TEIS
Teknisk rapport

Författare Lasse Karagiannis
Uppgift 7

TEIS ECS – Embedded Computer System

Kontrollerad JA
Version 1

Fil Lasse_Karagiannis_vhdl_uppgift_7.pdf

TEIS AB

TEIS ECS - Embedded Computer System -

Lasse Karagiannis

16-10-30

Sammanfattning: TEIS datorsystem är en enkel men komplett dator som exekveras på ett FPGA-kort. Datorn består av en CPU, adressbuss-decoder, ROM, samt ett ingångsfilter för att välja manuell eller automatisk klockning av processorn. Datorn presenterar även register, bussar och chip select signaler på kortets sjusegmentdisplayer och lysdioder.

INNEHÅLLSFÖRTECKNING

1 INLEDNING

Denna rapport beskriver ett datorsystem skrivet i VHDL. Systemet har analyserats genom simulering och verifiering i ModelSim och därefter validerats på ett lämpligt FPGA-kort.

2 KRAVSPECIFIKATION

Tabell 1. Kravspecifikation från TEIS uppgift_vhdl_7

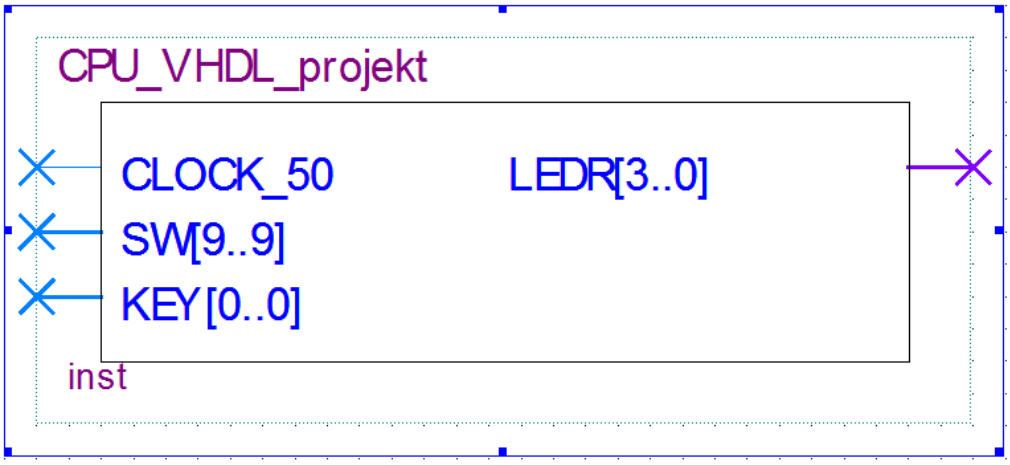
Kravspecifikation med checklista för utförda krav.

Krav	Beskrivning	Utfört Ja/Nej
0	Projektnamn: "TEIS_ECS" (TEIS Embedded Computer System)	JA
1	Ladda ner och slutför rapporten, utgå ifrån den äldre rapporten som finns i schemat (fornamn_efternamn_vhdl_uppgift_7). Kunden godtar att du blandar svenska och engelska.	JA
2	Granska koden och ge synpunkter på förbättringar.	JA
3	Beräkna kostnaden för projektet. Företaget debiterar kunden med 350 SEK/tim.	JA
Leverans		
4	Leveransen ska ske till Itslearning. Leveransen ska vara en mapp med: • rapport med namnet "fornamn_efternamn_vhdl_uppgift_7" (word eller pdf) och • arkiverat Quartus-projekt. Namnet på mappen ska vara "fornamn_efternamn_vhdl_uppgift_7" (zip eller rar). Sista leveransdagen se kurs schemat.	JA

3 ÖVERSIKT AV SYSTEMARKITEKTUR OCH KOMPONENTHIERARKI

TEIS datorsystem består av en CPU, adressbuss-decoder, ROM, samt ett ingångsfilter för att välja manuell eller automatisk klockning av processorn. Resultat presenteras på LEDG 3..0. Datorn klockas normalt manuellt för operatör ska kunna se vad som händer vid varje enskild klockcykel. I Figur 1 visas toppnivån med in och utgångar. Figur 2 visar det kort som används. Figur 3 visar systemarkitekturen och figur 4 hierarkien.

