|  |
| --- |
|  |
| HW/SW design rapport |
|  |
|  |
| **Aleksandar Djuric** |
| **2023-11-28** |

**Sammanfattning**: Rapporten fokuserar på VGA\_IP, en VHDL-baserad IP-kärna som effektivt hanterar grafiska data i FPGA-system. Kärnan fungerar som en brygga mellan VGA-skärm och internminne och möjliggör visualisering med RGB-stöd. Dess kompatibilitet med Avalon CPU-arkitekturen gör det möjligt att integrera grafiska visualiseringar med processorberäkningar, vilket ger en bred användning inom FPGA-baserade system. Rapporten betonar kärnans enkla integration och möjligheten till anpassning för specifika projektbehov.

INNEHÅLLSFÖRTECKNING

[1. KRAVSPECIFIKATION 3](#_Toc152081821)

[2. Konstruktionsbeskrivning 5](#_Toc152081822)

[2.1 VGA\_IP 5](#_Toc152081823)

[2.2 VGA\_SYNC 6](#_Toc152081824)

[2.2.1 In/utgångar 6](#_Toc152081825)

[2.3 Delsystem 7](#_Toc152081826)

[2.3.1 Clock divider process 7](#_Toc152081827)

[2.3.2 Metastability process & Switch process 7](#_Toc152081828)

[2.3.3 Counter process 9](#_Toc152081829)

[2.3.4 RGB validering 10](#_Toc152081830)

[2.3.5 Adressberäkning för VGA sync 12](#_Toc152081831)

[3. Double-Port Ram 14](#_Toc152081832)

[3.1 VHDL kod 16](#_Toc152081833)

[4. Platform designer 17](#_Toc152081834)

[5. Testprotokoll 18](#_Toc152081835)

[6. Projekt kostnad 18](#_Toc152081836)

[7. Bilaga 19](#_Toc152081837)

