TEIS

VGA-Protokollet

Teori med implementering i VHDL

Lasse Karagiannis

2016-12-20

Sammanfattning
VGA-protokollet har beskrivits och en delvis färdig implementering av en VGA-kontroller har bearbetats och fåtts att fungera.

Kravspecifikationen för uppgiften ges av Tabell 1 nedan.

Krav id	Beskrivning	Utfört Ja/nej
	Konstruktionskrav	
1	Skapa ett Quartus-projekt med namnet vhdl2_uppgift_1a	JA
2a	Kravspecifikationen är att konstruera en VGA-kontroller, kommentera koden, verifiera med testbänk (före valideringen) och validera koden på DE1-kortet.	JA
2b	VGA-protokollet ska följas och dela 50 MHz klockan i en process för att få 25 MHz.	JA
	Testprotokoll	
3	Testfall: VGA protokollets styrsignaler testas med ModelSim. Följande signaler ska verifieras: VGA_HS, VGA_VS, VGA_CLK och VGA_BLANK_N. Se bilaga. Validering på FPGA kortet: Testfall 1: Rita en röd fyrkant i övre vänster hörn när KEY(0) trycks ned Testfall 2: Rita en grön fyrkant i nedre vänster hörn när KEY(1) trycks ned Testfall 3: Rita en blå fyrkant i nedre höger hörn när KEY(2) trycks ned Dessa färger ska överlappa varandra och de ska blandas i mitten av skärmen. Se figur 14 i bilaga D.	JA
	VHDL-kod	
4	Regler och riktlinjer för VHDL och C ska följas.	JA
	Verifierings-/Valideringskrav	
5a	Verifiera med ModelSim.	JA
5b	Validera på DE1-kortet. Fyll i testprotokollet.	JA
	Rapportkrav	
6	Skriv en rapport (pdf eller word): 1) Framsida med titel, ditt namn, e-post, datum och en kort sammanfattning 2) Innehållsförteckning, sidnumrering och kapitel nummer 3) Kravspecifikation, eget kapitel. 4) Beskriv VGA-protokollet. Ett tydligt pulsdiagram i förhållande till H- och V-räknarna för VGA-signalerna: VGA_HS, VGA_VS, VGA_CLK, VGA_BLANK_N. (detta bör göras innan koden skrivs!), eget kapitel. 5) Testprotokoll för verifiering och ett enkelt protokoll för validering (olika protokoll), eget kapitel. 6) Konstruktionsbeskrivning, eget kapitel. a. System arkitektur b. Delsystem 7) Verifiering, Eget kapitel. a. Testbänken, Teststimuli (som bygger på testprotokollet för programmet) b. Bilder på resultatet från simuleringen med ModelSim (se bilaga C för exempel) c. Ifyllt testprotokoll för verifiering 8) Validering, beskriv resultatet med ett foto. a. Ifyllt testprotokoll för validering (ej samma som verifiering) 9) Bilaga a. VHDL-fil b. Do-fil c. Testbänk Ett dokumentationskrav är också att figurer och tabeller ska ha numrering och beskrivning. Figurbeskrivning under figuren och tabellbeskrivning ovanför tabellen.	JA

!

VGA-protokollet använder sig av analoga signaler för att rita upp skärmen, vilket är p.g.a. tidigare katodstråle teknologi för styrning av elektronstrålens passeringar över fosfor belagd yta för uppritning av bokstäver och bild.

En vanligt VGA-mode är bildvisning på 640 x 480 pixlar med 16 färger. Avsikten är här att beskriva hur detta fungerar med avseende på signaler protokollet kräver.

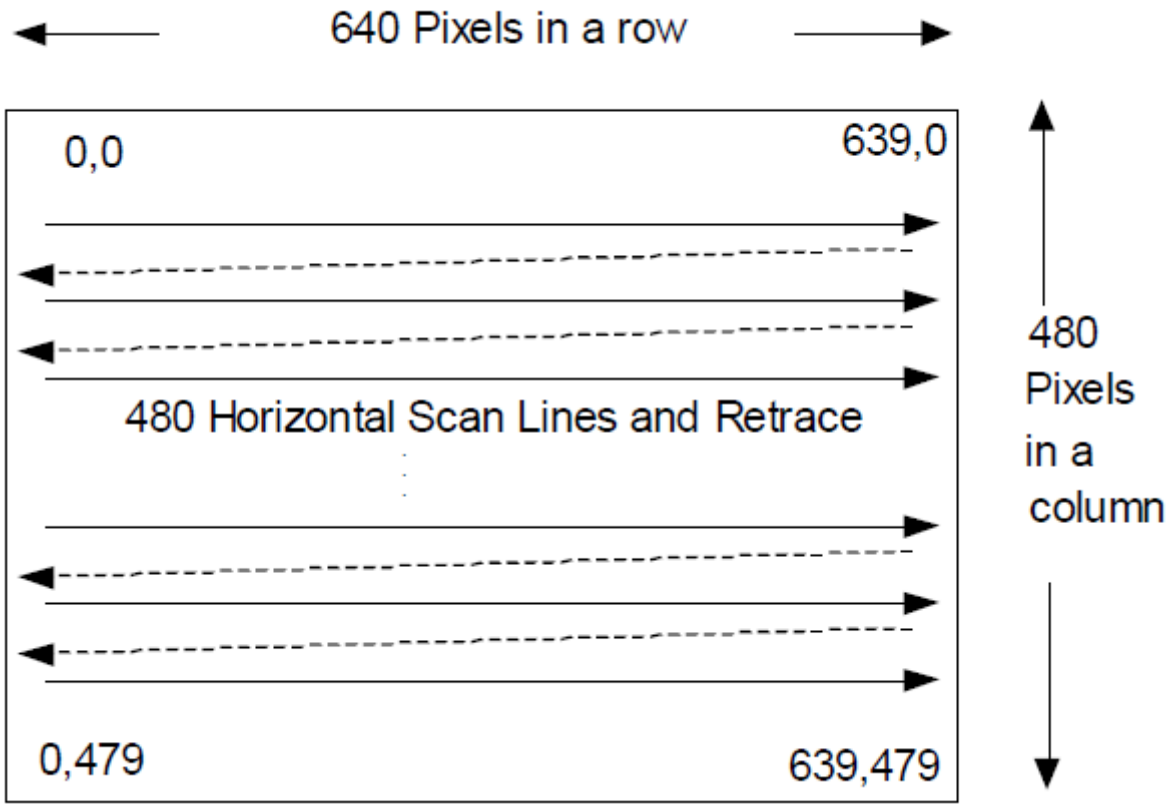
För att underlätta förklaringar kommer terminologi från katodstråle-teknologin att användas. Skärmen ritas upp 60 ggr per sekund, där en hel skärm sägs målas eller ritas upp under ett elektronstråle svep.

