Digitalt oscilloscop

Johan Böhlin 840319-7117
Martin Johansson 800620-5556
Anders Palmér 820301-4959
Jonatan Åkerlind 820711-6958

13 december 2006

Innehåll

1 Funktionsbeskrivning **2**

2 Beskrivning av lösningen **2**
2.1 ADC . . . 2
2.2 VGA . . . 3
2.3 CPU . . . 3
2.4 Klockning . . . 4

3 Problem vid konstruktionen **4**
3.1 Övergripande . . . 4
3.2 ADC-specifika problem . . . 4
3.3 VGA-specifika problem . . . 4

4 Resultat **5**

A Manual **6**

B Kod **7**
B.1 Toppnivå . . . 7
B.1.1 Mappning - oscil . . . 7
B.1.2 Minneskontroller - memctrl . . . 9
B.2 ADC . . . 10
B.2.1 Mappning - toplevel . . . 10
B.2.2 Initieringsmodul - Init . . . 11
B.2.3 I^2C-modul - i2c . . . 12
B.2.4 Sampelmodul - sample . . . 14
B.2.5 Interfacemodul - interface . . . 15
B.2.6 Minnesmodul - Memory . . . 16
B.3 VGA . . . 18
B.3.1 VGA-drivenhet - vga_controller . . . 18
B.4 CPU . . . 19
B.4.1 ADC control register interface - cpu_adc_reg . . . 19
B.4.2 ADC user interface interface - cpu_ui_interface . . . 20
B.4.3 SRAM memory interface - meminterface_cpu . . . 21
B.4.4 SDRAM memory interface - vga_memory_interface . . . 22
B.5 Programkod . . . 24
B.5.1 Huvudprogram . . . 24
B.5.2 Plotfunktion . . . 25
B.5.3 Teckenhantering . . . 25

C Syntesresultat **28**

D Simuleringsresultat **29**

1 Funktionsbeskrivning

Målet är att konstruera ett tvåkanals digitalt oscilloskop som läser in data via ljudkortet och presenterar det på en VGA-skärm. Mätdatan ska sparas i ett ram för vidare behandling och kunna presenteras i olika tids- och amplitudsupplösning. Olika intressanta värden som period, amplitud samt stig- och falltid skall kunna presenteras på skärmen, även rutmönster och skalor. Det ska finnas ett användarinterface med menysystem för att kunna ändra triggernivåer, tidsupplösning, etc.

2 Beskrivning av lösningen

Oscilloskopet består av flera delar internt i FPGA:n och externt. Internt har vi en mjuk CPU NIOS-II, en samplings-modul och en VGA-modul. Utanför har vi ett SDRAM, ett SRAM, en audio CODEC (ADC), en DAC och en bildskärm.

Samplingsmodulen samplar en signal via audiocodecen och skriver signalen i SRAMet. CPUn omvandlar och mappar den digitalt representerade signalen i SRAMet till en färgkod för varje pixel, lägger på informationstext och skriver ner bitmönstret till DRAMet. VGA-modulen läser sedan bitmönstret i DRAMet och lägger ut det som en bild på bildskärmen via DAC:en.

2.1 ADC

ADC-blocket är kopplad mot användarintefaceswitchar, SRAMet, CODECen, klockan och CPUn. Denna modul använder CODECen för att sampla en signal för att enligt vissa kriterier digitalt representera denna i SRAMet.

ADC-blocket består av två delar, en initieringsdel och en huvuddel för att kontinuerligt läsa sampel från CODECen, kontrollera triggvillkor samt skriva sampel i SRAMet.

Initieringsdelen initierar CODECen genom att via I^2C konfigurera CODECen att kontinuerligt i ca 96 kHz sampla både vänster och höger kanal på line-in. Denna del består av en I^2C-modul och en init-modul. Init-modulen skickar konfigurationsdata parallellt till I^2C-modulen. Denna modul skickar i sin tur datan vidare seriellt enligt I^2C-protokollet.

Huvuddelen läser kontinuerligt in sampel från CODECen och beroende på olika kriterier skriver det till SRAMet.

Huvuddelen består av tre delar: memory, sample och interface. Samplemodulen tar kontinuerligt (ca 96k gånger per sekund) emot ett seriellt sampelvärde som den lägger ut parallellt på den interna databussen och indikerar sedan att ett nytt sampel finns. Interface-modulen bestämmer enligt vissa villkor om en sampel ska skrivas till minnet eller ej. Varje gång ett villkor är uppnått skriver den nuvarande sampel och de följande (n*1024-1) sampels till minnet, där n är inställt via användargränsnittets switchar. Villkoren som ska gälla för att den ska skriva till minnet är att CPUn inte använder minnet och att en triggernivå är uppnådd och att det fortfarande finns sampels att skriva. När interfacemodulen flaggar för skrivning skriver memory-modulen den data som ligger på databussen till minnet på adressen som motsvarar vilket nummer aktuell sampel har.

2.2 VGA

VGA-blocket består dels av en drivenhet som genererar synk- och färgsignaler till VGA-DACen och dels ett interface mellan minnet och drivenheten. I minnet sparas skärmbilden i två framebuffers där varje pixel är kodad med två bitar. I minnesinterfacet finns ett register där basadressen till aktuell framebuffer sparas.

Skärmbilden visas med upplösningen 800x600 pixlar med en vertikal uppdateringsfrekvens på 72 Hz, vilket motsvarar en horisontell uppdateringsfrekvens på 48 kHz som i sin tur innebär att pixeluppdateringsfrekvensen blir 50 MHz. För varje pixel inom skärmens aktiva område begär VGA-drivenheten motsvarande pixeladress (0 - 479 999) från minnesinterfacet. VGA-drivenheten är skriven för att lätt kunna ändra upplösning och tillgängliga färger. Det som krävs för att ändra upplösning är att generera rätt klockfrekvens och ändra på några konstanter i koden.

Minnesinterfacet har en 16 ord djup FIFO med bredden 16 bitar för att matcha DRAMets ordbredd. FIFOn fungerar som en cache mellan minnet och VGA-drivenheten för att kunna dela DRAMet mellan VGAn och CPUn. Minnesinterfacet översätter adressen från VGA-drivenheten, fyller FIFOn från DRAMet och lägger ut motsvarande två bitar för den aktuella pixeln.

DRAMet är klockat i 100 MHz och VGA-drivenheten i 50 MHz, därför arbetar minnesinterfacet med båda frekvenserna där den snabba delen fyller FIFOn och den långsamma läser ur FIFOn och lägger ut data till VGA-drivenheten.

2.3 CPU

Den mjuka processorn NIOS-II från Altera kan konfigureras på många olika sätt med hjälp av Altera's programvara SOPC Builder. Processorn konfigurerades för en databredd på 32 bitar med en instruktionscache på 512 byte. Den konstruerades utan datacache eftersom dataaccesserna bedömdes vara allt för slumpmässiga för att dra nytta av en cache och dessutom klockas DRAMet snabbare än CPUn. Processorn är även konfigurerad med följande funktioner:

- Branch prediction
- Hardware multiply
- Hardware divide
- JTAG debug

För att kommunicera med ADC och VGA konstruerades interfacemoduler som integrerades med CPUn i SOPC Builder enligt Figur **??**. Till vänster i bilden visas datavägar med arbitreringsprioritet. Följande moduler är inkluderade:

- `cpu_adc_reg_0` är registret för kontroll av ADC,
- `vga_memory_interface_0` är interfacet mellan minnet och VGA-drivenheten,
- `meminterface_cpu_0` är interfacet mot SRAMet och
- `cpu_ui_interface_0` är kopplingen mot switchar, knappar och lysdioder.

Altera's moduler mot SDRAMet och PLLen användes färdiga som dom var och modulerna mot ADCn och VGAn konstruerades helt från grunden.

Det ska vara figur här, men det fungerade inte.
interfacemoduler

NIOS IDE använder ett UNIX-liknande gränssnitt för att skriva programvara för NIOS-processorn i C-kod. Detta innebär att programvaran blir relativt enkel att skriva. Programvaran behöver dessutom inte integreras med VHDL-koden, utan tankas ner i programminnet när hårdvaran finns färdig.

2.4 Klockning

De olika modulerna klockas med tre olika frekvenser. ADCn klockas i 25 MHz, CPUn; ADC-kontrollern; SRAMet; VGA-drivenheten klockas i 50 MHz och DRAMet och minnesinterfacet mellan DRAMet och VGAn klockas i 100 MHz. På kortet finns 50 MHz och 27 MHz tillgängligt för FPGAn. För att generera önskade klockfrekvenser skapades en PLL som genererar 25 MHz och 100 MHz ur 50 MHz. PLLen har ett kontrollinterface som finns med på databussen, men eftersom PLLen bara kör i standardläge används inte detta av mjukvaran.

3 Problem vid konstruktionen

3.1 Övergripande

Största problemet är att projektet vi valt omfattar mer än 1.5p, och att vi i början av läsperioden lade ner lite för lite tid. Vi har även varit dåliga på att läsa dokumentationen noga vilket har medfört onödiga fel och och därmed extra tidsåtgång.

3.2 ADC-specifika problem

Till en början problem med att timing krav inte möttes. Genom att byta ut den egna skrivna klockdelare mot en genererad pll och genom att skriva om alla moduler/processer till synkrona så löstes detta. Det var problem med i2c-modulen i början vid läsning av ack:ar. Det framgick ingenstans att i2c-datasignalen hade pull-up, vilket den hade. I övrigt inga stora problem.

3.3 VGA-specifika problem

Den första idén gick ut på att lägga en framebuffer i FPGAn för att få ett tillräckligt snabbt minne. Det visade sig att minnet skulle bli så stort att det inte fick plats i den aktuella FPGAn och därför fick DRAMet användas istället. Ett problem som inte hann lösas var att det uppstår ett flimmer på de översta pixelraderna och längst till vänster i bilden. Problemet beror inte på felaktiga synksignaler då dessa är verifierade med oscilloskop, utan beror troligtvis på adressmissar vid minnesläsningen från framebuffern.