# KRAVSPECIFIKATION

Tabell 1 Kravspecifikation

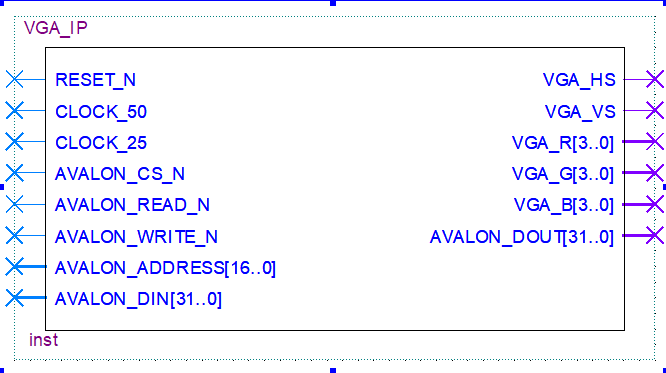
|  |  |  |
| --- | --- | --- |
| **Krav** | **Beskrivning** | **Utfört** |
|  | | ***Ja/nej*** |
| 001 | Kundprojektets arbetsnamn är VGA\_IP | **JA** |
| 002 | Skapa en IP komponent med VGA-komponenten och återanvänd tidigare VHDL- och C-projekt. Alltså VHDL koden från VHDL kursen och C koden från C kursen (VGA uppgiften). | **JA** |
| 003 | Krav på konstruktionskoden:  1)”filhuvud” i C-filen.  2) Namn på variabler med mera ska vara tydliga.  3) Rikligt med kommentarer i koden.  4) *Frivilligt:* Kontroll av gränsvärden ska göras, till exempel att värdet inte ligger utanför skärmen. Om detta inträffar ska ett felmeddelande skrivas ut till systemterminalen (Nios II ”console”). | **JA** |
| ***Funktionskrav*** | |  |
| 004 | **Följande drivrutiner** sparas under HAL och ska vara uppdaterade i VGA\_IP\_SW\_sw.tcl filen). Drivrutinerna kan kopieras från tidigare utfört projekt i C-kursen eller från ett äldre projekt. Frivilligt att lägga på flera drivrutiner.  **Funktion:** print\_pix(unsigned int x,unsigned int y,unsigned int rgb); **Funktionsbeskrivning:** Skriver en pixel med färgen *rgb* på koordinaten (x, y). Alternativ lösning för print\_pix; om funktionen behöver snabbas upp, kan den istället göras om till ett macro.  **Funktion:** print\_hline(unsigned int x\_start, unsigned int y\_start, unsigned int len, unsigned int RGB); **Funktionsbeskrivning:** Skriver en horisontell linje med färgen *rgb* och med längden *len* vilken startar på koordinaten (x\_start, y\_start).  **Funktion:** print\_vline(unsigned int x\_start, unsigned int y\_start, unsigned int len, unsigned int RGB); **Funktionsbeskrivning:** Skriver en vertikal linje med färgen *rgb* och med längden *len* vilken startar på koordinaten (x\_start, y\_start).  **Funktion:** print\_char(unsigned int x,unsigned int y,unsigned int rgb,unsigned int BG\_RGB,char Character); **Funktionsbeskrivning:** Skriver tecknet “Character” med färgen *rgb* och med bakgrundsfärgen ”BG\_RGB” på koordinaten (x, y). Lägg till de ASCI som behövs för att skriva det som krävs. Det går att bestämma hur tecknen ska visas på skärmen, se bilaga, äldre projekt och på webben.  **Funktion:** read\_pixel\_ram\_int(unsigned int x\_start, unsigned int y\_start); return: unsigned pixel\_data (obs! enbart de sista tre bitarna är intressanta). **Funktionsbeskrivning:** Läser pixel\_data från pixel RAM (3 bitar, RGB) från adress x och y (beräknat från x och y).  **Funktion**: clear\_screen (int rgb);  **Funktionsbeskrivning**: Denna funktion rensar skärmen genom att RGB-värdet ”000”(svart) skrivs till alla pixlar på skärmen.  **Funktion:** print\_circle(unsigned int radie, unsigned int x\_centrum, unsigned int y\_centrum, unsigned int rgb); **Funktionsbeskrivning:** Skriver en cirkel med radien “radie” och färgen” rgb” på mittkoordinaten (x, y). Cirkeln ska fyllas med samma färg. | **JA** |
| **Valideringskrav** | |  |
| 005 | Testprotokollet kan vara ofullständigt, men ska innehålla minst tre testfall (HW/SW design report). | **JA** |
| 006 | Skriv ditt namn på skärmen, fotografera skärmen och lägg in fotot som figur i rapporten (HW/SW design report). | **JA** |
| 007 | Rapporten ska också innehålla kostnaden för konsultuppdraget. Företaget konstukören är anställd på fakturerar 600 SEK/timme för projektet. Totalt antal timmar \* 600 SEK utgör den totala kostnaden. | **JA** |
| ***Struktur på leveransen och dokumentationskrav*** | | |
| 008 | Leverera följande struktur.   * VGA\_IP (top library and the sub library)   + VGA\_IP\_HW\_hw.tcl   + VGA\_IP\_SW\_sw.tcl   + hdl (HW),     - ….vhd (VHDL code)   + HAL (SW),     - Inc (library for h files)     - src (library for c files)   + doc (documentations),     - VGA\_datasheet     - Software manual for the programmers     - SW/HW Design Report (standard report)   + Examples     - Ett exempelsystem som skriver ut ditt namn på skärmen     - SOF och sopcinfo filen     - Arkiverade projektet     - C filen (main)   + Projects (HW)     - VGA\_IP (Quartus) |  |
| ***Leveranskrav*** | |  |
| 009 | Leveransen ska ske till plattformen Itslearning. Leveransen ska vara en IP-komponent enligt strukturen från krav 004. Namnet på filen ska vara ”fornamn\_efternamn\_HWSW\_task\_3.zip”. Sista leveransdag se Itslearning (för VG). |  |

# Konstruktionsbeskrivning

## VGA\_IP

VGA\_IP är en VHDL-baserad IP-kärna utformad för att effektivt hantera grafiska data. Den agerar som en brygga mellan en VGA-skärm och det interna minnet i FPGA:n genom användning av en Double-Port RAM (DP\_RAM). Genom att integrera VGA\_IP i FPGA-designer möjliggörs visualisering av grafisk information på en VGA-skärm med stöd för färgerna röd, grön och blå.