Vidare används uttrycket att elektronstrålen sveper från rad till rad.

För att underlätta beskrivningen av när synkningssignaler och blankningssignalen infaller tidsmässigt i förhållande till elektronstrålens svep över skärmen, så generaliseras begreppet pixlar genom att skärmen utökas till att förutom att innehålla pixlar som är synliga för ögat, till att även innehålla pixlar som inte är synliga för ögat, på sås sätt kan synksignalernas pulsavsnitt anges i termer av pixelpositioner istället för att ange tidpunkter.

Så för att underlätta beskrivningen av när synk-signaler och blanksignalen inträffar så gör vi antagandet att 640x 480 skärmen egentligen är 800 x 525. Detta alltså för att kunna ange synk och blanksignalernas påslag i termer av pixelpositioner, vilket är endast för att underlätta framställningen.

Vi har 640 pixlar i x-led, och 480 pixlar i y-led enligt Figur 1



" #$% & $'% & (

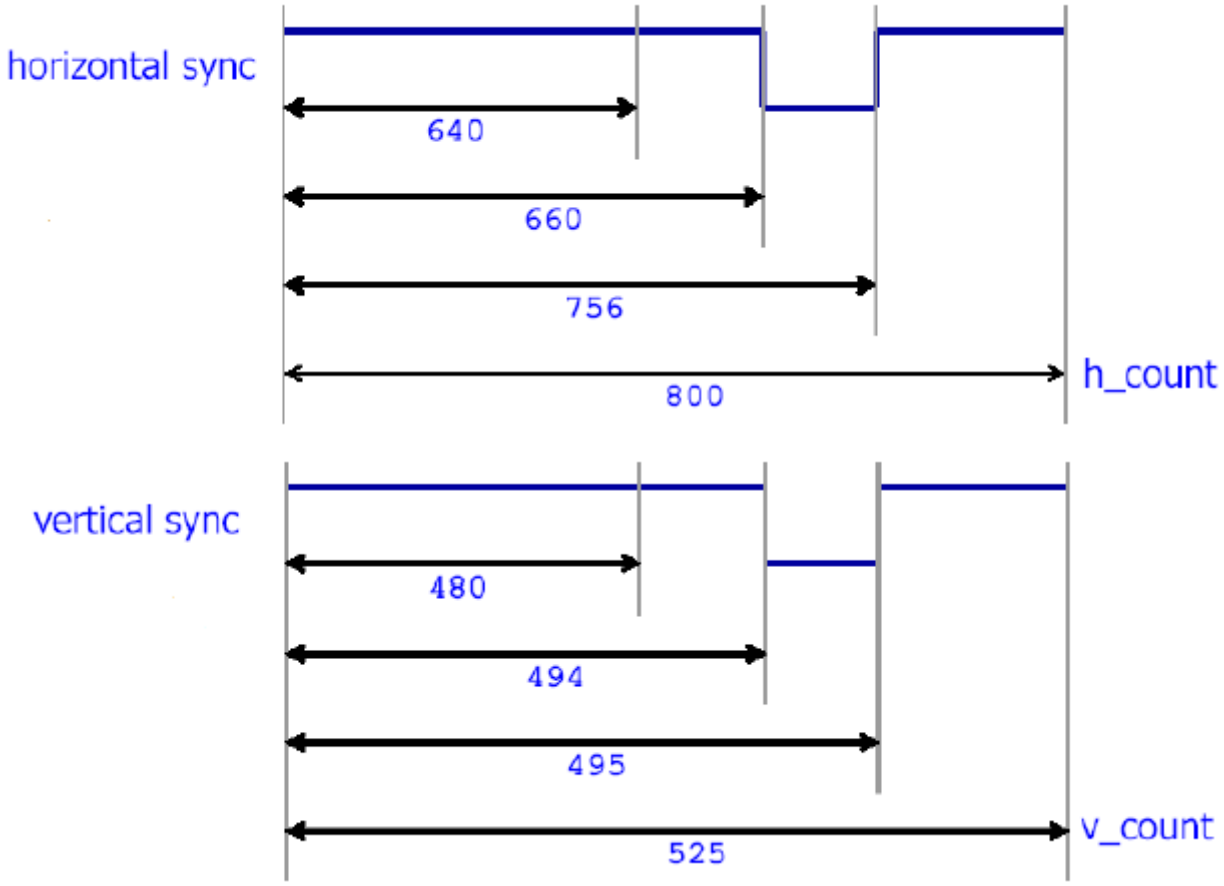
Varje pixel består av tre belagda cirklar av olika fosfortyp, som emitterar rött, grönt respektive blått ljus vid fosforescens, där den emitterade ljusintensiteten är proportionell emot den infallande partikel intensiteten. Färgbilder kan därför skapas genom adekvat kontroll av tre elektronstrålar som var för sig bestryker de röda, gröna och blå fosforskikten respektive.