4 Resultat

Ett digital oscilloskop som kan sampla en kanal samt presentera vissa data för signalen på en VGA-skärm. Triggnivå (begränsad upplösning), triggflank, trigg aktiverad och antal sampels kan ställas in via switchar. Vi hann inte med: menysystem, stig- och falltider, amplitudsupplösning. Det är även problem med adresseringen för VGA:n så hela skärmen kan inte användas. TODO FIX!

A Manual

Knappnummer	Funktion
0	reset

Switchnummer	Funktion
17 - 9	Triggnivåns översta 9 bitar av 16. Restrerande låga 7 bitar är alltid 0.
8	Trigg enable. Om aktiverad så startas samplingen på en trigg, annars samplas koninuerligt antal sampels.
7	Trigga på positiv flank. Annars trigga på fallande flank.
6 - 0	Hur många 1024 sampels som ska samplas. Måste vara större än 0.

Kanal 0, det vill säga vänster kanal används alltid som triggerkälla.

B Kod

B.1 Toppnivå

B.1.1 Mappning - oscil

```
-- VGA controller
-- generates vga signals from framebuffer
-- $Id: vga_controller.vhd 32 2006-11-17 12:01:07Z palmer $

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
--use ieee.numeric_std.all;
use ieee.std_logic_arith.all;

entity osci is
        port (
               -- 1) global signals:
                  CLOCK_50 : IN STD_LOGIC;
                  DRAM_CLK : OUT STD_LOGIC;
                                 SW     : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
                                 KEY : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                          LEDR : OUT STD_LOGIC_VECTOR(17 downto 0);
                          LEDG : OUT STD_LOGIC_VECTOR(8 downto 0);

               -- the_sdram_0
                  signal DRAM_ADDR : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
                                 signal DRAM_BA_0 : OUT STD_LOGIC;
                                 signal DRAM_BA_1 : OUT STD_LOGIC;
                  signal DRAM_CAS_N : OUT STD_LOGIC;
                  signal DRAM_CKE : OUT STD_LOGIC;
                  signal DRAM_CS_N : OUT STD_LOGIC;
                  signal DRAM_DQ : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
                  signal DRAM_UDQM : OUT STD_LOGIC;
                  signal DRAM_LDQM : OUT STD_LOGIC;
                  signal DRAM_RAS_N : OUT STD_LOGIC;
                  signal DRAM_WE_N : OUT STD_LOGIC;

               -- the_vga_controller
                                 signal VGA_R : out std_logic_vector(9 downto 0);
                                 signal VGA_G : out std_logic_vector(9 downto 0);
                                 signal VGA_B : out std_logic_vector(9 downto 0);
                                 signal VGA_BLANK : out std_logic;
                                 signal VGA_SYNC : out std_logic;
                                 signal VGA_CLK : out std_logic;
                                 signal VGA_HS : out std_logic;
                                 signal VGA_VS : out std_logic;

                          -- adc
                                 signal AUD_XCK         :       out std_logic;
                                 signal AUD_ADCLRCK     :       in std_logic;
                                 signal AUD_BCLK        :       in std_logic;
                                 signal AUD_ADCDAT      :       in std_logic;
                                 signal I2C_SDAT        :       inout std_logic;
                                 signal I2C_SCLK        :       out std_logic;

                          -- SRAM
                                 signal SRAM_DQ         :       inout std_logic_vector(15 downto 0);
                                 signal SRAM_ADDR       :       out std_logic_vector(17 downto 0);
                                 signal SRAM_OE_N       :       out std_logic;
                                 signal SRAM_WE_N       :       out std_logic;
                                 signal SRAM_CE_N   :   out std_logic
               );
end entity osci;

architecture oasci_arch of osci is
-- for vga
signal vga_addr : std_logic_vector(18 downto 0);
signal vga_data : std_logic_vector(1 downto 0);
-- communication between cpu and adc
signal  done_cpu        : std_logic;
signal  done_sample     : std_logic;
-- signals for sharing SRAM
signal  cpu_req_n       : std_logic;
signal  cpu_addr        : std_logic_vector(17 downto 0);
signal  cpu_dq  : std_logic_vector(15 downto 0);
signal  cpu_oe_n        : std_logic;
signal  adc_req_n       : std_logic;
signal  adc_addr        : std_logic_vector(17 downto 0);
signal  adc_dq  : std_logic_vector(15 downto 0);
signal  adc_we_n        : std_logic;


component avalon_cpu
                                Port (
               -- 1) global signals:
                  signal clk : IN STD_LOGIC;
                  signal clk_1 : OUT STD_LOGIC;
                  signal clk_adc : OUT STD_LOGIC;
                  signal reset_n : IN STD_LOGIC;

               -- the_cpu_adc_reg_0
                  signal done_cpu_from_the_cpu_adc_reg_0 : OUT STD_LOGIC;
                  signal done_sample_to_the_cpu_adc_reg_0 : IN STD_LOGIC;
```

```
            -- the_cpu_ui_interface_0
               signal key_to_the_cpu_ui_interface_0 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
               signal ledg_from_the_cpu_ui_interface_0 : OUT STD_LOGIC_VECTOR (8 DOWNTO 0);
               signal ledr_from_the_cpu_ui_interface_0 : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
               signal sw_to_the_cpu_ui_interface_0 : IN STD_LOGIC_VECTOR (17 DOWNTO 0);

            -- the_meminterface_cpu_0
               signal mem_addr_from_the_meminterface_cpu_0 : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
               signal mem_data_to_the_meminterface_cpu_0 : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
               signal mem_oe_from_the_meminterface_cpu_0 : OUT STD_LOGIC;
               signal req_n_from_the_meminterface_cpu_0 : OUT STD_LOGIC;

            -- the_sdram_0
               signal zs_addr_from_the_sdram_0 : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
               signal zs_ba_from_the_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
               signal zs_cas_n_from_the_sdram_0 : OUT STD_LOGIC;
               signal zs_cke_from_the_sdram_0 : OUT STD_LOGIC;
               signal zs_cs_n_from_the_sdram_0 : OUT STD_LOGIC;
               signal zs_dq_to_and_from_the_sdram_0 : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
               signal zs_dqm_from_the_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
               signal zs_ras_n_from_the_sdram_0 : OUT STD_LOGIC;
               signal zs_we_n_from_the_sdram_0 : OUT STD_LOGIC;

            -- the_vga_memory_interface_0
               signal ivga_fbaddr_to_the_vga_memory_interface_0 : IN STD_LOGIC_VECTOR (18 DOWNTO 0);
               signal ovgadata_from_the_vga_memory_interface_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
            );
end component;

component vga_controller
  Port (
    VGA_DATA    : in std_logic_vector(1 downto 0);
    FB_ADDR     : out std_logic_vector(18 downto 0);
    VGA_R       : out std_logic_vector(9 downto 0);
    VGA_G       : out std_logic_vector(9 downto 0);
    VGA_B       : out std_logic_vector(9 downto 0);
    VGA_BLANK   : out std_logic;
    VGA_SYNC    : out std_logic;
    VGA_CLOCK   : out std_logic;
    VGA_HS      : out std_logic;
    VGA_VS      : out std_logic;
    clk         : in std_logic;
    rst         : in std_logic
    );
end component;

begin
 cpu : avalon_cpu port map(
                clk => CLOCK_50,
                clk_1 => DRAM_CLK,
                clk_adc => AUD_XCK,
                reset_n => KEY(0),

                done_cpu_from_the_cpu_adc_reg_0 => done_cpu,
                done_sample_to_the_cpu_adc_reg_0 => done_sample,

                KEY_to_the_cpu_ui_interface_0 => KEY,
                LEDG_from_the_cpu_ui_interface_0 => LEDG,
                LEDR_from_the_cpu_ui_interface_0 => LEDR,
                SW_to_the_cpu_ui_interface_0 => SW,

                mem_addr_from_the_meminterface_cpu_0 => cpu_addr,
                mem_data_to_the_meminterface_cpu_0 => cpu_dq,
                mem_oe_from_the_meminterface_cpu_0 => cpu_oe_n,
                            req_n_from_the_meminterface_cpu_0 => cpu_req_n,

                zs_addr_from_the_sdram_0 => DRAM_ADDR,
                zs_ba_from_the_sdram_0(1) => DRAM_BA_1,
                zs_ba_from_the_sdram_0(0) => DRAM_BA_0,
                zs_cas_n_from_the_sdram_0 => DRAM_CAS_N,
                zs_cke_from_the_sdram_0 => DRAM_CKE,
                zs_cs_n_from_the_sdram_0 => DRAM_CS_N,
                zs_dq_to_and_from_the_sdram_0 => DRAM_DQ,
                zs_dqm_from_the_sdram_0(1) => DRAM_UDQM,
                zs_dqm_from_the_sdram_0(0) => DRAM_LDQM,
                zs_ras_n_from_the_sdram_0 => DRAM_RAS_N,
                zs_we_n_from_the_sdram_0 => DRAM_WE_N,

                ivga_fbaddr_to_the_vga_memory_interface_0 => vga_addr,
                ovgadata_from_the_vga_memory_interface_0 => vga_data
                       );

 vga : vga_controller Port map (
    VGA_DATA => vga_data,
    FB_ADDR => vga_addr,
    VGA_R => VGA_R,
    VGA_G => VGA_G,
    VGA_B => VGA_B,
    VGA_BLANK => VGA_BLANK,
    VGA_SYNC => VGA_SYNC,
    VGA_CLOCK => VGA_CLK,
    VGA_HS => VGA_HS,
    VGA_VS => VGA_VS,
    clk => CLOCK_50,
    rst => KEY(0)
    );

        toplevel : entity work.toplevel(arch)
```

```
                port map (
                        CLOCK_50,
                        AUD_ADCLRCK,
                        AUD_BCLK,
                        AUD_ADCDAT,
                        I2C_SDAT,
                        I2C_SCLK,
                        SW,
                        KEY,
                        adc_dq,
                        adc_addr,
                        adc_we_n,
                        done_cpu,
                        done_sample,
                        --ack0,
                        adc_req_n
                        );


memctrl : entity work.memctrl(arch)
                port map (
                 CLOCK_50,
                 SRAM_DQ,
                 SRAM_ADDR,
                 SRAM_OE_N,
                 SRAM_WE_N,
                 SRAM_CE_N,
                 -- From cpu (only read)
                 cpu_req_n,
                 cpu_addr,
                 cpu_dq,
                 cpu_oe_n,
                 -- From adc (only write)
                 adc_req_n,
                 adc_addr,
                 adc_dq,
                 adc_we_n
         );

end;
```