Denna IP-kärna har ytterligare funktionalitet som gör den användbar i samband med Avalon CPU-arkitekturen. Den erbjuder en gränssnittsbro för att enkelt ansluta och utbyta data med en Avalon-kompatibel processorenhet. Genom denna förmåga fungerar VGA\_IP som en integrerad länk mellan grafiska visualiseringar och processorbaserade beräkningar, vilket möjliggör en bredare och mer mångsidig användning inom FPGA-baserade system. VGA\_IP är designad för enkel integration och erbjuder möjligheter till vidareutveckling för att skräddarsy grafikfunktionerna efter specifika behov inom projekt och tillämpningar.

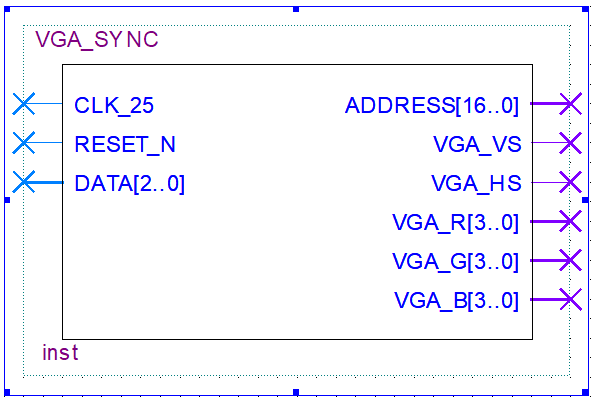


Figur 1: En överblick av VGA\_IP arkitektur

## VGA\_SYNC

RGB-signalerna lagras i en data-array, vilken läses från DP\_RAM och visar sedan resultatet på en skärm med upplösningen 620x480 och stöd för färgerna röd, blå och grön. Detta representerar en grundläggande VGA\_sync, som kan vidareutvecklas efter behov.

Nedan följer en figur och en tabell som visar ingångar och utgångar för konstruktionen.



Figur 2: Överblick av VGA-sync arkitektur

### In/utgångar

Följande tabell visar In och utgångar för VGA\_sync.

Tabell 2: Visar In-och utgångar för VGA\_sync

|  |  |  |
| --- | --- | --- |
| **IN** | **Beskrivning** |  |
| CLOCK\_50 | Systemklocka | 50Mhz |
| reset\_n | Reset |  |
| SW[2..0] | User input | Switchar |
| **UT** |  |  |
| VGA\_VS | Vertical sync |  |
| VGA\_HS | Horizontal sync |  |
| VGA\_R[3..0] | Röd färg |  |
| VGA\_G[3..0] | Grön färg |  |
| VGA\_B[3..0] | Blå färg |  |

## Delsystem

Delsystem innefattar processer och signaler som samverkar för att uppnå systemets mål. Nedan beskrivs varje delsystem och dess funktionella roll i systemet.

### Clock divider process

Clock divider process är en kritisk komponent i systemet som ansvarar för att generera en klocksignal med en lägre frekvens än den ingående klockan CLOCK\_50. Detta görs genom att dela ingångsklockan med två. Processen fungerar enligt följande:

1. Vid en låg (0) nivå på nollställningssignalen reset\_n, sätts clk\_25 till låg (0) för att klart indikera att systemet är nollställt.
2. Vid stigande flank av ingångsklockan CLOCK\_50, inverteras clk\_25 om nollställningssignalen är hög (1).

Resultatet av denna process är en 25 MHz klocka (clk\_25) som används för att tidsreglera andra delar av systemet.