Hur lång tid tar elektronstrålens passering över en pixel? Ögat kräver en bild-uppdateringsfrekvens på minst 30Hz, och om vi väljer det dubbla 60Hz, så betyder det att tiden för att rita 640*480 = 307200 pixlar måste vara 1/60 = 0.01666s, men eftersom vi behöver synk-signaler så måste elektronstrålen svepa längre på samma tid, motsvarande 800*525 pixlar.

800*525 = 420 000 pixlar måste kunna svepas över på 0.0166s. Antal sekunder per pixel blir 0.0166/420 000 =3.966E-8 eller 39ns, vilket ger inversen, den s.k. pixelfrekvensen, antalet pixlar per sekund till 25 210 084, eller 25.2 MHz. Detta ger att klockan till VGA-avkodningen måste halveras då systemklockan på 50MHz används.

För att elektronstrålens passeringar över skärmen skull synkroniseras kolumnvis i x och radvis i y, så behövdes synksignaler som gick från hög till låg nivå en bestämd tid efter att elektronstrålen ritat svept över varje rad, samt en en bestämd tid efter att elektronstrålen ritat en en hel skärm, dvs, efter 480 rader. Se Figur 2 för ett tidschema för synksignalerna, normaliserat till tiden det tar att rita en

pixel.



" &) * + + (,

Horisontella synksignalen VGA_HS skall enligt bilden gå låg från pixel 659 till 755(pixelräkningen i programmet börjar från noll, till skillnad från Figur 2, som börjar på ett).
Vertikala synksignalen VGA_VS skall enligt Figur 2 vara låg under hela rad 493 (radräkningen börjar också från noll)

Förutom synkroniseringssignaler så måste blankningssignalen finnas. Blankningssignalen VGA_BLANK är hög under den tiden då elektronstrålen avses att rita skärmen och går låg då elektronstrålen är ”utanför” ritfältet. Då bilden går från pixel 0 till 639 i x-led, och från 0 till 479 i y-led, och hela svepet slutar vid 799 i x-led respektive 524 i y-led, så skall blankningssignalen gå låg från 640 till 799 i x-led, samt från 480 till 524 i y-led.

Sammanfattningsvis gäller alltså följande:

Elektronstrålen sveper från pixel 0 till 799 i x-led

Elektronstrålen sveper från rad 0 till rad 524 i y-led

Horisontella synksignalen VGA_HS går låg då elektronstrålen befinner sig vid pixlar 659 till 755 i x-led.

Vertikala synsignalen VGA_VS går låg då elektronstrålen befinner sig på rad 493

Blankningssignalen VGA_BLANK, går låg då elektronstrålen befinner sig utanför ritfältet

dvs. från pixlar 640 i till 799 i x-led för raderna 0 till 479, och för samtliga pixlar i x-led för raderna 480 till 524.

För att få en flimmerfri bild så skall VGA_CLK vara 25.2MHz för bilduppdateringsfrekvensen 60Hz.

-

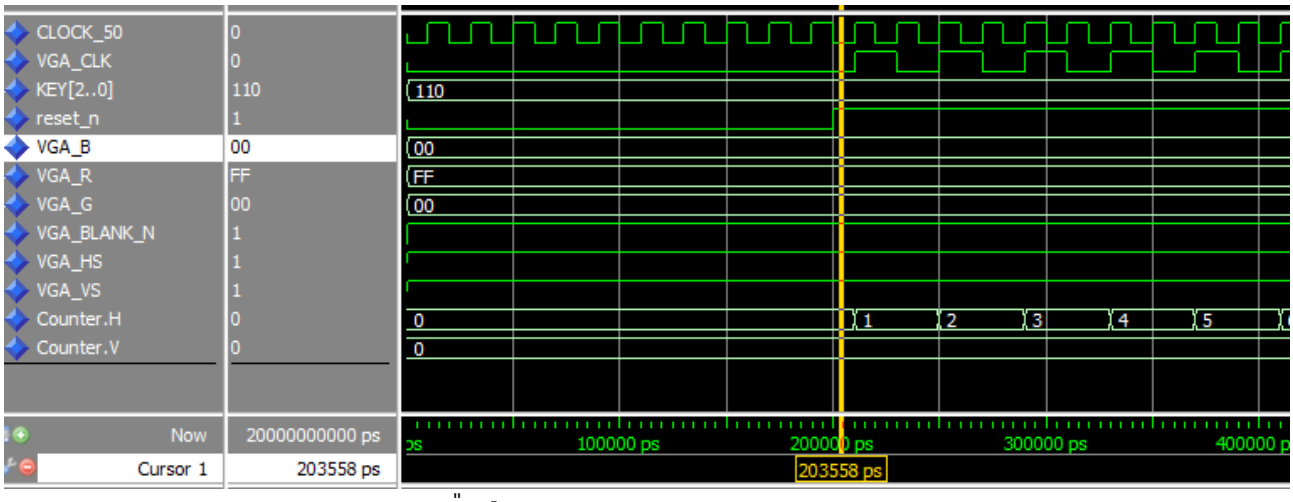
Följande testas. Knapp 0 trycks in vilket ska generera en röd fyrkant.
Fyrkanten ska ockupera pixlar 0 till 360 i x-led och rader 0 till 320 p.g.a. raden