B.1.2 Minneskontroller - memctrl

```
-----------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/cpu/avalon_cpu/memctrl.vhd $
-- Last changed:        $LastChangedDate: 2006-12-12 16:42:09 +0100 (ti, 12 dec 2006) $
-- By:                          $LastChangedBy: majohan $
-- Revision:            $Revision: 177 $
-----------------------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;


entity memctrl is
        port(pgm_clk : in std_logic;
                 -- To SRAM
                 SRAM_DQ        :       inout std_logic_vector(15 downto 0);
                 SRAM_ADDR      :       out std_logic_vector(17 downto 0);
                 SRAM_OE_N      :       out std_logic;
                 SRAM_WE_N      :       out std_logic;
                 SRAM_CE_N  :   out std_logic;
                 -- From cpu (only read)
                 cpu_req_n      : in std_logic;
                 cpu_addr       : in std_logic_vector(17 downto 0);
                 cpu_dq         : out std_logic_vector(15 downto 0);
                 cpu_oe_n       : in std_logic;
                 -- From adc (only write)
                 adc_req_n      : in std_logic;
                 adc_addr       :       in   std_logic_vector(17 downto 0);
                 adc_dq         :       in std_logic_vector(15 downto 0);
                 adc_we_n       :       in   std_logic
         );
end memctrl;

architecture arch of memctrl is

begin

process (pgm_clk)
begin

        if rising_edge(pgm_clk) then

                -- The ADC has highest priority
                -- let it use the SRAM if it needs to
                if adc_req_n = '0' then
                        SRAM_DQ <= adc_dq;
                        SRAM_ADDR <= adc_addr;
                        SRAM_OE_N <= '1'; -- Only write
                        SRAM_WE_N <= adc_we_n;
                        SRAM_CE_N <='0';

                -- The CPU has lower priority
                -- let it use the SRAM if it needs to (and the ADC does not)
                elsif cpu_req_n = '0' then
                        cpu_dq <= SRAM_DQ;
                        SRAM_ADDR <= cpu_addr;
```

```
                        SRAM_OE_N <= cpu_oe_n;
                        SRAM_WE_N <= '1'; -- Only read
                        SRAM_CE_N <= '0';

                -- If no module needs the use the SRAM, put high impedence on databuss and disable everything
                else
                        SRAM_DQ <= (others => 'Z');
                        SRAM_CE_N <= '1';
                        SRAM_OE_N <= '1';
                        SRAM_WE_N <= '1';
                end if;

        end if;

end process;


end architecture arch;
```

B.2 ADC

B.2.1 Mappning - toplevel

```
library ieee;
use ieee.STD_LOGIC_1164.all;

-- interface towards osci.vhd
-- this was during testing of the adc-block the toplevel mapped directly to
-- pins on the FPGA, thereby the names

entity toplevel is
        port(
                 CLOCK_50               :       in    std_logic;
                 AUD_ADCLRCK    :       in    std_logic;
                 AUD_BCLK               :       in    std_logic;
                 AUD_ADCDAT             :       in    std_logic;
                 I2C_SDAT               :       inout std_logic;
                 I2C_SCLK               :       out   std_logic;
                 SW                             :       in    std_logic_vector(17 downto 0);
                 KEY                    :       in    std_logic_vector(3 downto 0);
                 SRAM_DQ                :       out   std_logic_vector(15 downto 0);
                 SRAM_ADDR              :       out   std_logic_vector(17 downto 0);
                 SRAM_WE_N              :       out   std_logic;
                 done_cpu               :       in    std_logic;
                 done_sample    :       out   std_logic;
                 req_mem                :       out   std_logic
                );

end entity toplevel;

architecture arch of toplevel is

-- Here the components in the ADC-block are connected to each other

signal  sample_done     : std_logic;
signal  cnt_reset       : std_logic;
signal  mem_done        : std_logic;
signal  mem_write       : std_logic;
signal  chan0           : std_logic_vector(15 downto 0);
signal  chan1           : std_logic_vector(15 downto 0);
signal  i2c_done        : std_logic;
signal  i2c_reg         : std_logic_vector(6 downto 0);
signal  i2c_data        : std_logic_vector(8 downto 0);
signal  i2c_go          : std_logic;

begin

        memory : entity work.memory(arch)
                port map (
                        CLOCK_50,
                        KEY(0),
                        cnt_reset,
                        mem_write,
                        chan0,
                        chan1,
                        SRAM_WE_N,
                        req_mem,
                        SRAM_ADDR,
                        SRAM_DQ,
                        mem_done
                        );

        sample : entity work.sample(arch)
                port map (
                        CLOCK_50,
                        KEY(0),
                        AUD_ADCLRCK,
                        AUD_BCLK,
                        AUD_ADCDAT,
                        chan0,
                        chan1,
                        sample_done
                        );

        interface       : entity work.interface(arch)
                port map (
                        pgm_clk => CLOCK_50,
```

```
                        reset => KEY(0),
                        num_samples(6 downto 0) => SW(6 downto 0),
                        trig_level(15 downto 7) => SW(17 downto 9),
                        trig_level(6 downto 0)=>(others => '0') ,
                        trig_pos => SW(7),trig_ext => '0',
                        trig_en => SW(8),trig_src => "01",
                        chan0 => chan0,
                        chan1 => chan1,
                        sample_done => sample_done,
                        write_done => mem_done,
                        cpu_done_cpu => done_cpu,
                        cpu_done_sample => done_sample,
                        cnt_reset => cnt_reset,
                        write => mem_write
                        );


        init    : entity work.init(arch)
                port map (
                        CLOCK_50,
                        KEY(0),
                        i2c_done,
                        i2c_reg,
                        i2c_data,
                        i2c_go
                        );

        i2c     : entity work.i2c(arch)
                port map (
                        CLOCK_50,
                        i2c_reg,
                        i2c_data,
                        i2c_go,
                        KEY(0),
                        i2c_done,
                        I2C_SDAT,
                        I2C_SCLK
                        );

end;
```

B.2.2 Initieringsmodul - Init

```
-------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/adc/source/init.vhd $
-- Last changed:        $LastChangedDate: 2006-12-12 16:42:09 +0100 (ti, 12 dec 2006) $
-- By:                          $LastChangedBy: majohan $
-- Revision:            $Revision: 177 $
-------------------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;


entity init is
        port(pgm_clk : in std_logic;
           reset                : in std_logic;
                 done           : in std_logic;
                 reg    : out std_logic_vector(6 downto 0);
                 data   : out std_logic_vector(8 downto 0);
                 go     : out std_logic);
end init;

---------------A R C H I T E C T U R E---------------

architecture arch of init is



begin

process (pgm_clk,reset)

variable state : std_logic_vector(9 downto 0);
variable done_prev : std_logic;

constant S0 : std_logic_vector(9 downto 0):="0000000001";
constant S1 : std_logic_vector(9 downto 0):="0000000010";
constant S2 : std_logic_vector(9 downto 0):="0000000100";
constant S3 : std_logic_vector(9 downto 0):="0000001000";
constant S4 : std_logic_vector(9 downto 0):="0000010000";
constant S5 : std_logic_vector(9 downto 0):="0000100000";
constant S6 : std_logic_vector(9 downto 0):="0001000000";
constant S7 : std_logic_vector(9 downto 0):="0010000000";
constant S8 : std_logic_vector(9 downto 0):="0100000000";
constant S9 : std_logic_vector(9 downto 0):="1000000000";


begin

-- Reset, async
if (reset='0') then
        state:=S0;
        reg<="0000000";
        data<="000000000";
        go<='0';
```

```
        done_prev:='0';
elsif rising_edge(pgm_clk) then


        -- If i2c-module is ready to write, write data to the next register (if not finished writing to all needed configuration registers)
        -- registers and data according to WM8731_WM8731L.pdf datasheet.
        if (done='1' and done_prev='0') then
                case state is

                        when S0 =>
                                reg<="0001111"; -- Reset
                                data<="000000000"; -- issue reset
                                go<='1';
                                state(0):='0';
                                state(1):='1'; -- S1

                        when S1 =>
                                reg<="0000000"; -- Left line In
                                data<="000010111"; -- Disable sim. load, disable mute, 0db volume.
                                go<='1';
                                state(1):='0';
                                state(2):='1'; -- S2

                        when S2 =>
                                reg<="0000001"; -- Right line In
                                data<="000010111"; -- Disable sim. load, disable mute, 0db volume.
                                go<='1';
                                state(2):='0';
                                state(3):='1'; -- S3

                        when S3 =>
                                reg<="0000100"; -- Analogue Audio Path Control
                                data<="000000000"; --No sidetone, don't use DAC, don't bypass, use line input
                                go<='1';
                                state(3):='0';
                                state(4):='1'; -- S4

                        when S4 =>
                                reg<="0000101"; -- Digital Audio path Control
                                data<="000000001"; -- no soft mute, no de-emphasis, no ADC high pass filter
                                go<='1';
                                state(4):='0';
                                state(5):='1'; -- S5

                        when S5 =>
                                reg<="0000110"; -- Power Down Control
                                data<="000000000"; -- No power down
                                go<='1';
                                state(5):='0';
                                state(6):='1'; -- S6

                        when S6 =>
                                reg<="0000111"; -- Digital Audio Interace Format
                                data<="001000011"; -- don't invert BITCLK, master mode, right channel data right,
                                                        -- MSB on 1st BCLK, 16 bits sample length, DSP mode.
                                go<='1';
                                state(6):='0';
                                state(7):='1'; -- S7

                        when S7 =>
                                reg<="0001000"; -- Sampling control
                                data<="000000000"; -- core clock is MCLK, 48kHz( double clk in gives 96kHz), normal mode.
                                go<='1';
                                state(7):='0';
                                state(8):='1'; -- S8

                        when S8 =>
                                reg<="0001001"; -- Activate Control
                                data<="000000001"; -- Activate.
                                go<='1';
                                state(8):='0';
                                state(9):='1'; -- S9

                        -- we will automaticly come here after last statement above.
                        when others => -- S9
                            go<='0';

                end case; -- End state swtich

        end if; -- End done =1 and done_prev =0

   done_prev:=done;

end if; -- End rising edge

end process;

end architecture arch;
```