**VHDL kod**

    ------------------ Clock divider process ------------------

    clock\_divider : PROCESS (CLOCK\_50, reset\_n)

    BEGIN

        IF reset\_n = '0' THEN

            clk\_25 <= '0';

        ELSIF rising\_edge(CLOCK\_50) THEN

            -- Divide the clock by two

            clk\_25 <= NOT clk\_25;

        END IF;

    END PROCESS;

    ------------------ Clock divider process ------------------

### Metastability process & Switch process

Metastability Process är en del av systemet som hanterar problemet med metastabilitet som kan uppstå när asynkrona signaler används. Det är viktigt att förhindra att metastabilitet sprids i systemet. Processen fungerar enligt följande:

1. Vid låg (0) nivå på nollställningssignalen reset\_n sätts både reset\_t1 och reset\_t2 till låg (0).
2. Vid stigande flank av ingångsklockan CLOCK\_50, sätts först reset\_t1 till hög (1) och sedan reset\_t2 sätts till samma värde som reset\_t1.

Detta säkerställer att metastabilitet hanteras ordentligt genom att använda två registersteg och att metastabiliteten inte sprider sig till andra delar av systemet.

Switch Process övervakar och lagrar värdena från brytarsignalerna i SW. Det är viktigt för systemet att ha tillgång till de aktuella switchvärdena för att fatta beslut om att generera VGA-utgångar. Processen fungerar enligt följande:

1. Vid låg (0) nivå på reset\_t2 sätts både sw\_t1 och sw\_t2 till en 3-bitars vektor med nollvärden.
2. Vid stigande flank av ingångsklockan CLOCK\_50, lagras de aktuella värdena från SW i sw\_t1, och sedan kopieras sw\_t1 till sw\_t2.

Denna process garanterar att switchvärdena är tillgängliga och stabila för resten av systemet att använda.

**VHDL kod**

    ------------------ Metastability process ------------------

    RESET\_PROC : PROCESS (CLOCK\_50, reset\_n)

    BEGIN

        IF reset\_n = '0' THEN

            reset\_t1 <= '0';

            reset\_t2 <= '0';

        ELSIF rising\_edge(CLOCK\_50) THEN

            reset\_t1 <= '1';

            reset\_t2 <= reset\_t1;

        END IF;

    END PROCESS;

    SW\_PROC : PROCESS (CLOCK\_50, reset\_t2)

    BEGIN

        IF reset\_t2 = '0' THEN

            sw\_t1 <= "000";

            sw\_t2 <= "000";

        ELSIF rising\_edge(CLOCK\_50) THEN

            sw\_t1 <= sw;

            sw\_t2 <= sw\_t1;

        END IF;

    END PROCESS;

    ------------------ Metastability process ------------------

### Counter process

Counters Process är ansvarig för att generera räknare som används för att bestämma positionen på bildskärmen. Det finns två räknare: x\_counter och y\_counter, som representerar den horisontella och vertikala positionen på VGA-bildskärmen. Processen fungerar enligt följande:

1. Vid låg (0) nivå på nollställningssignalen reset\_n, rensas båda räknarna till noll.
2. Vid stigande flank av clk\_25, inkrementeras x\_counter med ett varv när det når värdet 799, och y\_counter inkrementeras när x\_counter når värdet 706 och y\_counter inte har nått sitt maximala värde på 524.

Dessa räknare används sedan för att generera horisontella och vertikala synkroniseringssignaler för VGA samt för att kontrollera vilka RGB-utgångar som ska vara aktiva baserat på switchvärden.

Följande VHDL kod visar counter processen.

    ------------------ Counters process ------------------

    PROCESS (clk\_25, reset\_t2)

    BEGIN

        IF reset\_n = '0' THEN

            -- clear counter signals

            x\_counter <= (OTHERS => '0');

            y\_counter <= (OTHERS => '0');

        ELSIF rising\_edge(clk\_25) THEN

            -- counters

            -- increment x\_counter (counter.H) every clock pulse

            ------------ x\_counter ------------

            IF x\_counter >= 799 THEN

                x\_counter <= (OTHERS => '0');

            ELSE

                x\_counter <= x\_counter + 1; -- increment x\_counter

            END IF;

            ------------ x\_counter ------------

            -- increment y\_counter (counter.V) when x\_counter is 706

            ------------ y\_counter ------------

            IF x\_counter = 706 THEN

                IF y\_counter = 524 THEN

                    y\_counter <= (OTHERS => '0');

