleef mijn project overzichtelijk.

Wat zou ik anders aanpakken?

Als ik meer tijd had gehad, had ik de besturing willen aanpassen. Nu gebruik ik switches, wat technisch goed werkt, maar voor een gebruiker wat traag aanvoelt. Een systeem waarbij je met de pijltjestoetsen een cursor over het scherm beweegt, zou de "game feel" verbeteren, maar vereist complexere logica (geheugen van cursor coördinaten).

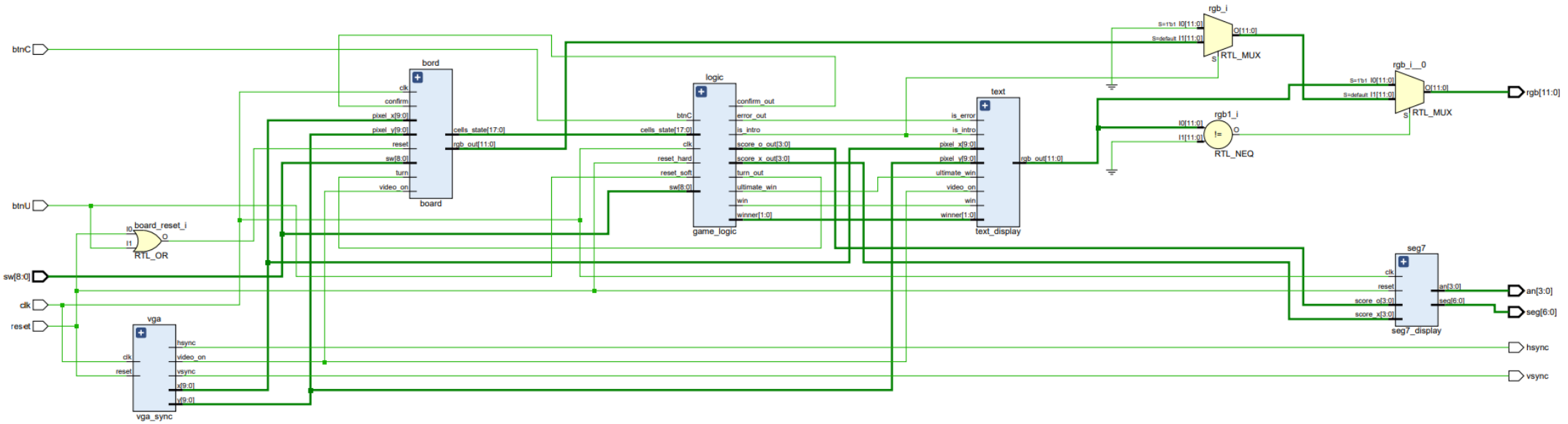
Wat vond ik het moeilijkst / interessantst?