3.1 Symbol



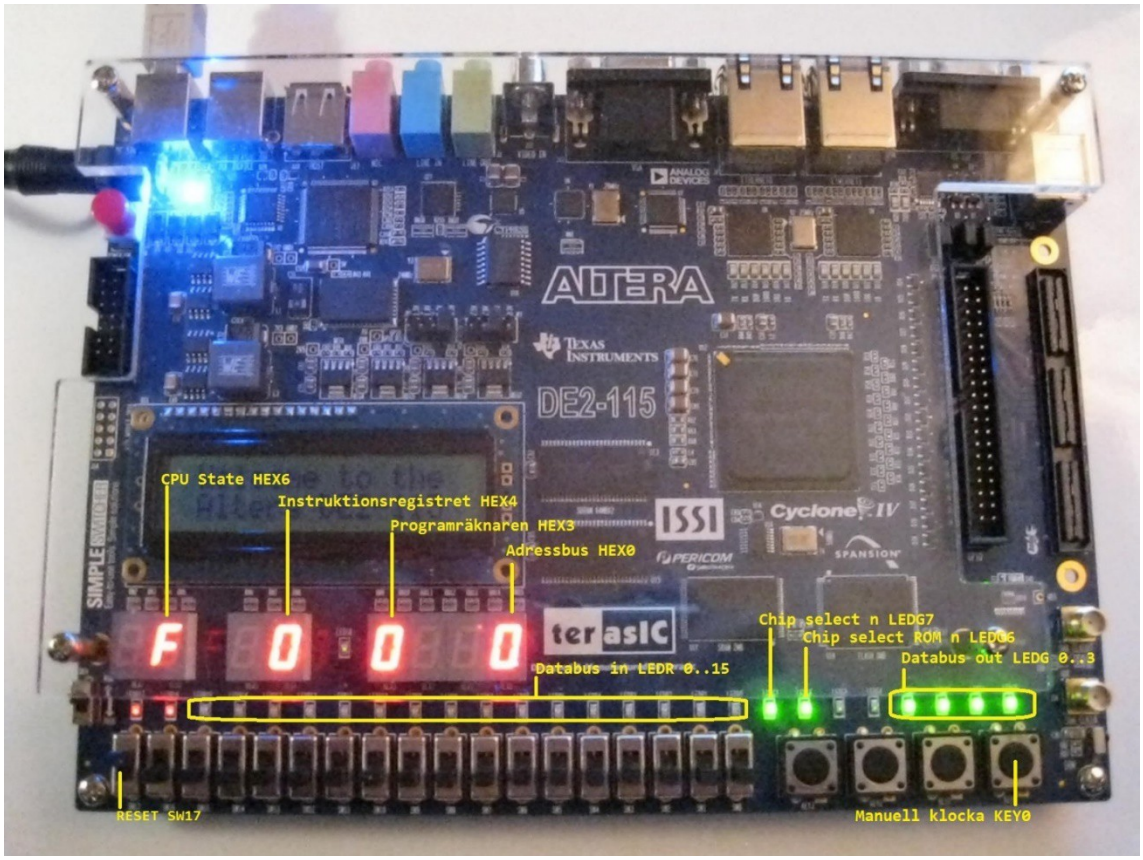
Figur 1. Toppnivån för TEIS mikrodatorsystem

3.2 In/utgångar

Systemet har 3 ingångar, egentligen 4 stycken varav en av dem *Use_Manual_Clock* är kopplad direkt till Vcc. Detta framkommer i INPUT_FILTER.vhd, som är det första delsystemet, användarens input möter. SW9 är reset_n. KEY0 är den manuella klockan.
Så här ser Pin plannern ut:

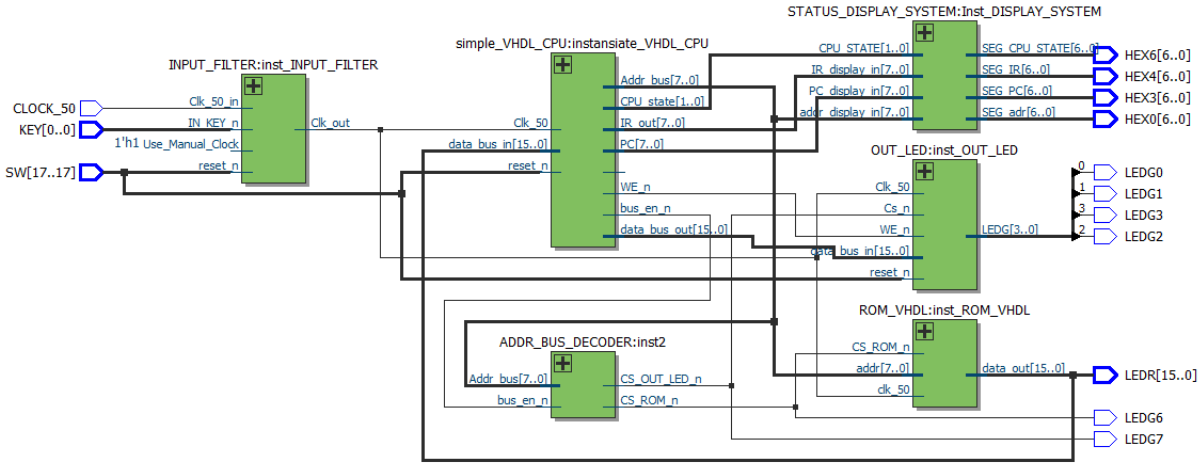
Node Name	Direction	Location
in CLOCK_50	Input	PIN_AF14
in KEY[0]	Input	PIN_AA14
out LEDR[3]	Output	PIN_V18
out LEDR[2]	Output	PIN_V17
out LEDR[1]	Output	PIN_W16
out LEDR[0]	Output	PIN_V16
in SW[9]	Input	PIN_AE12

Figur 2. Pinn planner vyn för TEIS mikrodatorsystem

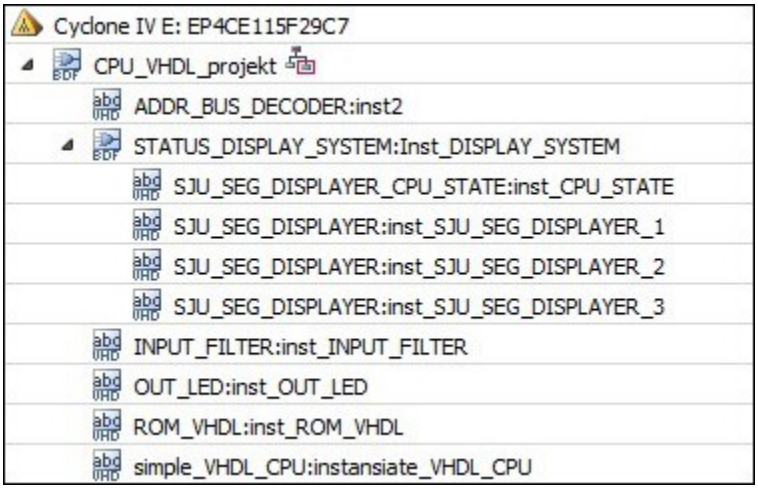


Figur 3. TEIS datorsystem på DE2-115 **(byt figuren till det kortet som används i detta projekt)**

3.3 TEIS Embedded Datorsystem arkitektur (ECS)



Figur 4. Systemarkitekturen för TEIS datorsystem



Figur 5. Komponenthierarki i TEIS datorsystem

Systemet använder ett ROM för att lagra programkoden i. Se minnesmappen i tabell 2. Resultat lagras i systemets register.

Tabell 2. Minnesmapp

Typ	Minnesområde	Storlek	Minnestyp
Programkod	0x0 – 0xF	15	ROM
I/O	0x10	1	RAM

3.4 General package and library

Svara på frågorna:

1) Vilka ”library” och ”package” som används.

En kort presentation av varje ”package”
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;'

Bibliotket/Library som används är som vanligt IEEE. Tre olika packages används totalt i projektet och dessa är std_logic_1164, numeric_std samt std_logic_unsigned.
Package std_logic_1164 innehåller typdefinitionen för typen std_logic samt std_logic_vector samt funktionsdeklarationer för behandling av signaler av denna typ.