VGA_R <= (others => '1') when (unsigned(x_counter) >= 0 AND unsigned(x_counter) <= 360 AND unsigned(y_counter) >= 0 AND unsigned(y_counter) <= 320) AND KEY(0) ='0'
else (others => '0');

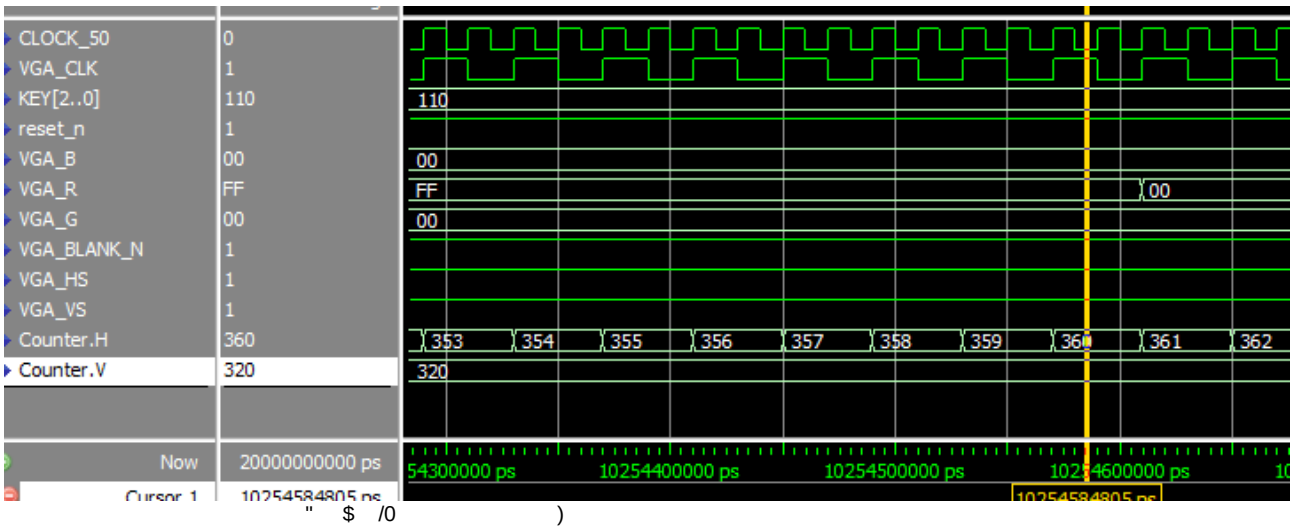
Vidare skall synksignalerna och blankningssignalerna vara korrekta. Se Tabell 1 för testprotokollet:

Signal	Krav	Godkänt/Ej godkänt
VGA_CLK	25MHz	Godkänt, Figur 3
VGA_R	0xFF, för 0 <= x <= 360 OCH 0<= y <= 320	Godkänt, Figur 3 och Figur 4
VGA_HS	'0', för 659<= x <=755, '1' annars	Godkänt, Figur 5 och Figur 6
VGA_VS	´0´, för y = 493, '1' annars	Godkänt, Figur 7
VGA_BLANK	'1', för 0<=x<= 639 OCH 0<= y <= 479, '0' annars	Godkänt, Figur 8, Figur 9, Figur 10

Figur 3 veriferar att VGA_CLK är 25MHz, samt att VGA_R är 0xFF från pixel 0, rad 0, samt att VGA_HS, VGA_VS och VGA_BLANK är höga då elektronstrålen är i början av ritfältet-