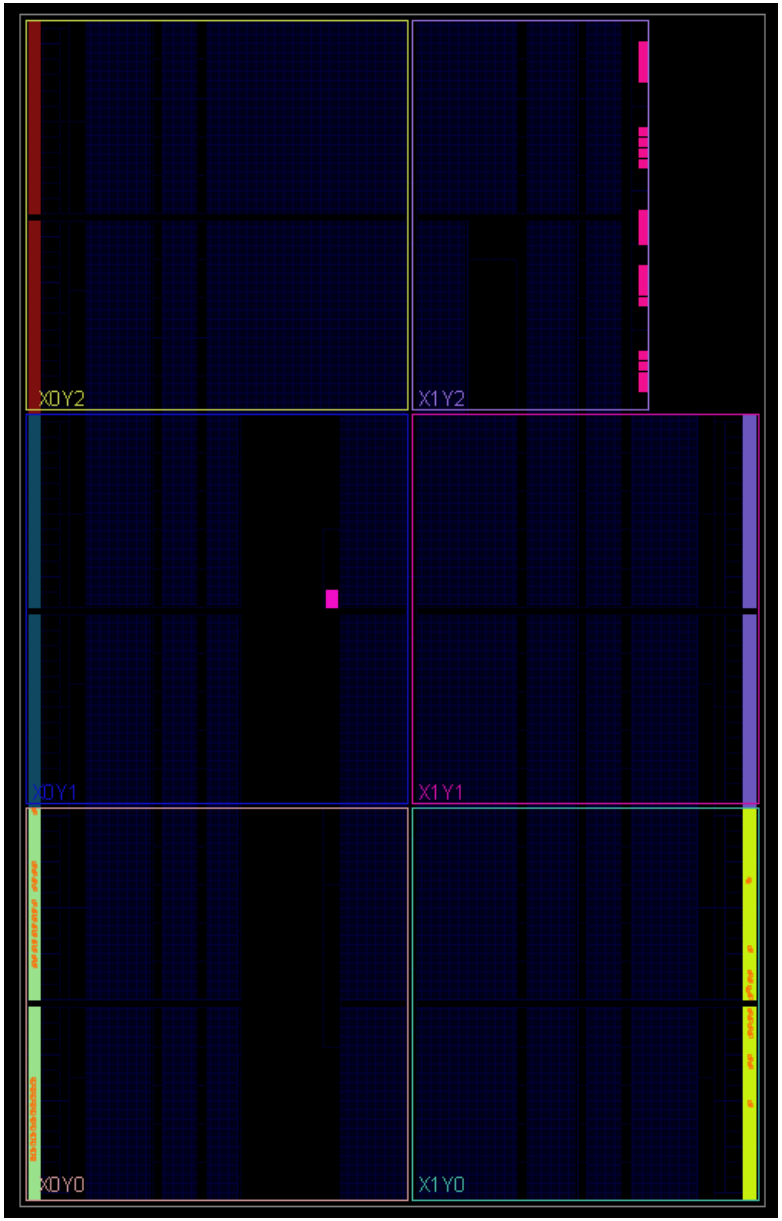
Het moeilijkste was het oplossen van de timing-problemen (de race conditions waarbij X in O veranderde). Het gaf echter wel de meeste voldoening toen ik via de State Machine de oplossing vond en het spel plotseling perfect