                ELSE

                    y\_counter <= y\_counter + 1; -- increment y\_counter

                END IF;

            END IF;

            ------------ y\_counter ------------

        END IF;

    END PROCESS;

    ------------------ Counters process ------------------

### RGB validering

Denna del av VHDL-koden innehåller konkurrenta asynkrona uttryck som är avgörande för att generera synkroniseringssignaler för en VGA-skärm och för att styra de röda, gröna och blå (RGB) komponenterna av utgångssignalen beroende på systemets interna status och switchinställningar. Nedan följer delar av koden, i slutet av denna text finner du denna del av kod komplett.

Horisontell och Vertikal Synkroniseringssignaler (VGA\_HS och VGA\_VS)

  VGA\_HS <= '0' when x\_counter >= 660 and x\_counter <= 755 else

            '1';

  VGA\_VS <= '0' when y\_counter = 494 else

            '1';

1. VGA\_HS: Denna signal representerar den horisontella synkroniseringen för VGA. Den sätts låg (0) när x\_counter är inom intervallet 660 till 755, vilket indikerar början av en ny horisontell bildskärmsrads. Annars sätts den hög (1).
2. VGA\_VS: Denna signal representerar den vertikala synkroniseringen för VGA. Den sätts låg (0) när y\_counter är exakt 494, vilket markerar början av en ny vertikal bildskärmsram. Annars sätts den hög (1).

Dessa synkroniseringssignaler är avgörande för att koordinera och synkronisera bildskärmsåtergivningen och säkerställa att den visas korrekt.

Följande kod styr RGB-färgsignalerna som används för att producera färgen på VGA-skärmen baserat på x\_counter, y\_counter.

  VGA\_R <= (OTHERS => '1') when DATA(2) = '1' and

                                (x\_counter >= 0 and x\_counter <= 639) and

                                (y\_counter >= 0 and y\_counter <= 479) else

           (OTHERS => '0');

  VGA\_G <= (OTHERS => '1') when DATA(1) = '1' and

                                (x\_counter >= 0 and x\_counter <= 639) and

                                (y\_counter >= 0 and y\_counter <= 479) else

           (OTHERS => '0');

  VGA\_B <= (OTHERS => '1') when DATA(0) = '1' and

                                (x\_counter >= 0 and x\_counter <= 639) and

                                (y\_counter >= 0 and y\_counter <= 479) else

           (OTHERS => '0');

**VGA\_R**: Denna signal kontrollerar den röda komponenten av VGA-utgången. Om x\_counter och y\_counter är inom specifika områden och när data är två (2) aktivera färg (röd).

**VGA\_G**: Denna signal styr den gröna komponenten av VGA-utgången. Den fungerar på ett liknande sätt som VGA\_R, när data är ett (1) aktivera färg (grön).

**VGA\_B**: Denna signal styr den blå komponenten av VGA-utgången. Den fungerar på ett liknande sätt som VGA\_R, när data är ett (0) aktivera färg (blå).

Dessa RGB-signaler är avgörande för att generera olika färger på VGA-skärmen och kan anpassas beroende på switchinställningarna och positionen på bildskärmen.

  ADDRESS <= std\_logic\_vector(to\_unsigned(y\_counter / 2 \* 320 + x\_counter / 2, ADDRESS'length))

             when x\_counter <= 639 and y\_counter <= 479 else

             (OTHERS => '0');

### Adressberäkning för VGA sync

I VGA Sync-modulen spelar adressberäkningen en central roll för att hantera skärmkoordinater och skapa en unik adress för DP-RAM (Double Port RAM). DP-RAM möjliggör samtidig läsning och skrivning från två olika portar. Nedan beskrivs detaljerat hur adressen beräknas och används.

ADDRESS <= std\_logic\_vector(to\_unsigned(y\_counter / 2 \* 320 + x\_counter / 2, ADDRESS'length))

when x\_counter <= 639 and y\_counter <= 479 else

(OTHERS => '0');

Denna kod visar att om x\_counter och y\_counter är inom gränserna för VGA-upplösningen (0 till 639 och 0 till 479), beräknas adressen genom att applicera en formel på de delade koordinaterna. Annars sätts adressen till noll.