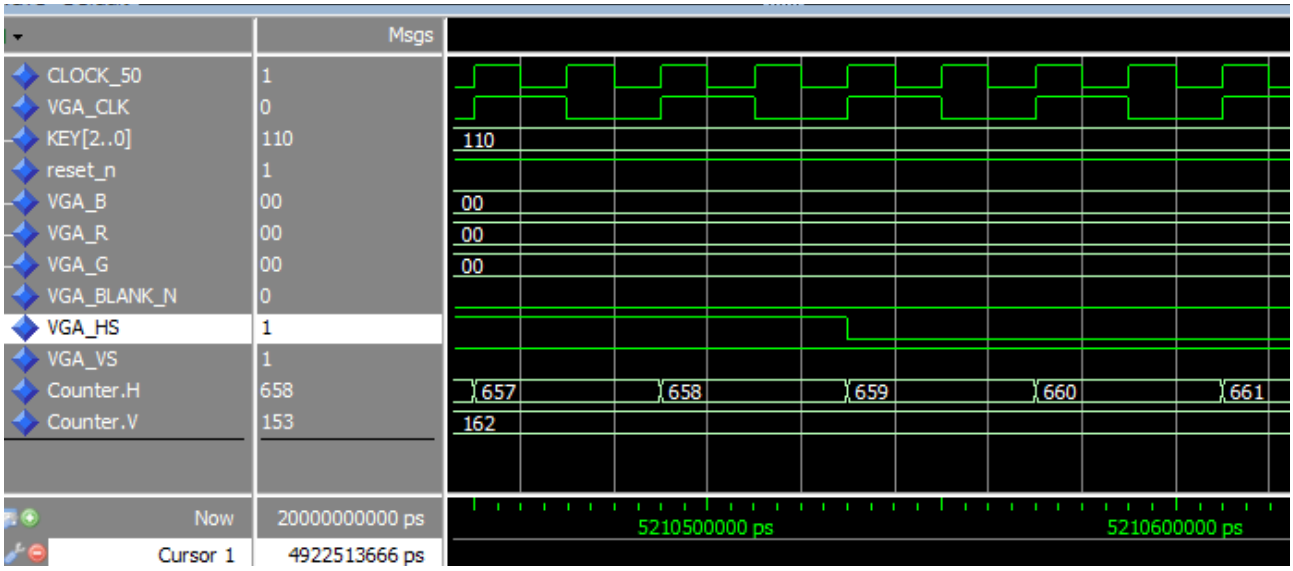


Figur 4 visar att VGA_R går låg efter pixel 360 på rad 320



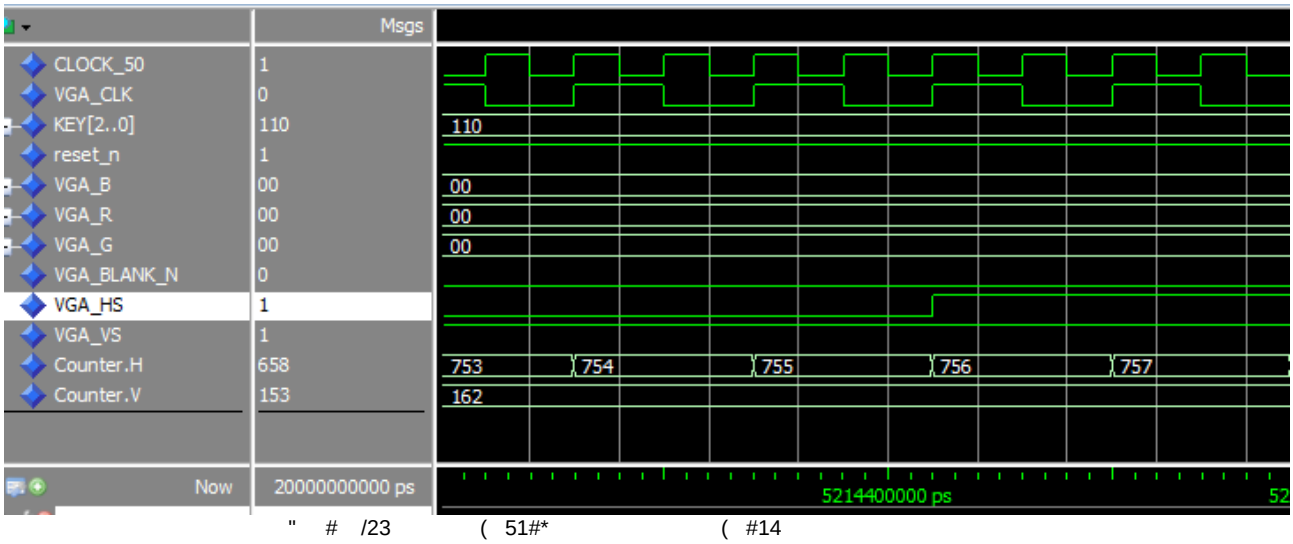
" $ /0)

Figur 5 visar att VGA_HS är låg fr.o.m pixel 659.