stabiel draaide. Het feit dat ik nu een eigen chip heb "geprogrammeerd" die zelfstandig een VGA-signaal genereert, vind ik het meest fascinerende aspect van dit vak.

15. BIJLAGEN

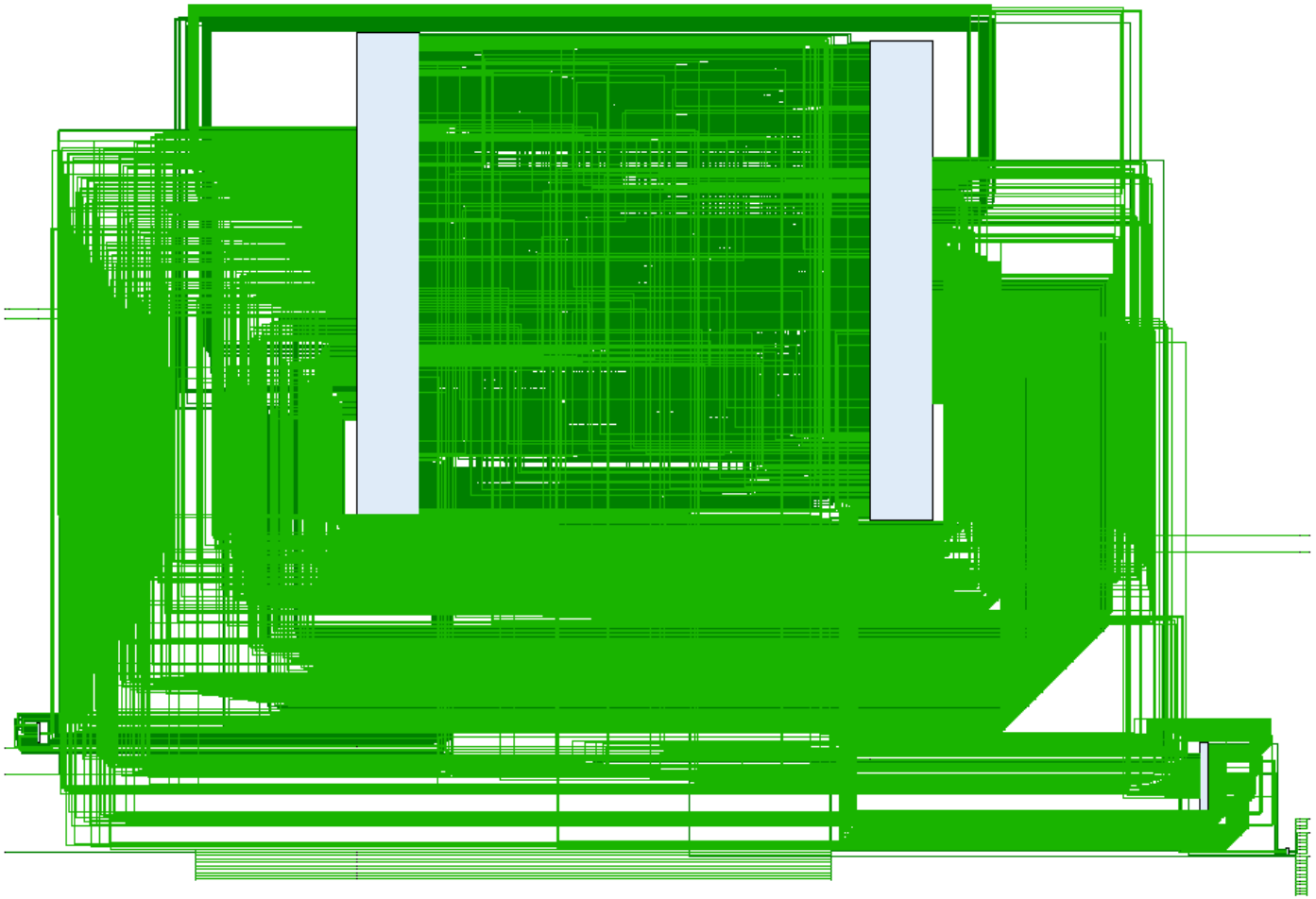
VIVADO (RTL, IMPLEMENTATIE, PIN MAPPING)