B.2.3 I^{2}C-modul - i2c

```
-----------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/adc/source/i2c.vhd $
-- Last changed:        $LastChangedDate: 2006-12-12 16:42:09 +0100 (ti, 12 dec 2006) $
-- By:                          $LastChangedBy: majohan $
-- Revision:            $Revision: 177 $
-----------------------------------------------------------
```

```
library ieee;
use ieee.STD_LOGIC_1164.all;

entity i2c is
        port(pgm_clk    : in std_logic;
                reg             : in std_logic_vector(6 downto 0);
                data    : in std_logic_vector(8 downto 0);
                go              : in std_logic;
            reset               : in std_logic;
                done            : out std_logic;
                SDO             : inout std_logic;
                i2c_clk         : buffer std_logic);
end i2c;

---------------A R C H I T E C T U R E---------------

architecture arch of i2c is

        -- used in combination with the three-state process to convert signal from 1 to high impedance before
        -- outputting the signal
        signal i2c_data : std_logic;

begin


process (pgm_clk,reset)

        constant WRITE_ADDR : std_logic_vector(7 downto 0):="00110100"; -- I2c write addr.
        variable going : bit; -- Indicate if we are on the go.
        variable bitcnt : integer; --Bit index counter. (not exactly)
        variable clkcnt : integer; -- Counter to create i2c counter.
   variable switch : bit; -- To switch bitcnt on every second

begin

-- Reset
if (reset='0') then
        done<='1';
        going:='0';
        clkcnt:=0;
else

        if rising_edge(pgm_clk) then

                -- Set initial values on signals and variables if currently not transfering and are told to do so.
                if (go='1' AND going='0') then
                        going:='1';
                        bitcnt:=0;
                        done<='0';
                        clkcnt:=0;
                        i2c_data<='1'; -- Must start high!
                        i2c_clk<='1'; -- Must start high!
                        switch:='1'; -- Must start high!
                end if;

                -- If its time to toggle clock.
                if (going='1' and clkcnt>5) then
                        -- Increment bitcnt when clk_out is zero
                        -- This ensures that data on the i2c_data has been vaild
                        -- during a low to high transition (because it has gone from high to low == one period).
                        if (i2c_clk='1') then
                                i2c_clk<='0';
                        else
                                i2c_clk<='1';
                        end if;

                        -- Bit counter increment on every second clock
                        if switch='1' then
                            bitcnt:=bitcnt+1;
                        end if;
                        switch:=not switch;

                        clkcnt:=0;
                end if;

                -- If currently transfering
                if (going='1') then

                        -- Se page 43 in WM8731_WM8731L.pdf datasheet.

                        if (bitcnt=0 and clkcnt>2) then -- Startbit
                                i2c_data<='0';
                                bitcnt:=bitcnt+1;

                        elsif (bitcnt=1) then -- Startbit

                        elsif (bitcnt>=2 and bitcnt<=9) then -- Write "Write addr"
                                -- bitcnt starts at 2
                                i2c_data<=WRITE_ADDR(9-bitcnt);

                        elsif (bitcnt=10) then -- write ack
                                i2c_data<='0';

                        elsif (bitcnt>=11 and bitcnt<=17) then -- Write register addres
                                -- bitcnt starts at 11
                                i2c_data<=reg(17-bitcnt);

                        elsif (bitcnt=18) then -- write MSB of data
                                i2c_data<=data(26-bitcnt);
```

```
                        elsif (bitcnt=19) then -- write ack
                                i2c_data<='0';

                        elsif (bitcnt>=20 and bitcnt<=27) then -- Write rest of data
                                -- bitcnt starts at 20
                                i2c_data<=data(27-bitcnt);

                        elsif (bitcnt=28) then -- write ack
                                i2c_data<='0';

                        elsif (bitcnt=29) then -- Stopbit
                                i2c_data<='0';
                                switch:='1';

                        elsif (bitcnt=30 AND clkcnt>2) then -- Stopbit
                                i2c_data<='1';

                        elsif (bitcnt=31) then
                                going:='0';
                                done<='1';

                        end if;

                end if; -- End going

                -- increment i2c clock div.
                clkcnt:=clkcnt+1;

        end if; -- End rising edge

end if; -- !End reset

end process;

-- To convert 1's to high impedance
tristate : process (i2c_data)
        begin
                if      i2c_data='1' then
                        SDO<='Z';
                else
                        SDO<='0';
                end if;
end process;

end architecture arch;
```

B.2.4 Sampelmodul - sample

```
-----------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/adc/source/sample.vhd $
-- Last changed:        $LastChangedDate: 2006-12-12 16:42:09 +0100 (ti, 12 dec 2006) $
-- By:                          $LastChangedBy: majohan $
-- Revision:            $Revision: 177 $
-----------------------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;

entity sample is
        port(pgm_clk    : in std_logic;
                 reset          : in std_logic;
                 adc_lrc        : in std_logic;
                 adc_bclk       : in std_logic;
                 adc_dat        : in std_logic;
                 chan0          : out std_logic_vector(15 downto 0);
                 chan1          : out std_logic_vector(15 downto 0);
                 done           : out std_logic);
end sample;

---------------A R C H I T E C T U R E---------------

architecture arch of sample is

begin

        process (pgm_clk,reset)

        variable cnt : integer; -- Bit counter
        variable chan : std_logic_vector(31 downto 0);
        variable adc_lrc_prev : std_logic; -- Previous value for adc_lrc
        variable adc_bclk_prev : std_logic; -- Previous value for adc_lrc
        variable started : std_logic; -- if currently reading bits from adc_dat

        begin

                -- Reset
                if (reset='0') then
                        -- Clear previous values and flag not done.
                        adc_lrc_prev:='0';
                        adc_bclk_prev:='0';
                        done<='0';
                        cnt:=32;
                        started:='0';
                elsif rising_edge(pgm_clk) then


                        -- On falling edge of adc_lrc (that is, we have new sample),
```

```
                        -- clear bit counter, flag undone and start reading bits.
                        if (adc_lrc='0' and adc_lrc_prev='1') then
                                cnt:=0;
                                done<='0';
                                started:='1';
                        end if;


                        -- if started, read a bit and save to internal vector
                        if (adc_bclk_prev='0' and adc_bclk='1' and cnt<32 and started='1') then
                                chan(31-cnt):=adc_dat;
                                cnt:=cnt+1;
                        end if;

                        -- Output sample and flag done
                        if (cnt>31 and started='1') then
                                chan0<=chan(31 downto 16);
                                chan1<=chan(15 downto 0);
                                done<='1';
                                cnt:=0;
                                started:='0';
                        end if;

                        -- Save previous values.
                        adc_lrc_prev:=adc_lrc;
                        adc_bclk_prev:=adc_bclk;

                end if; -- End rising edge

        end process;

end architecture arch;
```