Package numeric_std introducerar signed och unsigned och tillhandahåller funktioner för numerisk manipulering, som var tänkt att ersätta ett package skrivet av företaget Synposis, nämligen std_logic_unsigned och std_logic_signed.
Använder man numeric_std kan man blanda hur man ska tolka en std_logic_vector, på varje programrad för sig.

4 INGÅENDE KOMPONENTER

4.1 CPU – komponent

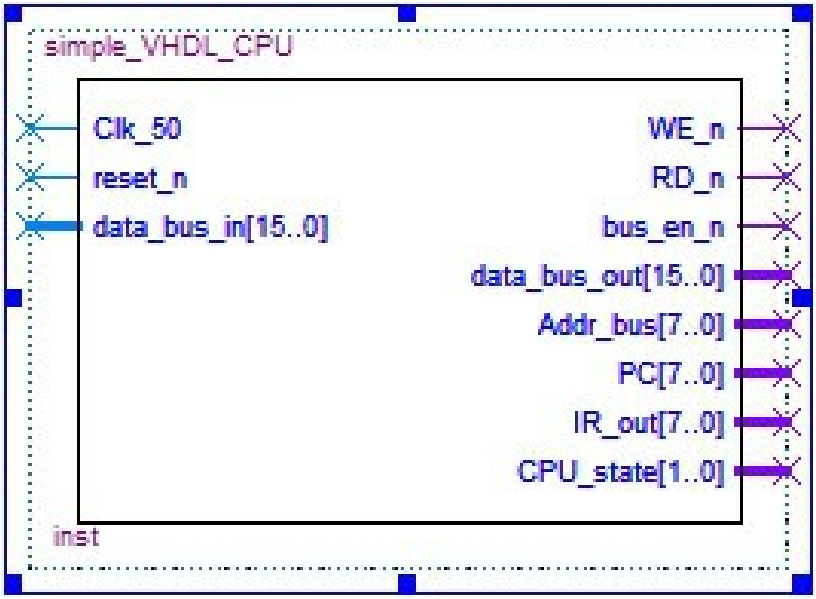
Komponent namn: simple_VHDL_CPU

Instansnamn: instansiate_VHDL_CPU

4.1.1 Funktion, arkitektur och tillståndsmaskin

CPU-komponentens in och utgångar visas i nästa figur. CPU:n styrs av reset eller klocksignal. CPU:n arbetar med en tillståndsmaskin som kan exekvera instruktionerna NOP, LOAD, STORE och JMP. Internt använder CPU:n programräknaren, instruktionsregistret och dataregistret. Programräknaren pekar ut vilken instruktion som skall hämtas från ROM.

Vid reset initieras programräknaren, bussar och register till 0 medan enable-signaler initieras till 1. Vid positiv klocksignal initieras enable-signalerna till 1 och går sedan in i tillståndsmaskinen. Se figur 5.



Figur 6. CPU symbol

4.1.2 In/utgångar

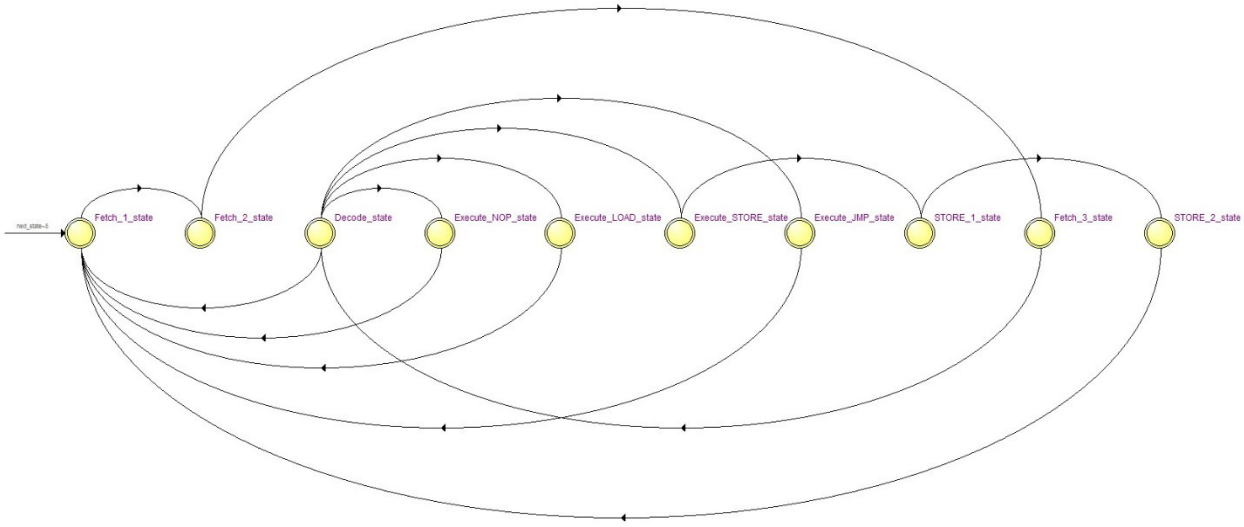
CPU:ns in och utsignaler visas i Tabell 3.