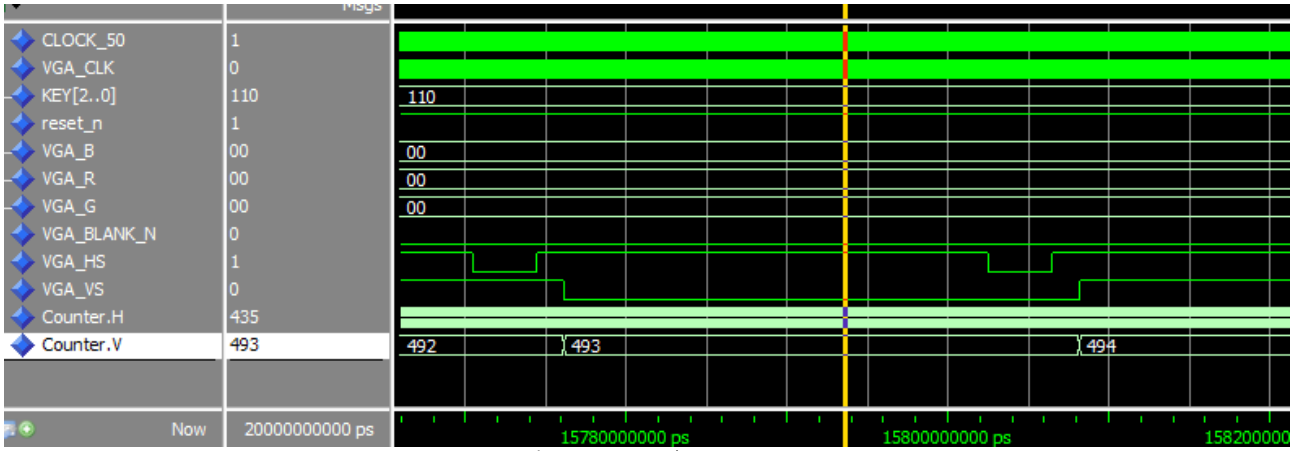


" 1 /23 (#14

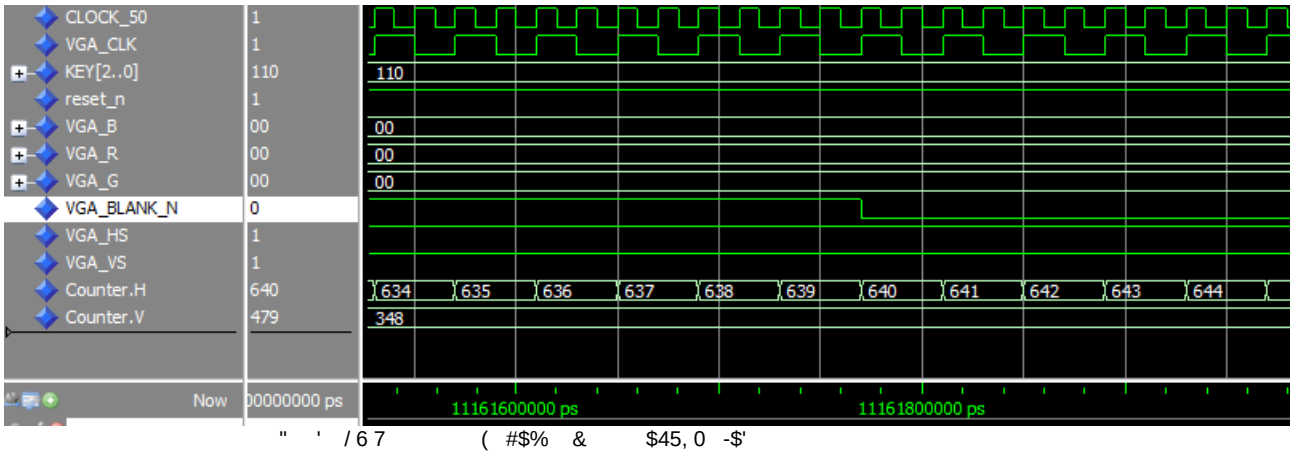
Figur 6 visar att VGA_HS går hög igen efter pixel 755