Den beräknade adressen används för att hantera läsning och skrivning till DP-RAM, möjliggörande av samtidig åtkomst från olika portar. Detta kopplar samman den interna representationen av bilden med dess visuella representation och säkerställer korrekt rendering på skärmen i realtid. Adressberäkningen är därmed en kritisk komponent för att uppnå korrekt synkronisering och bildpresentation samtidigt som double port-funktionen för DP-RAM utnyttjas.

**VHDL kod**

  ADDRESS <= std\_logic\_vector(to\_unsigned(y\_counter / 2 \* 320 + x\_counter / 2, ADDRESS'length))

             when x\_counter <= 639 and y\_counter <= 479 else

             (OTHERS => '0');

-- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  --                          Sync pulses

  -- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  VGA\_HS <= '0' when x\_counter >= 660 and x\_counter <= 755 else

            '1';

  VGA\_VS <= '0' when y\_counter = 494 else

            '1';

  -- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  -- RGB signals

  -- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  VGA\_R <= (OTHERS => '1') when DATA(2) = '1' and

                                (x\_counter >= 0 and x\_counter <= 639) and

                                (y\_counter >= 0 and y\_counter <= 479) else

           (OTHERS => '0');

  VGA\_G <= (OTHERS => '1') when DATA(1) = '1' and

                                (x\_counter >= 0 and x\_counter <= 639) and

                                (y\_counter >= 0 and y\_counter <= 479) else

           (OTHERS => '0');

  VGA\_B <= (OTHERS => '1') when DATA(0) = '1' and

                                (x\_counter >= 0 and x\_counter <= 639) and

                                (y\_counter >= 0 and y\_counter <= 479) else

           (OTHERS => '0');

# Double-Port Ram

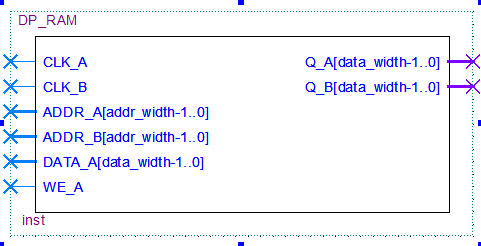
DP\_RAM har konfigurerats för att hantera grafikdata som kan visas på en VGA-monitor. Implementeringen i FPGA används för att kontrollera VGA-gränssnittet, där varje pixel representeras av 3 bitar för färger (R, G och B). Denna konfiguration möjliggör representation av en bred färgskala och ger flexibilitet för grafisk visualisering.

**DATA\_WIDTH**: Varje pixel representeras av 3 bitar för färger (R, G, B).

**DATA\_LENGTH:** Antalet dataelement i RAM-enheten, vilket påverkar den totala kapaciteten hos DP\_RAM.

**ADDR\_WIDTH:** Bredden på adressbussen för att adressera olika dataelement i RAM-enheten.

DP\_RAM är integrerad med de två klocksignalerna, CLK\_A och CLK\_B, vilket möjliggör en synkroniserad och parallell hantering av dataåtkomst. Den stöder även Write Enable (WE\_A) för att möjliggöra skrivoperationer och två utgångar, Q\_A och Q\_B, för att returnera läsdata från de olika portarna.



Figur 3: Överblick av DP\_ram arkitektur

Följande två tabeller visar in och utgångar samt generic data som används för DP\_ram.