RTL ANALYSE FINAAL

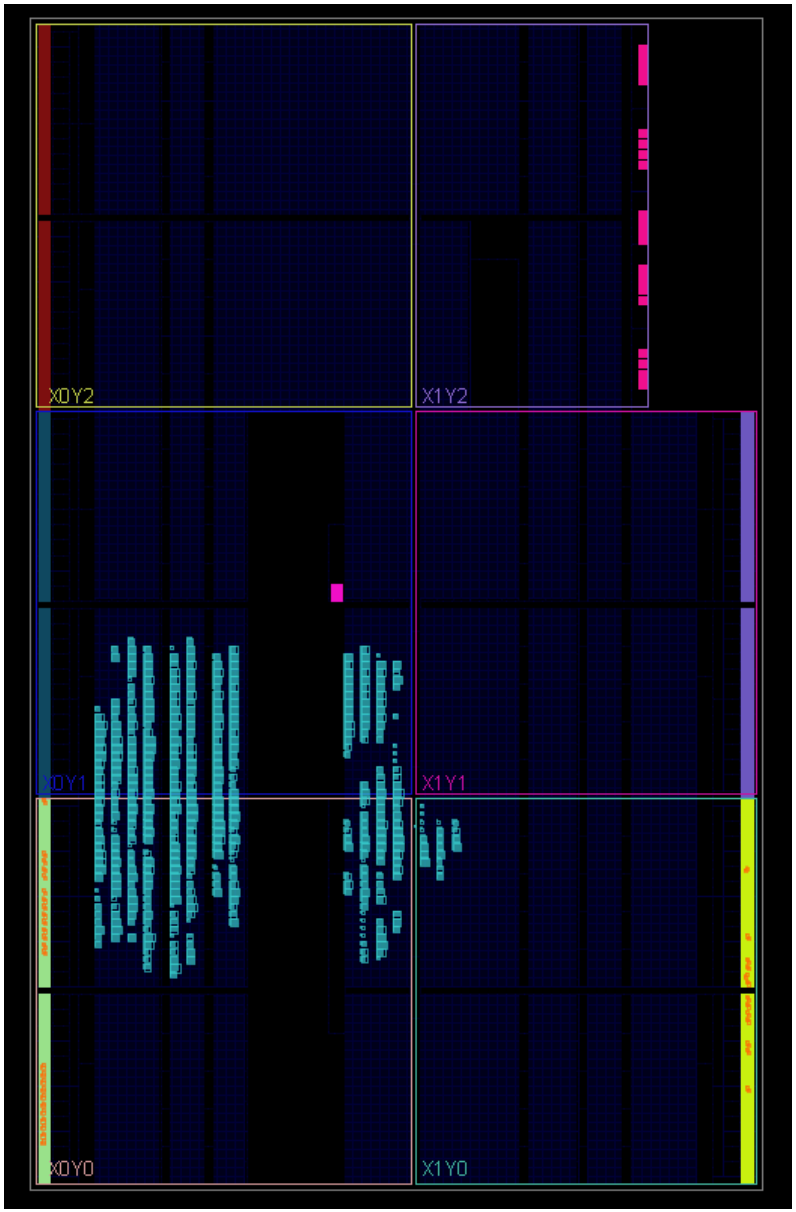



X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0

SYNTHESIS SCHEMATIC

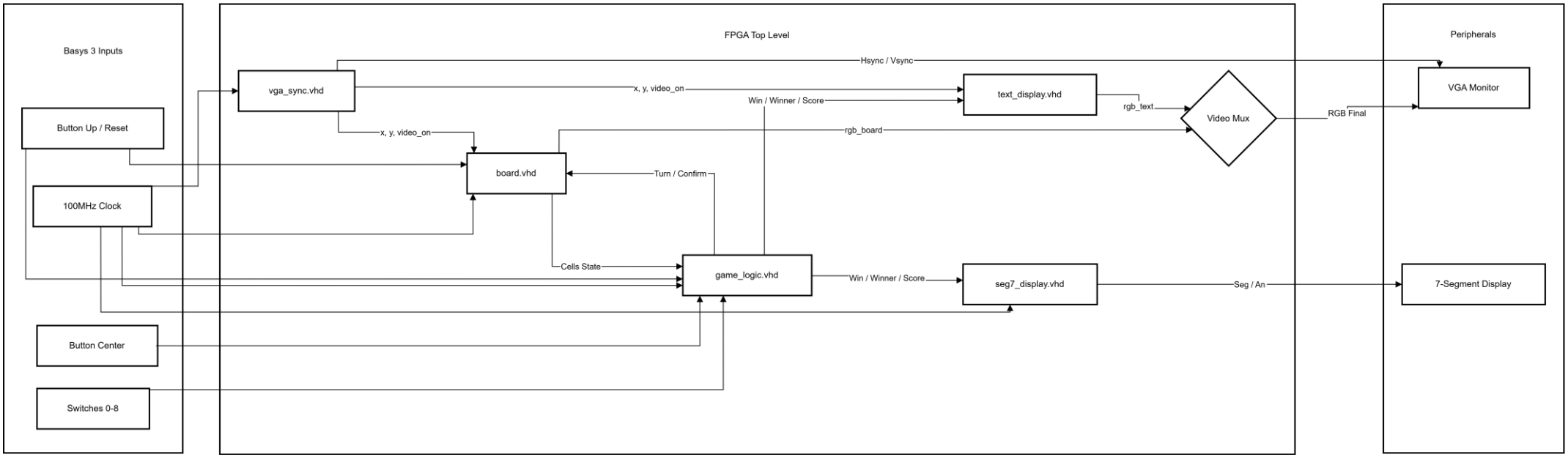


IMPLEMENTED DESIGN

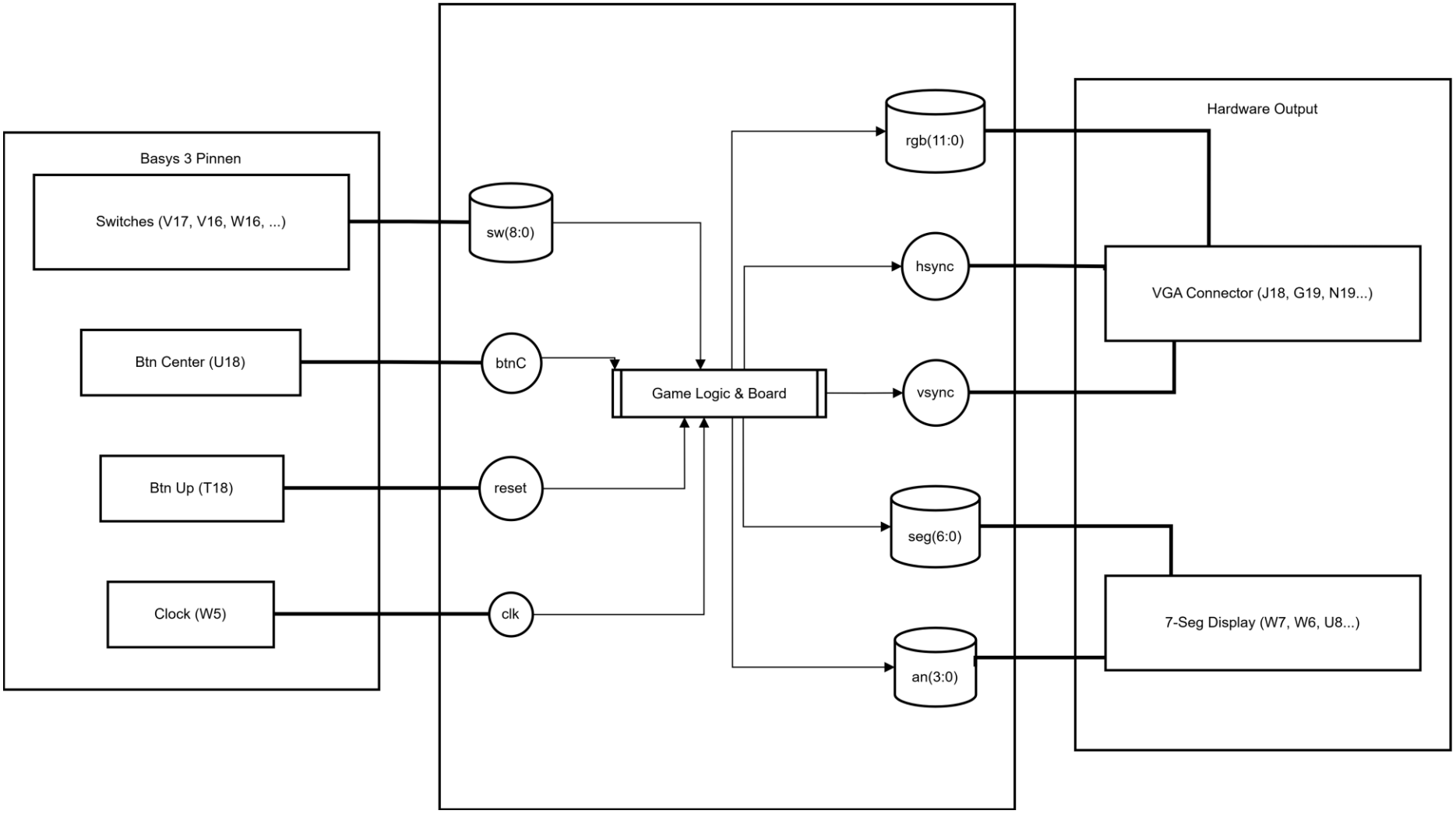


SCHEMA’S

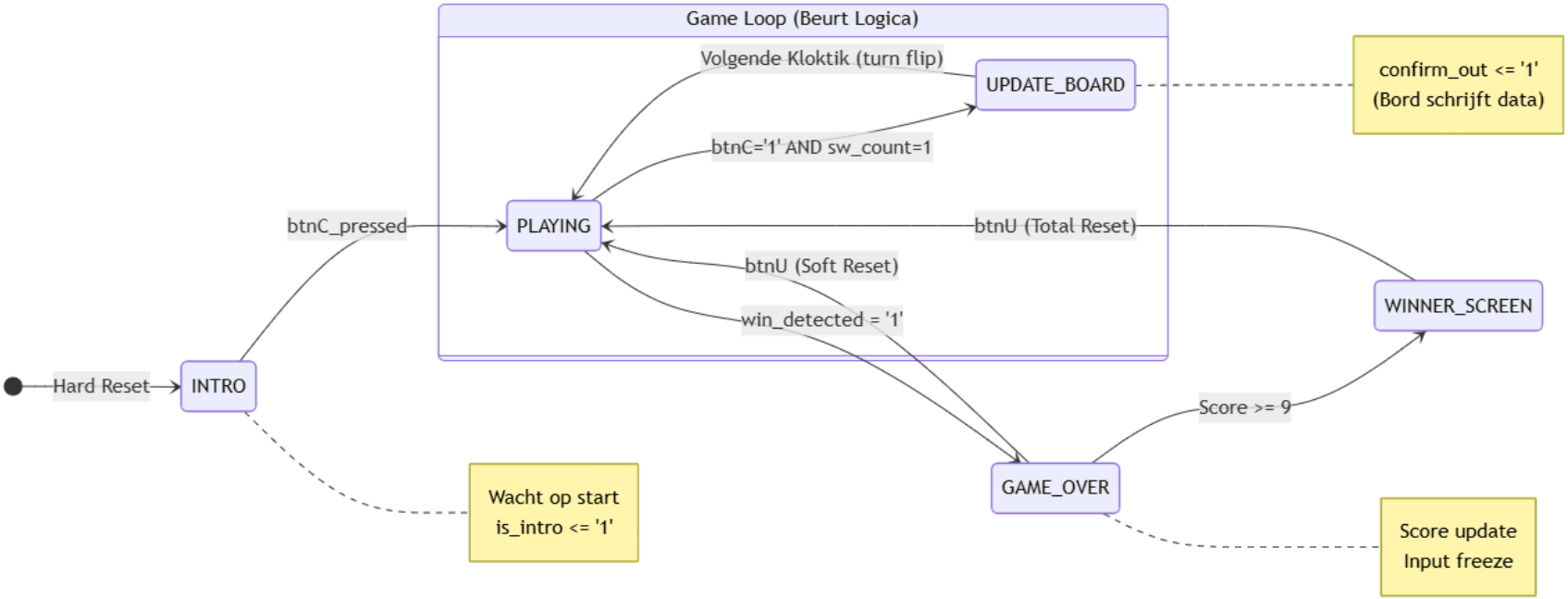
TOP LEVEL LOGIC



PIN MAPPING


Basys 3 Pinnen
Switches (V17, V16, W16, ...)
Btn Center (U18)
Btn Up (T18)
Clock (W5)
sw(8:0)
btnC
reset
clk
Game Logic & Board
rgb(11:0)
hsync
vsync
seg(6:0)
an(3:0)
Hardware Output
VGA Connector (J18, G19, N19...)
7-Seg Display (W7, W6, U8...)