Tabell 3. CPU:ns in och utsignaler

Signal	Namn	Riktning	Typ
Clocksignal 50 MHz	Clk_50	in	std_logic
Reset	reset_n	in	std_logic
Write enable	WE_n	ut	std_logic
Read enable	RD_n	ut	std_logic
Bus enable	bus_en_n	ut	std_logic
Databus ut	data_bus_out	ut	std_logic_vector(15 downto 0)
Databus in	data_bus_in	ut	std_logic_vector(15 downto 0)
Adressbuss	Addr_bus	ut	std_logic_vector(7 downto 0)
Programräknare	PC	ut	std_logic_vector(7 downto 0)
Instruktionsregister	IR_out	ut	std_logic_vector(7 downto 0)
Tillstånd	CPU_state	ut	std_logic_vector(1 downto 0))

4.1.3 Tillståndsmaskin

Tillståndsmaskinen arbetar i fyra tillstånd, fetch, decode, execute och store. Fetch hämtar nästa instruktion. Decode avkodar vad som ska göras. Execute utför det som beslutats i decode-fasen. Store lagrar data till minnet. Figur 7 nedan visar tillståndsmaskinen i detalj.



Figur 7. Tillståndsmaskinen (rita om den så tillstånden kan urskiljas)

Tillstånds beskrivning:

- **Fetch**

  CPU_state får värdet ”00”. Data hämtas in till instruktionsregistret. Värdet på programräknaren läggs ut på adressbussen. rd och bus enable sätts till 0. I IR läggs instruktionen in från databussen.

- **Decode:**

  Först ökas programräknaren med ett och CPU_state får värdet ”01”. Berode på vilken instruktion som ligger i instruktionsregistret väljs vilken instruktion som skall exekveras i fasen Execute

- **Execute**

  CPU_state får värdet”10”. Om NOP exekveras görs ingenting. Om Load exekveras läggs IR in i dataregister. Om Store exekveras kopieras registret till databussen och adressbussen får värdet i IR. Om JMP exekveras får programräknaren värdet i IR

- **Store**

  CPU_state får värdet ”11”. Först sätts write enable och bus enable till 0 och sedan sätts de till 1.

4.1.4 Beskrivning av CPU:ns register, operationer, databuss, adressbuss och kontrollsignaler

CPU:n arbetar internt med tre olika register. Programräknare, instruktionsregister och dataregister. Se Tabell 4 för detaljerad beskrivning.

Tabell 4. CPUns interna register

Register	Namn	Beskrivning
Programräknare	PC_reg	Innehåller vilken adress CPU ska läsa från ROM
Instruktionsregister	IR	Innehåller vilken instruktion CPU ska utföra
Dataregister	CPU_REG_0	Internt register för att överföra data

Instruktioner är kodaded med 16 bitar, där de högsta 4 bitarna anger instruktionstypen och resterande 3 nibblar är argumentet till instruktionen. Samtliga nibblar används inte vid varje instruktion, detta framgår med ”-” i tabell 5, som beskriver de tillgängliga assemblerinstruktionerna.