Tabell 3: Visar In-och utgångar för DP\_RAM.vhd

|  |  |  |
| --- | --- | --- |
| **IN** | **Beskrivning** |  |
| CLK\_A | Systemklocka | 50Mhz |
| CLK\_B | VGA klocka | 25Mhz |
| ADDR\_A | Adress port A | Std\_logic\_vector (17 bits) |
| ADDR\_B | Adress port B | Std\_logic\_vector (17 bits) |
| DATA\_A | Data port A | Std\_logic\_vector (3 bits) |
| WE\_A | Write enable | std\_logic |
| **OUT** |  |  |
| Q\_A | Data ut, port A | Std\_logic\_vector (3 bits) |
| Q\_B | Data ut, port B | Std\_logic\_vector (3 bits) |

Tabell 4: visar generic data för DP\_RAM.vhd

|  |  |
| --- | --- |
| DATA\_WIDTH | Integer = 3, (RGB) |
| DATA\_LENGTH | Integer = 76800 (Data element RAM) |
| ADDR\_WIDTH | Integer = 17 (Element adress) |

## VHDL kod

library ieee;

  use ieee.std\_logic\_1164.all;

  use ieee.numeric\_std.all;

entity DP\_RAM is

  generic (

    DATA\_WIDTH  : integer := 3;     -- Colors, (RGB)

    DATA\_LENGTH : integer := 76800; -- Data elements in the RAM

    ADDR\_WIDTH  : integer := 17     --  Element addresses

  );

  port (

    CLK\_A  : in    std\_logic;                                   -- 50 Mhz

    CLK\_B  : in    std\_logic;                                   -- 25 Mhz

    ADDR\_A : in    std\_logic\_vector((ADDR\_WIDTH - 1) downto 0); -- 17 bits

    ADDR\_B : in    std\_logic\_vector((ADDR\_WIDTH - 1) downto 0); -- 17 bits

    DATA\_A : in    std\_logic\_vector((DATA\_WIDTH - 1) downto 0); -- 3 bits

    --    data\_b : IN STD\_LOGIC\_VECTOR((DATA\_WIDTH - 1) DOWNTO 0); -- 3 bits data

    WE\_A   : in    std\_logic;

    Q\_A    : out   std\_logic\_vector((DATA\_WIDTH - 1) downto 0); -- 3 bits

    Q\_B    : out   std\_logic\_vector((DATA\_WIDTH - 1) downto 0)  -- 3 bits

  );

end entity DP\_RAM;

architecture RTL of DP\_RAM is

  -- Build a 2-D array type for the RAM

  SUBTYPE word\_t IS std\_logic\_vector((DATA\_WIDTH - 1) downto 0);

  type memory\_t is ARRAY(DATA\_LENGTH - 1 downto 0) OF word\_t;

  -- Declare the RAM

  SHARED variable ram : memory\_t;

begin

  -- Port A (Read/Write)

  PORT\_A\_RAM : process (CLK\_A) is

  begin

    if (rising\_edge(CLK\_A)) then

      if (WE\_A = '1') then

        ram(to\_integer(unsigned(ADDR\_A))) := DATA\_A; -- WRITE

      end if;

      Q\_A <= ram(to\_integer(unsigned(ADDR\_A)));      -- READ

    end if;

  end process PORT\_A\_RAM;

  -- Port B (Read only) VGA

  PORT\_B\_RAM : process (CLK\_B) is

  begin

    if (rising\_edge(CLK\_B)) then

      Q\_B <= ram(to\_integer(unsigned(ADDR\_B)));