B.2.5 Interfacemodul - interface

```
-----------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/adc/source/interface.vhd $
-- Last changed:        $LastChangedDate: 2006-12-12 16:42:09 +0100 (ti, 12 dec 2006) $
-- By:                          $LastChangedBy: majohan $
-- Revision:            $Revision: 177 $
-----------------------------------------------------------
library ieee;
use ieee.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;


entity interface is
        port(   -- general signals
                pgm_clk                 : in std_logic;
                reset                   : in std_logic;
                -- configuration parameters
                num_samples     : in std_logic_vector(6 downto 0);
                trig_level              : in std_logic_vector(15 downto 0);
                trig_pos                : in std_logic;
                trig_ext                : in std_logic;
                trig_en                 : in std_logic;
                trig_src                : in std_logic_vector(1 downto 0);
                -- sample data
                chan0                   : in std_logic_vector(15 downto 0);
                chan1                   : in std_logic_vector(15 downto 0);
                -- various signals to communicate with other modules
                sample_done     : in std_logic;
                write_done              : in std_logic;
                cpu_done_cpu    : in std_logic;
                cpu_done_sample: out std_logic;
                cnt_reset               : out std_logic;
                write                   : out std_logic);
end interface;


---------------A R C H I T E C T U R E---------------

architecture arch of interface is

begin

process (pgm_clk,reset)

        variable sample                         : signed(15 downto 0); -- sample to check against trig
        variable trig_level_int                 : signed(15 downto 0); -- internal trig level
        variable sample_cnt                     : unsigned(16 downto 0); -- Sample counter
        variable is_sampling                    : std_logic;
        variable cpu_done_cpu_arrived           : std_logic;
                -- For previous values
        variable write_done_prev                : std_logic;
        variable sample_done_prev               : std_logic;
        variable cpu_done_cpu_prev              : std_logic;
        variable sample_prev                    : signed(15 downto 0);

begin

-- Reset
if (reset='0') then
        sample_prev:="0000000000000000";
        sample_cnt:="00000000000000000";
        write_done_prev:='0';
        sample_done_prev:='0';
        is_sampling:='0';
```

```
        cnt_reset<='1';
        write<='0';
elsif rising_edge(pgm_clk) then

        -- Normally do not write, write is set to '1' further down if writing
        write<='0';

        if (sample_done='1' and sample_done_prev='0' and cpu_done_cpu_arrived='1') then

                if is_sampling='0' then
                        -- Save sample and choose trig_level
                        case trig_src is
                                when "01" => -- Channel 1
                                        sample:=signed(chan0);
                                        trig_level_int:=signed(trig_level);
                                when "10" => -- Channel 2
                                        sample:=signed(chan1);
                                        trig_level_int:=signed(trig_level);
                                when others => -- External trig for 0 and 3
                                         -- To be able to use same algorithm for external trig
                                        sample(15 downto 1):="000000000000000";
                                        sample(0):=trig_ext;
                                        trig_level_int(15 downto 1):="000000000000000";
                                        trig_level_int(0):=trig_pos;
                        end case;

                        if trig_en='1' then

                                case trig_pos is
                                        when '1' =>             -- Trig on rising edge
                                                if (sample_prev<trig_level_int) and (sample>=trig_level_int) then
                                                        is_sampling:='1';
                                                end if;
                                        when others =>  -- Trig on falling edge
                                                if (sample_prev>trig_level_int) and (sample<=trig_level_int) then
                                                        is_sampling:='1';
                                                end if;
                                end case;

                        else -- if trig is not enabled, always sample
                                is_sampling:='1';
                        end if;

                end if; -- end is_sampling

                if (is_sampling = '1') then
                        write<='1';
                        sample_cnt:=sample_cnt+to_unsigned(1,17);

                        -- check if requested amount of samples has been reached
                        if std_logic_vector(sample_cnt(16 downto 10)) = num_samples  then
                                is_sampling:='0';
                                cpu_done_cpu_arrived:='0';
                                sample_cnt:="00000000000000000";
                        end if;

                end if; -- is_sampling='1'

           -- Save for next iteration
           sample_prev:=sample;

        end if; -- end sample_done

        -- Detect cpu_done_cpu
        if (cpu_done_cpu='1' and cpu_done_cpu_prev='0') then
                cpu_done_cpu_arrived:='1';
        end if;

-- Save for next iteration
sample_done_prev:=sample_done;
write_done_prev:=write_done;
cpu_done_cpu_prev:=cpu_done_cpu;

cnt_reset<=not is_sampling;
cpu_done_sample<=not is_sampling;


end if; -- End if not Reset

end process;

end architecture arch;
```

B.2.6 Minnesmodul - Memory

```
-------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/adc/source/memory.vhd $
-- Last changed:        $LastChangedDate: 2006-12-12 16:42:09 +0100 (ti, 12 dec 2006) $
-- By:                          $LastChangedBy: majohan $
-- Revision:            $Revision: 177 $
-------------------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;


entity memory is
```

```
        port(pgm_clk    : in std_logic;
                reset         : in std_logic;
             cnt_reset  : in std_logic;
                write         : in std_logic;
                chan0         : in std_logic_vector(15 downto 0);
                chan1         : in std_logic_vector(15 downto 0);
                mem_we        : out std_logic;
                req_n         : out std_logic;
                mem_addr      : out std_logic_vector(17 downto 0);
                mem_data      : out std_logic_vector(15 downto 0);
                done          : out std_logic);
end memory;


---------------A R C H I T E C T U R E---------------

architecture arch of memory is

begin

process (pgm_clk, reset)

        variable addr_cnt : unsigned(17 downto 0); -- Address pointer in memory
        variable start_write : std_logic; -- If to start write or not.
        variable write_state : std_logic_vector(7 downto 0);  -- In which write state we are

        constant S0 : std_logic_vector(7 downto 0):="00000001";
        constant S1 : std_logic_vector(7 downto 0):="00000010";
        constant S2 : std_logic_vector(7 downto 0):="00000100";
        constant S3 : std_logic_vector(7 downto 0):="00001000";
        constant S4 : std_logic_vector(7 downto 0):="00010000";
        constant S5 : std_logic_vector(7 downto 0):="00100000";
        constant S6 : std_logic_vector(7 downto 0):="01000000";
        constant S7 : std_logic_vector(7 downto 0):="10000000";

begin

        -- Reset
if (reset='0') then -- Flag undone, set start state
        addr_cnt:=to_unsigned(0,18);
        write_state:=S0; -- Start state
        start_write:='0';
        mem_we<='1';
        req_n<='1';
        done<='0';
elsif rising_edge(pgm_clk) then

        -- interface must init this device by issue cnt_reset='1' to
        -- Reset addresspointer.
        if (cnt_reset='1') then
                addr_cnt:=to_unsigned(0,18);
                write_state:=S0;
                start_write:='0';
                mem_we<='1';
                req_n<='1';
        end if;


        -- If interface module issued write
        if (write='1' AND start_write='0') then
           done<='0';
                start_write:='1'; -- Start write.
                write_state:=S0;
        end if;

        -- If Start write and we have acknowlegde from memoryctrl to write.
        if (start_write='1') then

                mem_addr<=std_logic_vector(addr_cnt); -- Address in address counter


                case write_state is
                        when S0 =>
                                req_n<='0'; -- request memory
                                mem_data<=chan0; -- Write channel0
                                write_state(0):='0';
                                write_state(1):='1'; -- S1
                        when S1 =>
                                mem_we<='0'; -- Bring write high
                                write_state(1):='0';
                                write_state(2):='1'; -- S2
                        when S2 =>
                                mem_we<='1'; -- Bring write low (Now chan0 have been written)
                                addr_cnt:=addr_cnt+1; -- Increment address
                                write_state(2):='0';
                                write_state(3):='1'; -- S3
                        when S3 =>
                                mem_data<=chan1; -- Write channel1
                                write_state(3):='0';
                                write_state(4):='1'; -- S4
                        when S4 =>
                                mem_we<='0'; -- Bring write high
                                write_state(4):='0';
                                write_state(5):='1'; -- S5
                        when S5 =>
                                mem_we<='1'; -- Bring write low (Now chan1 have been written)
                                write_state(5):='0';
                                write_state(6):='1'; -- S6
                        when S6 =>
```

```
                                write_state(6):='0';
                                write_state(7):='1'; -- S7
                                done<='1';
                                addr_cnt:=addr_cnt+1; -- Increment address
                        when others => -- S7
                                req_n<='1'; -- memory is not need anymore
                                done<='1';
                                start_write:='0';
                end case;


        end if; -- start_write and ack

end if; -- rising edge

end process;

end architecture arch;
```