FINITE STATE MACHINE



16. BRONNENLIJST

AMD Xilinx. (2023). *Vivado Design Suite User Guide: Implementation* (UG904, v2023.2). https://docs.xilinx.com/r/en-US/ug904-vivado-implementation

Digilent. (2014, 12 april). *Basys 3 FPGA Board Reference Manual*. Geraadpleegd op 17 september 2025, van https://digilent.com/reference/programmable-logic/basys-3/reference-manual

Pedroni, V. A. (2020). *Circuit Design with VHDL* (3e editie). MIT Press.

TinyVGA. (z.d.). *VGA Signal 640 x 480 @ 60 Hz Timing*. Geraadpleegd op 24 september 2025, van http://www.tinyvga.com/vga-timing/640x480@60Hz

D. Van Merode. (2025). *Cursusmateriaal Digital Systems Development: Introductie VHDL en VGA-aansturing* [PowerPoint-slides]. Departement Elektronica-ICT, AP-Hogeschool.

Basys 3 Artix-7 FPGA Trainer Board: Recommended for introductory users : Amazon.nl: Elektronica. (n.d.). https://www.amazon.nl/-/en/Basys-Artix-7-FPGA-Trainer-Board/dp/B00NUE1WOG

***Schematic Diagram Electronic Device This Electrical**: stockillustratie 2291272367 | Shutterstock*. (n.d.). Shutterstock. https://www.shutterstock.com/nl/image-illustration/schematic-diagram-electronic-device-this-electrical-2291272367