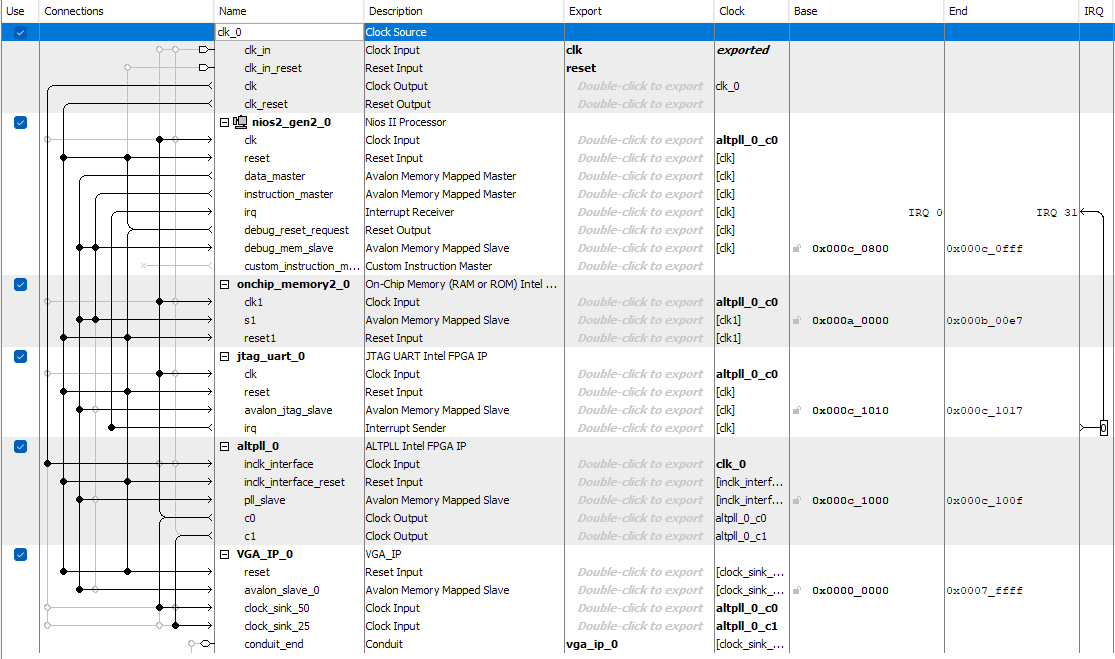
    end if;

  end process PORT\_B\_RAM;

end architecture RTL;

# Platform designer

Följande bild visar hur systemet har kopplats ihop med hjälp av Intel Quartus Platform Designer.



Figur 4: Quartus Platform designer av VGA\_IP

# Testprotokoll

Följande testprotokoll är utformat för att testa färg och färgskala vid användning av VGA\_IP och kan utökas vid behov.

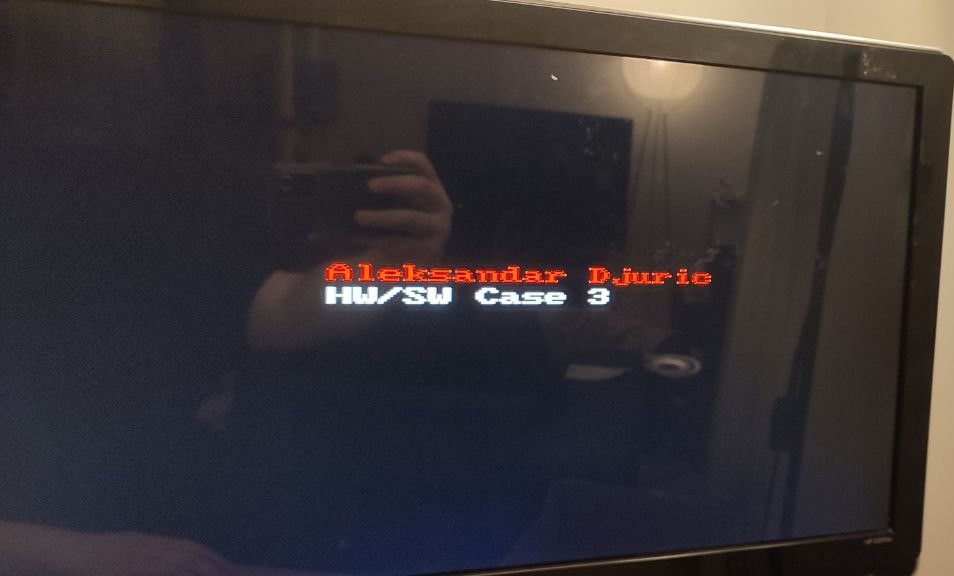
Tabell 5: Testprotokoll

|  |  |  |
| --- | --- | --- |
| **Testfall** |  | **Fungerar (JA/NEJ)** |
| **1** | Skriv text med färg RGB |  |
| **2** | Skriv text med färg RGB + bakgrund färg |  |
| **3** | Färgskala |  |
| **4** |  |  |
| **5** |  |  |

# Projekt kostnad

Kostnad för projektet är 40h\*600kr = 24000kr

# Bilaga



Figur 5: Bild som visar utskrift på DE-10 Lite kort