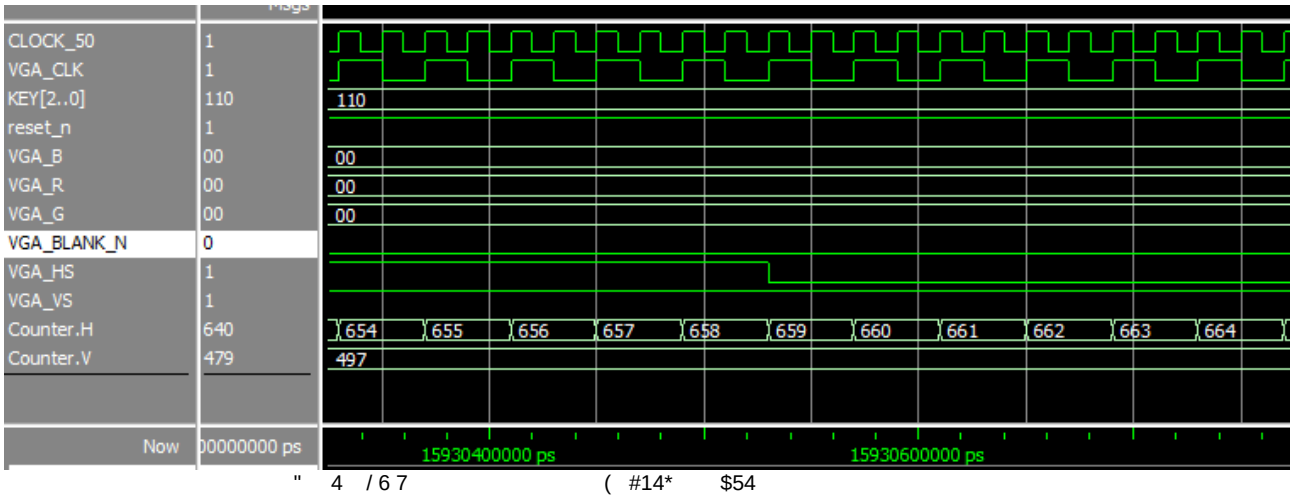


Figur 7 visar att VGA_VS är låg under hela rad 493

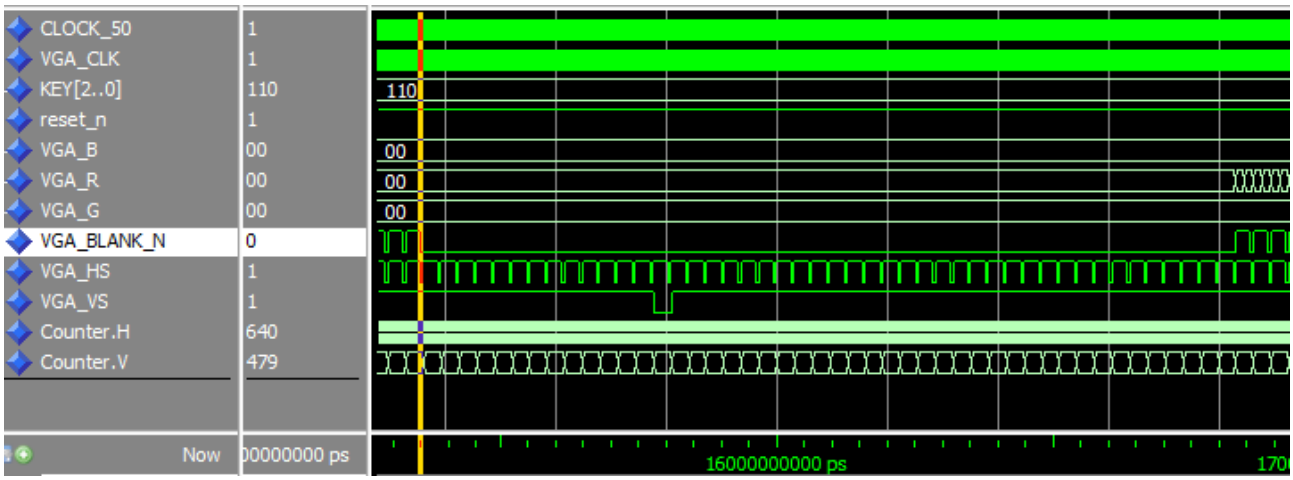


Figur 8 indikerar att VGA_BLANK går låg för varje rad fram till 497 fr.o.m. pixel 640, i figuren har radnumret helt godtyckligt valts till 348. Figur 9 visar då VGA_BLANK går låg för sista gången vid pixel 640, därefter är VGA_BLANK låg för alla pixlar fr.o.m. rad 480 till rad 524 då ett helt elektronsvep är avklarat, Figur 10.




CLOCK_50
VGA_CLK
KEY[2..0]
reset_n
VGA_B
VGA_R
VGA_G
VGA_BLANK_N
VGA_HS
VGA_VS
Counter.H
Counter.V
1
1
110
1
00
00
00
0
1
1
640
479
110
00
00
00
654
655
656
657
658
659
660
661
662
663
664
497
Now
15930400000 ps
15930600000 ps

" 4 / 6 7 (#14* $54


CLOCK_50
VGA_CLK
KEY[2..0]
reset_n
VGA_B
VGA_R
VGA_G
VGA_BLANK_N
VGA_HS
VGA_VS
Counter.H
Counter.V
1
1
110
1
00
00
00
0
1
1
640
479
110
00
00
00
Now
16000000000 ps

" % / 6 7 + , ,&, $'% + , 1 $

$

Validering har gjorts genom att verifiera att fyrkanterna ritas upp genom att trycka på knapparna KEY[2..0]. Resultatet är såsom specifikationen kräver, nämligen att röd rektangel ritas upp då KEY0 hålls intryckt, grön då KEY1 hålls intryckt och blå då KEY2 hålls intryckt.
För att göra en uttömmande validering så skulle man kunna använda Qsys ”In circuits and probes”, men detta har inte kunden begärt, så däeför ha endast en ytlig okulär inspektion gjorts av resultatet.

1 +

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

use ieee.numeric_std.all;

LIBRARY work;

entity vhdl2_uppgift_1a is

port(
	CLOCK_50	: in std_logic;
	reset_n		: in std_logic;
	KEY		: in std_logic_vector(2 downto 0);
	VGA_CLK		: out std_logic;
	VGA_VS		: out std_logic;
	VGA_HS		: out std_logic;
	VGA_BLANK_N	: out std_logic;
	VGA_R		: out std_logic_vector(7 downto 0);
	VGA_G		: out std_logic_vector(7 downto 0);
	VGA_B		: out std_logic_vector(7 downto 0)

);

end vhdl2_uppgift_1a;

architecture rtl of vhdl2_uppgift_1a is

	signal x_counter : unsigned(9 downto 0);
	signal y_counter : unsigned(9 downto 0);

	---------- could use a record for x/y counters ----------

	type HV_type is -- Horizontal/Vertical type
		record
			H : integer range 0 to 1023; -- Horizontal (x) signal
			V : integer range 0 to 525; -- Vertical (y) signal
		end record;

	signal counter			: HV_type;--for simulation only
	signal clk_25			: std_logic;
	signal first_flip_flop		: std_logic_vector(2 downto 0);
	signal second_flip_flop		: std_logic_vector(2 downto 0);


begin

------------------ 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;


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

		Metastability:process(CLOCK_50, reset_n)
		begin
			if reset_n = '0' then
			   -- Something might need to be reset. -Forget it Lennart!
			elsif rising_edge(CLOCK_50) then
			   -- There should be a double D-flip-flop for the inputs
			   first_flip_flop <= KEY;
			   second_flip_flop <= first_flip_flop;
			end if;
		end process;
```

```
------------------ Counters process ------------------

              process(clk_25, reset_n)
              begin
                     if reset_n = '0' then
                        -- clear counter signals
                        x_counter <= (others => '0');
                        counter.H <= 0;

                        y_counter <= (others => '0');
                        counter.V <= 0;
                     elsif rising_edge(clk_25) then

                        -- counters
                            -- increment x_counter (counter.H) every clock pulse

                            if unsigned(x_counter) >= 799 then
                               x_counter <= (others => '0');
                              counter.H <= 0;