B.3 VGA

B.3.1 VGA-drivenhet - vga_controller

```
-- VGA controller
-- generates vga signals from framebuffer
-- $Id: vga_controller.vhd 139 2006-12-10 11:40:25Z jonatan $

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
--use ieee.numeric_std.all;
use ieee.std_logic_arith.all;

entity vga_controller is
  Port (
    VGA_DATA    : in std_logic_vector(1 downto 0);
    FB_ADDR     : out std_logic_vector(18 downto 0);
    VGA_R       : out std_logic_vector(9 downto 0);
    VGA_G       : out std_logic_vector(9 downto 0);
    VGA_B       : out std_logic_vector(9 downto 0);
    VGA_BLANK   : out std_logic;
    VGA_SYNC    : out std_logic;
    VGA_CLOCK   : out std_logic;
    VGA_HS      : out std_logic;
    VGA_VS      : out std_logic;
    clk         : in std_logic;
    rst         : in std_logic
    );
end vga_controller;

architecture vga_controller_arch of vga_controller is
  constant HSYNC_TOT : integer range 0 to 2047 := 1040; -- total horizontal
                                                        -- count (pixels)
  constant H_BACK_PORCH : integer range 0 to 127 := 64;  -- horiz front porch
  constant H_SYNC_PULSE : integer range 0 to 255 := 120;  -- horiz synch pulse width
  constant H_ACTIVE : integer range 0 to 1023 := 800;  -- horiz active region
  constant H_FRONT_PORCH : integer range 0 to 63 := 56;  -- horiz back porch

  constant VSYNC_TOT : integer range 0 to 1023 := 666;  -- total vertical count
                                                        -- (lines)
  constant V_BACK_PORCH : integer range 0 to 31 := 23;  -- vert front porch
  constant V_SYNC_PULSE : integer range 0 to 7 := 6;  -- vert sync pulse width
  constant V_ACTIVE : integer range 0 to 1023 := 600;  -- vert active region
  constant V_FRONT_PORCH : integer range 0 to 63 := 37;  -- vert back porch

  signal vert_count, horiz_count : integer range 0 to 2047;  -- counters

  signal lR : std_logic_vector(9 downto 0);  -- local red
  signal lG : std_logic_vector(9 downto 0);  -- local green
  signal lB : std_logic_vector(9 downto 0);  -- local blue

begin  -- vga_controller_arch
  -- assign local vars to DAC RGB input
  VGA_R <= lR;
  VGA_G <= lG;
  VGA_B <= lB;
  VGA_SYNC <= '1';
  VGA_BLANK <= '1';
  VGA_CLOCK <= clk;
  process(clk)
    begin
      if rising_edge(clk) then
         if rst = '0' then
           vert_count <= 0;
           horiz_count <= 0;
           VGA_HS <= '1';
           VGA_VS <= '1';
         else
           if horiz_count > HSYNC_TOT-1 then --clear horizontal counter
             horiz_count <= 0;

             if vert_count > VSYNC_TOT-1 then --clear vertical counter
               vert_count <= 0;
             else
               vert_count <= vert_count + 1;
               if vert_count < V_SYNC_PULSE then --generate vertical sync pulse
                                                 --vertical front porch
```

```
            VGA_VS <= '0';
            FB_ADDR <= (others => '0');
          else
            VGA_VS <= '1';       --deassert vertical sync pulse
          end if;
        end if;

      else
        horiz_count <= horiz_count + 1;
      end if;

      if horiz_count < H_SYNC_PULSE then --horizontal front porch
        VGA_HS <= '0';                    --generate horizontal sync pulse
        lR <= (others => '0');
        lG <= (others => '0');
        lB <= (others => '0');
        FB_ADDR <= conv_std_logic_vector((vert_count - V_SYNC_PULSE - V_BACK_PORCH) * H_ACTIVE,19);
      elsif horiz_count < H_SYNC_PULSE + H_BACK_PORCH then --horizontal front porch
        VGA_HS <= '1';            --deassert horizontal sync pulse
        lR <= (others => '0');
        lG <= (others => '0');
        lB <= (others => '0');
      elsif (horiz_count < H_SYNC_PULSE + H_BACK_PORCH + H_ACTIVE) AND
        (vert_count > V_SYNC_PULSE + V_BACK_PORCH - 1) AND
        (vert_count < V_SYNC_PULSE + V_BACK_PORCH + V_ACTIVE) then
        -- active area

          --VGA_HS <= '1';

          -- fetch from memory
          FB_ADDR <= conv_std_logic_vector((vert_count - V_SYNC_PULSE - V_BACK_PORCH) * H_ACTIVE + (horiz_count - H_SYNC_PULSE - H_BACK_PORCH),19);
                        case VGA_DATA is
                            when "11" =>
                                    lR <= (others => '1'); --Set colour white
                                    lG <= (others => '1');
                                    lB <= (others => '1');
                                when "10" =>
                                    lR <= (others => '1'); --Set colour red
                                    lG <= (others => '0');
                                    lB <= (others => '0');
                                when "01" =>
                                    lR <= (others => '0'); --Set colour green
                                    lG <= (others => '1');
                                    lB <= (others => '0');
                                when "00" =>
                                    lR <= (others => '0'); --Set colour black
                                    lG <= (others => '0');
                                    lB <= (others => '0');
                                when others =>
                                    null;
                        end case;
      elsif (horiz_count > H_SYNC_PULSE + H_BACK_PORCH + H_ACTIVE) then
        -- horizontal back porch
        --VGA_HS <= '1';
        lR <= (others => '0');
        lG <= (others => '0');
        lB <= (others => '0');
        FB_ADDR <= conv_std_logic_vector((vert_count - V_SYNC_PULSE - V_BACK_PORCH + 1) * H_ACTIVE ,19);
      --elsif (vert_count > V_BACK_PORCH + V_SYNC_PULSE + V_ACTIVE) then
      else
        --vertical back porch
        FB_ADDR <= (others => '0');
        lR <= (others => '0');
        lG <= (others => '0');
        lB <= (others => '0');
      end if;
    end if;
  end if;
  end process;
end vga_controller_arch;
```

B.4 CPU

B.4.1 ADC control register interface - cpu_adc_reg

```
-----------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/adc/source/meminterface_cpu.vhd $
-- Last changed:        $LastChangedDate: 2006-12-02 12:20:27 +0100 (lö, 02 dec 2006) $
-- By:                          $LastChangedBy: johboh $
-- Revision:            $Revision: 90 $
-----------------------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;


entity cpu_adc_reg is
        port(-- interface towards cpu
                cpu_clk                 : in std_logic;
                cpu_addr                : in std_logic_vector(1 downto 0);  -- address presented by "cpu", not used
                cpu_data_in     : in std_logic_vector(31 downto 0);  -- data full 32 bits only
                cpu_data_out    : out std_logic_vector(31 downto 0);  -- data full 32 bits only
                cpu_we                  : in std_logic;                 -- slave port WE
                cpu_re                  : in std_logic;                         -- slave port RE
                cpu_cs                  : in std_logic;                 -- slave port CS
                -- interface towards interface.vhd
```

```
                done_sample    : in std_logic;
                done_cpu               : out std_logic);

end cpu_adc_reg;


architecture arch of cpu_adc_reg is

begin

process(cpu_clk)

begin
        if rising_edge(cpu_clk) then
                -- if cpu wants to read data then give it done_sample
                if cpu_re='1' and cpu_cs='1' then
                        cpu_data_out(31 downto 1)<=(others =>'0');
                        cpu_data_out(0)<=done_sample;
                -- if cpu wants to write then set done_cup
                elsif cpu_we='1' and cpu_cs='1' then
                        done_cpu<= cpu_data_in(0);
                end if;
        end if; -- rising_edge(cpu_clk)
end process;

end architecture arch;
```

B.4.2 ADC user interface interface - cpu_ui_interface

```
-------------------------------------------------------
-- FileName:            $HeadURL$
-- Last changed:        $LastChangedDate$
-- By:                          $LastChangedBy$
-- Revision:            $Revision$
-------------------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;


entity cpu_ui_interface is
        port(-- interface towards cpu
                cpu_clk                 : in std_logic;
                cpu_addr                : in std_logic_vector(1 downto 0);  -- address presented by "cpu"
                cpu_data_in     : in std_logic_vector(31 downto 0);  -- data full 32 bits only
                cpu_data_out    : out std_logic_vector(31 downto 0);  -- data full 32 bits only
                cpu_we                  : in std_logic;                 -- slave port WE
                cpu_re                  : in std_logic;                         -- slave port RE
                cpu_cs                  : in std_logic;                 -- slave port CS
                -- interface towards hardware switches and leds
                LEDR                    : out std_logic_vector(17 downto 0);
                LEDG                    : out std_logic_vector(8 downto 0);
                SW                              : in std_logic_vector(17 downto 0);
                KEY                     : in std_logic_vector(3 downto 0));

end cpu_ui_interface;


architecture arch of cpu_ui_interface is

begin

process(cpu_clk)

begin
        if rising_edge(cpu_clk) then
                case cpu_addr is
                        when "00" => -- Switches (only read)
                                if cpu_re='1' and cpu_cs='1' then
                                        cpu_data_out(31 downto 18)<=(others => '0');
                                        cpu_data_out(17 downto 0)<= SW;
                                end if;
                        when "01" => -- Buttons (only read)
                                if cpu_re='1' and cpu_cs='1' then
                                        cpu_data_out(31 downto 4)<=(others => '0');
                                        cpu_data_out(3 downto 0)<= KEY;
                                end if;
                        when "10" => -- Red leds
                                if cpu_we='1' and cpu_cs='1' then
                                        LEDR<=cpu_data_in(17 downto 0);
                                elsif cpu_re='1' and cpu_cs='1' then
                                        cpu_data_out(31 downto 18)<=(others => '0');
                                        cpu_data_out(17 downto 0)<=LEDR'driving_value;
                                end if;
                        when "11" => -- Green leds
                                if cpu_we='1' and cpu_cs='1' then
                                        LEDG<=cpu_data_in(8 downto 0);
                                elsif cpu_re='1' and cpu_cs='1' then
                                        cpu_data_out(31 downto 10)<=(others => '0');
                                        cpu_data_out(8 downto 0)<=LEDG'driving_value;
                                end if;
                end case; -- cpu_addr
        end if; -- rising_edge(cpu_clk)

end process;
```

```
end architecture arch;
```

B.4.3 SRAM memory interface - meminterface_cpu

```
-----------------------------------------------------------
-- FileName:            $HeadURL: svn://paddington.ros.sgsnet.se/digiosc/trunk/adc/source/meminterface_cpu.vhd $
-- Last changed:        $LastChangedDate: 2006-12-12 16:42:09 +0100 (ti, 12 dec 2006) $
-- By:                          $LastChangedBy: majohan $
-- Revision:            $Revision: 177 $
-----------------------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;


entity meminterface_cpu is
        port(pgm_clk            : in            std_logic;
                 reset                  : in            std_logic;
                 mem_data               : in            std_logic_vector(15 downto 0);
                 cpu_address    : in            std_logic_vector(17 downto 0);
                 cpu_chipselect : in            std_logic;
                 cpu_read               : in            std_logic;
                 cpu_readdata   : out           std_logic_vector(15 downto 0);
                 cpu_waitrequest: out           std_logic;
                 mem_addr               : out           std_logic_vector(17 downto 0);
                 mem_oe                 : out           std_logic;
                 req_n                  : out           std_logic
                );
end meminterface_cpu;


architecture arch of meminterface_cpu is

        constant S0 : std_logic_vector(5 downto 0 ) := "000001";
        constant S1 : std_logic_vector(5 downto 0 ) := "000010";
        constant S2 : std_logic_vector(5 downto 0 ) := "000100";
        constant S3 : std_logic_vector(5 downto 0 ) := "001000";
        constant S4 : std_logic_vector(5 downto 0 ) := "010000";
        constant S5 : std_logic_vector(5 downto 0 ) := "100000";

begin


process(pgm_clk,reset)

        variable state          : std_logic_vector(5 downto 0);

begin

if reset='0' then
        state:=S0;
        req_n <='1';

elsif rising_edge(pgm_clk) then

                case state is
                        when S0 => -- cpu should wait, req is sent to be able to control memory
                                if cpu_chipselect='1' then
                                        cpu_waitrequest<='1';
                                        req_n <='0';
                                        state(0):='0';
                                        state(1):='1'; -- S1
                                end if;
                        when S1 => -- output address
                                mem_addr<=cpu_address;
                                state(1):='0';
                                state(2):='1'; -- S2
                        when S2 => --open memory
                                mem_oe<='0';
                                state(2):='0';
                                state(3):='1'; -- S3
                        when S3 => -- read data
                                cpu_readdata <= mem_data;
                                state(3):='0';
                                state(4):='1'; -- S4
                        when S4 => -- give data to cpu
                                cpu_waitrequest<='0';
                                state(4):='0';
                                state(5):='1'; -- S5
                        when others => -- S5 -- close memory and withdraw req
                                mem_oe<='1';
                                req_n <='1';
                                state:=S0;
                end case;



end if; -- rising_edge

end process;

end architecture arch;
```