                                   -- increment y_counter (counter.V) for every X = 799

                                   if unsigned(y_counter) >= 525 then
                                     y_counter <= (others => '0');
                                     counter.V <= 0;
                                   else
                                     y_counter <= y_counter + 1;
                                     counter.V <= counter.V + 1;
                                   end if;

                            else
                                x_counter <= x_counter + 1;
                                counter.H <= counter.H +1;
                            end if;
                     end if;
              end process;


------------------ Concurrent statements ------------------

-- Output divided clock to VGA_CLK

       VGA_CLK <= clk_25;

------- Sync pulses -------

       VGA_HS <= '0' when (unsigned(x_counter) >= 659) AND (unsigned(x_counter) <= 755)

                                          else '1';

       VGA_VS <= '0' when (unsigned(y_counter)) = 493 else '1';

-- Blanking

       VGA_BLANK_N <= '1' when (unsigned(x_counter) <= 639) and (unsigned(y_counter)

                     <= 479) else '0';


------- RGB signals for validation -------

       VGA_R <= (others => '1') when
                                    (unsigned(x_counter) >= 0 AND unsigned(x_counter) <= 360   AND
                                    unsigned(y_counter) >= 0 AND unsigned(y_counter) <= 320)   AND
                                     KEY(0) ='0' else (others => '0');


       VGA_G <= (others => '1') when
                                    (unsigned(x_counter) >= 0 AND unsigned(x_counter) <= 410  AND
                                    unsigned(y_counter) >= 280 AND unsigned(y_counter) <= 480) AND
                                     KEY(1) ='0' else (others => '0');



       VGA_B <= (others => '1') when
                                     (unsigned(x_counter) >= 300 AND unsigned(x_counter) <= 640 AND
                                      unsigned(y_counter) >= 240 AND unsigned(y_counter) <= 480)AND
                                       KEY(2) ='0' else (others => '0');

------------------ Concurrent statements ------------------
       -- Red         x from 0 to 360,      y from 0 to 320
       -- Green       x from 0 to 410,      y from 280 to 480
       -- Blue        x from 300 to 640,    y from 240 to 480

end rtl;
```

+

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY vhdl2_uppgift_1a_vhd_tst IS
END vhdl2_uppgift_1a_vhd_tst;
ARCHITECTURE vhdl2_uppgift_1a_arch OF vhdl2_uppgift_1a_vhd_tst IS
-- constants
constant sys_clk_period: TIME:=20ns;
-- signals
SIGNAL CLOCK_50 : STD_LOGIC;
SIGNAL KEY : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL reset_n : STD_LOGIC;
SIGNAL VGA_B : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL VGA_BLANK_N : STD_LOGIC;
SIGNAL VGA_CLK : STD_LOGIC;
SIGNAL VGA_G : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL VGA_HS : STD_LOGIC;
SIGNAL VGA_R : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL VGA_VS : STD_LOGIC;
COMPONENT vhdl2_uppgift_1a
        PORT (
        CLOCK_50 : IN STD_LOGIC;
        KEY : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        reset_n : IN STD_LOGIC;
        VGA_B : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        VGA_BLANK_N : OUT STD_LOGIC;
        VGA_CLK : OUT STD_LOGIC;
        VGA_G : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        VGA_HS : OUT STD_LOGIC;
        VGA_R : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        VGA_VS : OUT STD_LOGIC
        );
END COMPONENT;
BEGIN
        i1 : vhdl2_uppgift_1a
        PORT MAP (
-- list connections between master ports and signals
        CLOCK_50 => CLOCK_50,
        KEY => KEY,
        reset_n => reset_n,
        VGA_B => VGA_B,
        VGA_BLANK_N => VGA_BLANK_N,
        VGA_CLK => VGA_CLK,
        VGA_G => VGA_G,
        VGA_HS => VGA_HS,
```

```
        VGA_R => VGA_R,
        VGA_VS => VGA_VS
        );
clock : PROCESS
-- variable declarations
BEGIN
      CLOCK_50 <= '0';
               wait for sys_clk_period/2;
               CLOCK_50 <= '1';
               wait for sys_clk_period/2;
END PROCESS clock;

reset_n <= '0', '1' after 10*sys_clk_period;

always : PROCESS
-- optional sensitivity list
-- (        )
-- variable declarations
BEGIN
        -- code executes for every event on sensitivity list
                KEY <= "110";
                wait for 20ms;
WAIT;
END PROCESS always;
END vhdl2_uppgift_1a_arch;
```

5 8 !

```
transcript on
if {[file exists rtl_work]} {
        vdel -lib rtl_work -all
}
vlib rtl_work
vmap work rtl_work


vcom -93 -work work {C:/Users/root/Documents/TEIS2016/Systemkurs_VHDL/Task1/vhdl2_uppgift_1a.vhd}


vcom -93 -work work
{C:/Users/root/Documents/TEIS2016/Systemkurs_VHDL/Task1/simulation/modelsim/vhdl2_uppgift_1a.vht}


vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L cyclonev -L rtl_work -L work
-voptargs="+acc"  vhdl2_uppgift_1a_vhd_tst


add wave *


add wave -position insertpoint  \
sim:/vhdl2_uppgift_1a_vhd_tst/i1/counter.H
add wave -position insertpoint  \
sim:/vhdl2_uppgift_1a_vhd_tst/i1/counter.V


view structure
view signals
run 20 ms
```