B.4.4 SDRAM memory interface - vga_memory_interface

```
-- $Id: vga_memory_interface.vhd 122 2006-12-08 09:13:06Z jonatan $

-- this is the toplevel module for the VGA interface. It has two Avalon-MM
-- ports, one master and one slave. The master requests transfers from memory
-- to the local FIFO and the slave accepts control data from the CPU. Available
-- registers:
-- name         description                          offset    direction R/W
-- AddrOfst     Framebuffer base address in memory   0x00      W

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.STD_LOGIC_UNSIGNED.all;

entity vga_memory_interface is

  port (
    clk_50 : in std_logic;                 -- 50 MHz clock
    clk_100 : in std_logic;              -- 100 MHz clock
    rstn : in std_logic;                 -- active low reset

    -- signals to/from vga_controller (type export)
    iVGA_FBaddr : in std_logic_vector(18 downto 0);  -- pixel address requested
                                                     -- from vga_controller
    oVGAdata : out std_logic_vector(1 downto 0);  -- pixel data to vga_controller

    -- Avalon-MM master port: manages read requests from SDRAM to fill VGA FIFO
    -- inputs:
    waitreq : in std_logic;              -- '1' == port busy, wait for '0' until
                                         -- transfer request can begin (hold
                                         -- mem_addr, burstcount and readreq
                                         -- constant until waitreq == '0')
    readdatav : in std_logic;            -- data presented on data_in is valid,
                                         -- i.e. use this as WE for the FIFO
    data_in : in std_logic_vector(15 downto 0);
    -- outputs:
    mem_addr : out std_logic_vector(23 downto 0);  -- 24 bits => up to 16MB
                                                   -- address space. Addresses
                                                   -- bytes restricted to word
                                                   -- boundaries, i.e. 0x00
                                                   -- 0x02 0x04 0x06 are the
                                                   -- first four valid
                                                   -- addresses each returning
                                                   -- 2 bytes of data.
    burstcount : out std_logic_vector(3 downto 0);  -- "1000" == burst of 8 x
                                                    -- 16 bit words (maximum burst)
    readreq : out std_logic;             -- '1' == read request

    -- Avalon-MM slave port: control interface which presents the VGA-hardware
    -- registers to the cpu
    -- inputs:
    reg_addr : in std_logic_vector(1 downto 0);  -- address presented by "cpu"
    reg_data : in std_logic_vector(31 downto 0);  -- data bus full 32 bits only
                                                  -- input
    reg_we : in std_logic;               -- slave port WE
    reg_cs : in std_logic               -- slave port CS
  );

end vga_memory_interface;

architecture vga_memory of vga_memory_interface is
  signal AddrOfst : std_logic_vector(23 downto 0);  -- Memory address offset
  signal pixAddr : std_logic_vector(2 downto 0);  -- subaddress in each 16 bit word
  signal wrreq_sig : std_logic;             -- write request to FIFO
  signal empty_sig : std_logic;         -- FIFO empty
  signal full_sig : std_logic;          -- FIFO full
  --signal almost_empty : std_logic;
  signal rd_req : std_logic;                    -- request new word to decoder
  signal q_sig : std_logic_vector(15 downto 0);  -- data from FIFO
  signal usedw_sig : std_logic_vector(8 downto 0);  -- number of used words in FIFO
  signal FIFOclear_sig : std_logic;     -- asynchroneous clear FIFO

  component vga_address_decoder
    port (
      clk_100       : in std_logic;
      clk        : in  std_logic;
      rstn          : in  std_logic;
      iVGAaddr      : in  std_logic_vector(18 downto 0);
      iAddrOfst     : in  std_logic_vector(23 downto 0);
      iFIFOusedw    : in std_logic_vector(8 downto 0);
      oMemAddr      : out std_logic_vector(23 downto 0);
      oPixAddr      : out std_logic_vector(2 downto 0);
      oFIFORdRq     : out std_logic;
      oFIFOclear    : out std_logic
    );
  end component;

  component vga_fifo
    port (
      aclr              : in STD_LOGIC;
      rdclk             : IN STD_LOGIC ;
      wrclk             : IN STD_LOGIC ;
      data              : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      rdreq             : IN STD_LOGIC ;
      wrreq             : IN STD_LOGIC ;
      rdempty           : OUT STD_LOGIC ;
      wrfull            : OUT STD_LOGIC ;
      q                 : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
```

```
      wrusedw           : OUT STD_LOGIC_VECTOR (8 DOWNTO 0)
    );
  end component;
  type burst_states is (init, wait_req, read_data);
  signal current_state : burst_states;
  signal burst_count : integer range 0 to 15;
  signal mem_addr_sig : std_logic_vector(23 downto 0);  -- local storage of memory address
  signal expecting_data : std_logic;    -- 1 when waiting for data from memory

begin  -- vga_memory
  burstcount <= "1000";
--  vga_fifo_empty <= empty_sig;

  addr_decode : vga_address_decoder port map(
    clk_100      => clk_100,
    clk          => clk_50,
    rstn         => rstn,
    iVGAaddr     => iVGA_FBaddr,
    iAddrOfst    => AddrOfst,
    iFIFOusedw   => usedw_sig,
    oMemAddr     => mem_addr_sig,
    oPixAddr     => pixAddr,
    oFIFORdRq    => rd_req,
    oFIFOclear   => FIFOclear_sig
  );

  vga_fifo_inst : vga_fifo PORT MAP (
    rdclk        => clk_50,
    wrclk        => clk_100,
    data         => data_in,
    rdreq        => rd_req,
    wrreq        => wrreq_sig,
    rdempty      => empty_sig,
    wrfull       => full_sig,
    q            => q_sig,
    wrusedw      => usedw_sig,
    aclr         => FIFOclear_sig
  );


  wrreq_sig <= expecting_data and readdatav;  -- generate write signal to FIFO
                                              -- when expecting data and data is
                                              -- valid

  process (clk_100,rstn)
  begin
    if rising_edge(clk_100) then
      if rstn = '0' then
        current_state <= init;
        expecting_data <= '0';
      else

        --state machine to control SDRAM->FIFO transfers
        case current_state is

          -- init state: wait until space available in FIFO
          when init =>
            burst_count <= 0;
            expecting_data <= '0';
            if usedw_sig < "111110111" then  -- is there space to fill up the FIFO?
              readreq <= '1';           -- yes, issue read request from memory
              mem_addr <= mem_addr_sig;  -- do this here so that the address is
                                         -- stable throughout the entire burst
              current_state <= wait_req;
            else
              readreq <= '0';           -- no, don't issue read request from memory
            end if;

          -- wait_req state: wait until requested word is ready from memory
          when wait_req =>
            if waitreq = '0' then       -- is the data ready?
              current_state <= read_data;  -- yes, write word into FIFO
              --readreq <= '0';
              expecting_data <= '1';
              readreq <= '0';
            else
              expecting_data <= '0';             -- no, don't write word into FIFO
            end if;

          -- read_data state: write data from memory into FIFO
          when read_data =>
            if readdatav = '1' then
              burst_count <= burst_count + 1;  -- one more valid word written
                                               -- into the FIFO
            end if;
            if burst_count < 7 then     -- have we received 8 words?
              current_state <= read_data;  -- no, continue receiving
            else
              current_state <= init;    -- yes, stop writing words into FIFO
                                        -- and go to init (waiting) state
              expecting_data <= '0';    -- stop expecting data from memory
            end if;

        end case;
      end if;
    end if;
  end process;

  process (clk_50, rstn)
```

```
  begin  -- process
    if rising_edge(clk_50) then
      if rstn = '0' then                  -- synchronous reset (active low)
        AddrOfst <= (others => '0');
      else
        -- output 2-bit pixel data from 16-bit memory word
        case pixAddr is
          when "000" => oVGAdata <= q_sig(1 downto 0);
          when "001" => oVGAdata <= q_sig(3 downto 2);
          when "010" => oVGAdata <= q_sig(5 downto 4);
          when "011" => oVGAdata <= q_sig(7 downto 6);
          when "100" => oVGAdata <= q_sig(9 downto 8);
          when "101" => oVGAdata <= q_sig(11 downto 10);
          when "110" => oVGAdata <= q_sig(13 downto 12);
          when "111" => oVGAdata <= q_sig(15 downto 14);
        end case;
        if (reg_cs = '1' ) and (reg_we = '1') then           -- save new address offset
          AddrOfst <= reg_data(23 downto 0);
        end if;
      end if;
    end if;
  end process;

end vga_memory;
```

B.5 Programkod

B.5.1 Huvudprogram

```
/* $Id: testapli.c 87 2006-12-01 11:20:57Z jonatan $
 */

#include "system.h"
#include "testapli.h"
#include "digiosc_vga_shared_memory_ctl.h"
#include "digiosc_adc_ctl.h"
#include "digiosc_led_ctl.h"
#include "plotter.h"
#include "print_char.h"

int main(void)
{
  int update_fb = 0;
  char switch_value = 0;
  int leds=0;
  int i;
  int val=0;

  //initialisation
  init_char();
  current_fb = fb0;
  fb0_set = 1;
  digiosc_adc_cpu_unlock_mem();

  blank_screen(current_fb);

  digiosc_vgamem_setofst(current_fb);
  current_fb = fb1;
  fb0_set = 0;
  // main loop
  while(1)
  {
      digiosc_write_leds_red(&leds);
      leds++;

      val+=digiosc_adc_adc_ready();
      digiosc_write_leds_green(&val);

      update_fb = checkSampling(current_fb);


      if (1==update_fb)
      {
       int totalSamples = ((digiosc_read_switches() & 0x7F) << 10); // Switches 6 to 0 times 1024
       print_string("TOTAL SAMPLES: ",&current_fb[50*520+6], CHAR_WHITE);
       print_int(totalSamples,6,&current_fb[50*520+13], CHAR_WHITE);
       print_string("TRIGGER ",&current_fb[50*520+17], CHAR_WHITE);
       print_string(((digiosc_read_switches() & 0x80)?"POSITIVE, ":"NEGATIVE, "),&current_fb[50*520+21], CHAR_WHITE);
       print_string(((digiosc_read_switches() & 0x100)?"ENABLED , LEVEL: ":"DISABLED, LEVEL: "),&current_fb[50*520+26], CHAR_WHITE);
       print_int(((digiosc_read_switches() & 0x2FE00) >> 2),6,&current_fb[50*520+33], CHAR_WHITE);

          //byt till ny fb
          digiosc_vgamem_setofst(current_fb);

          if (fb0_set==1)
          {
              fb0_set = 0;
              current_fb = fb1;
          }
          else
          {
              fb0_set = 1;
              current_fb = fb0;
          }
          blank_screen(current_fb);

      }
```

```
  }

  return 0;
}
```

B.5.2 Plotfunktion

```
#include "plotter.h"
#include "digiosc_adc_ctl.h"
#include "print_char.h"
#include "digiosc_sram_mem.h"
#include "digiosc_led_ctl.h"


int checkSampling(int* current_fb)
{
    int fb_changed = 0;

    if (1==digiosc_adc_adc_ready())
    {
      int totalSamples = ((digiosc_read_switches() & 0x7F) << 10); // Switches 6 to 0 times 1024
      fb_changed = 1;
      digiosc_adc_cpu_lock_mem();

      // Print out status of samples to screen
      // WARNING! fix signed on trig level
       int totalSampleValue = 0;
      int max = 0;
      int min = 65535;
      int xPixel = 0;
      int fbXY = 0;
      const int maxX=700;
      const char GREEN = 0x1;
      const char RED = 0x2;
      int sampelsPerPixel=totalSamples/maxX;

      for (xPixel=0;xPixel<maxX;xPixel++)
      {
        signed short int sampelValue = (signed short int)(digiosc_sram_read(xPixel*sampelsPerPixel) & 0xFFFF);
        totalSampleValue += sampelValue;
        if (sampelValue > max)max = sampelValue;
        if (sampelValue < min)min = sampelValue;
         if(xPixel == 0)
        {
          print_int(sampelValue,6,&current_fb[50*290+16], CHAR_WHITE);
        }

        int yPixel = 262 - (sampelValue >> 7);

        fbXY=100+xPixel + 800 * yPixel;

        *(current_fb+(fbXY >> 4)) |= ( GREEN << (((fbXY & 0xF )<< 1)));

      }
      digiosc_adc_cpu_unlock_mem(); // Writing 1 to flag finished

      print_string("MEAN VALUE: ",&current_fb[50*530+6], CHAR_GREEN);
      print_int(totalSampleValue/sampelsPerPixel,6,&current_fb[50*530+10], CHAR_GREEN);
      print_string("MAXIMUM VALUE: ",&current_fb[50*530+14], CHAR_GREEN);
      print_int(max,6,&current_fb[50*530+20], CHAR_GREEN);
      print_string("MINIMUM VALUE: ",&current_fb[50*530+24], CHAR_GREEN);
      print_int(min,6,&current_fb[50*530+30], CHAR_GREEN);


    }

      return fb_changed;
}
```

B.5.3 Teckenhantering

```
/* $Id: print_char.c 87 2006-12-01 11:20:57Z jonatan $
 */

#include "print_char.h"
#include "math.h"

void convert_left(int*, int*, int);
void convert_right(int*, int*, int);
int print_char_left(char *c,int *location, int color);
int print_char_right(char *c,int *location, int color);

/* adds character mask with current display (OR mask)
 * int *c      address to character array to display
 * int *loc    base (top left) memory address to write to
 */
void convert_right(int *c, int *loc, int color)
{
  int i;
  for (i = 0; i < 5;i++)
  {
    //write pixel on *(loc + offset)
```

```
    //write two pixel rows for 8x10 pixel character
    *(loc+2*i*50) = *(loc+2*i*50) | (c[i] & 0xFFFF0000) & color;
    *(loc+(2*i+1)*50) = *(loc+(2*i+1)*50) | (c[i] << 16) & color;
  }
  return;
}

/* adds character mask with current display (OR mask)
 * int *c      address to character array to display
 * int *loc    base (top left) memory address to write to
 */
void convert_left(int *c, int *loc, int color)
{
  int i;
  for (i = 0; i < 5;i++)
  {
    //Skriv pixel på start (loc) + offset(
    //Skriv två rader med pixlar för ett tecken 8x10
    *(loc+2*i*50) = *(loc+2*i*50) | (c[i] >> 16 ) & color;
    *(loc+(2*i+1)*50) = *(loc+(2*i+1)*50) | (c[i]& 0x0000FFFF) & color;
  }
  return;
}

/* prints a character on screen
 * char *c        character to print
 * int *location  base (top left) memory address to print to
 *                this pointer is updated to point to the next
 *                character position after successful execution
 */
int print_char(char *c,int *location, int color)
{

    convert_left(&char_c[*c],location, color);
    return 0;
}

int print_char_right(char *c,int *location, int color)
{
    convert_right(&char_c[*c],location, color);
    return 0;
}

int print_char_left(char *c,int *location, int color)
{
    convert_left(&char_c[*c],location, color);
    return 0;
}

int print_string(char *c, int *location, int color)
{
    int i = 0;
    while(1) {
        if ('\0' == c[i]){break;}
        print_char_left(&c[i],location, color);
        if ('\0' == c[i+1]){break;}
        print_char_right(&c[i+1],location, color);
        location++;
        i+=2;
    }
    return 0;
}

/* prints an integer using a fixed number of digits
 *
 */
int print_int(int number, int length, int *location, int color)
{
  if (length > 10)
    return -PRINTCHAR_ERR_TOOBIGNUMBER;

  char isNeg = 0, nullDone=0;
  int i;
  //int digit,middle = 0;
  char str[length+2];

  if (number<0) {isNeg=1; number*=-1;  }

  for (i=length;i>=0;i--)
  {
    if (nullDone==0)
    {
      str[i]=number%10+48;
      number/=10;
    } else str[i]=' ';
    if (number==0)
    {
      if(isNeg==1 && nullDone==0)
      {
        nullDone=1;
        str[--i]='-';
      }
      else
      {
        nullDone=1;
        str[--i]=' ';
      }

    }
```

```
  }

  str[length+1] = '\0';

  number = print_string(str, location, color);
  return number;
}
```

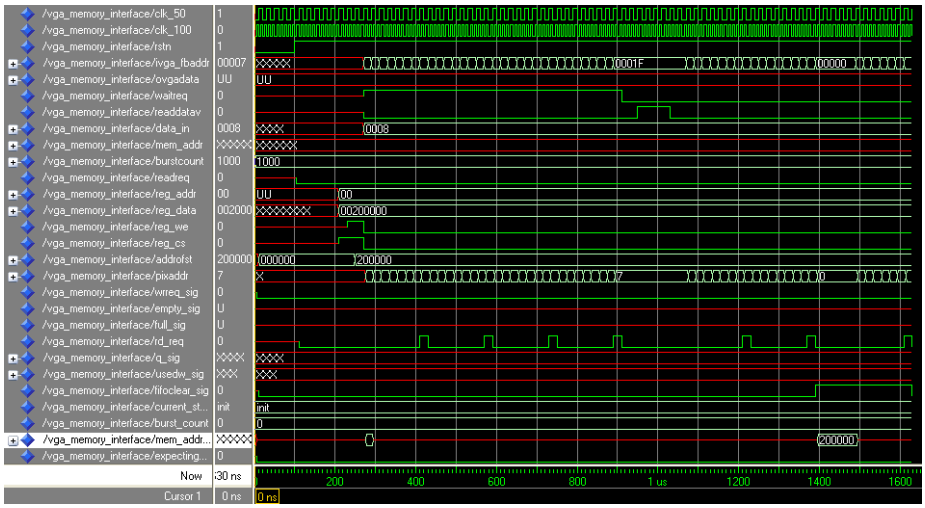
C Syntesresultat

Sammanfattning av resultat vid syntes med Quartus II:

Quartus II Version	6.0 Build 178 04/27/2006 SJ Web Edition
Family	Cyclone II
Device	EP2C35F672C6
Total logic elements	1,352 / 33,216 (13%)
Total registers	2595
Total pins	170 / 475 (36%)
Total virtual pins	0
Total memory bits	24,496 / 483,840 (5%)
Embedded Multiplier 9-bit elements	8 / 70 (11%)
Total PLLs	1 / 4 (25%)

D Simuleringsresultat

I Figur 1 visas resultatet från en simulering av minnesinterfacet mot VGA-drivenheten. Det som testas är att interfacet genererar läsförfrågningar mot FIFOn samt tömmer den efter att sista pixeln är ritad. Många signaler är odefinierade eftersom FIFOn inte är medtagen i simuleringen.



Figur 1: Simulering av minnesinterfacet mot VGA-drivenheten