Tabell 5: Assembler instruktioner, OP-kod och kodning

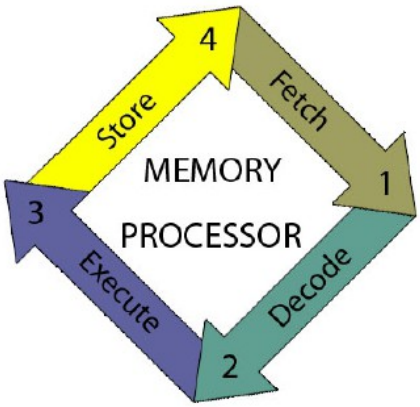
Instruktion	OP-kod	Exempel och förklaring
NOP	0x0---	No operation
LOAD_R0 #imm	0x1nnn	Load R0 with immediate
STORE_R0#ADDR	0x2-nn	Store R0 at 8-bit adress
JMP #ADDR	0x3-nn	Load IR content of code memory at adress nn

4.1.5 Beskrivning av CPU:ns arbetssätt

Maskinkoden eller operationskoden innehåller information som används av CPU:ns styrenhet. En digitalt system kan ses vara uppdelad i styrenhet och dataväg. Styrenheten manipulerar datavägen efter dedikerade bitfält i maskinkoden (OP-koden).

Assembler är maskinkoden skriven med en symbolisk beskrivning för att underlätta för programmeraren. Operationskoden beskriven med ettor och nollor, eller i Hex ersätts med en s.k. ”mnemonic”, ett minnesord. Mnemonics för instruktioner som hämtar data in till processorn, vare sig det är med ett omedelbart värde, skrivet i operationskoden s.k. ”immediate” eller från dataminnet, så heter instruktionerna antingen LOAD eller ld enligt konvention, med olika argument.
Rent allmänt kan sägas att om instruktionen har två eller flera argument så är det nästan alltid så, att det första argumentet till instruktionen är destinationspositionen, adressen/ registret där resultatet av operationen placeras.
Processorer exekverar konsekutiva instruktioner ur en lista, det s.k. kod-minnet. Processorn adresserar kod-minnet under den s.k. Fetch– fasen med programpekaren PC, och läser in aktuell instruktion i instruktionsregistret IR, detta går under en klockcykel.
Nästa klockcykel går processorn in i Decode-fasen. Om processorn arbetar enligt löpande band principen, s.k. ”pipelining” fortsätter Fetch-delen att göra redo för nästa Decode, men så är inte fallet för CPU:n i projektet. Under decode så ställer styrenheten om datavägen med hjälp av den avkodade instruktionen.

Nästa klockcykel är Execute, där resultatet från exempelvis ALU klockas in i interna resultat register. Sista klockcykeln görs Store då resultatet skrivs till minne, varvid CPU:n återupprepar processen med Fetch, se Figur 8.



Figur 8: CPU:ns arbetsätt

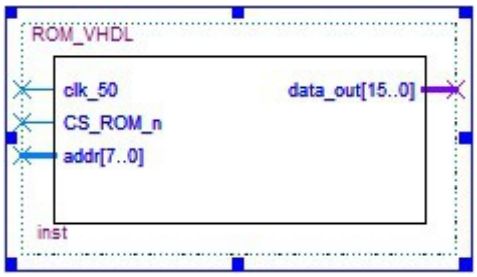
4.2 ROM – komponent

Komponent namn: ROM_VHDL

Instansnamn: inst_ROM_VHDL

4.2.1 Funktion och arkitektur

ROM används för att lagra maskinkoden som CPU:n ska exekvera. Vid positiv klockflank läggs data ut på databussen på den adress ROM har som insignal. Chip select används inte internt i ROM. ROM-komponentens in och utgångar visas i Figur 8, och innehållet i ROM visas i tabell 6. RTL-nivån på ROM visas i figur 9.



Figur 9. ROM Symbol

Tabell 6. Innehåll i ROM

Adress	Maskinkod [HEX]	Assemblerkod	Instruktion [HEX]	Data [HEX]
0	0000	NOP	0	000
1	100A	LOAD_R0 #A	1	00A
2	2010	STORE_R0 #10	2	010
3	1001	LOAD_R0 #1	1	001
4	2010	STORE_R0 #10	2	010
5	3001	JMP #1	3	001
6	0000	NOP	0	000
7	0000	NOP	0	000
8	0000	NOP	0	000
9	0000	NOP	0	000
10	0000	NOP	0	000
11	0000	NOP	0	000
12	0000	NOP	0	000
13	0000	NOP	0	000
14	0000	NOP	0	000
15	0000	NOP	0	000

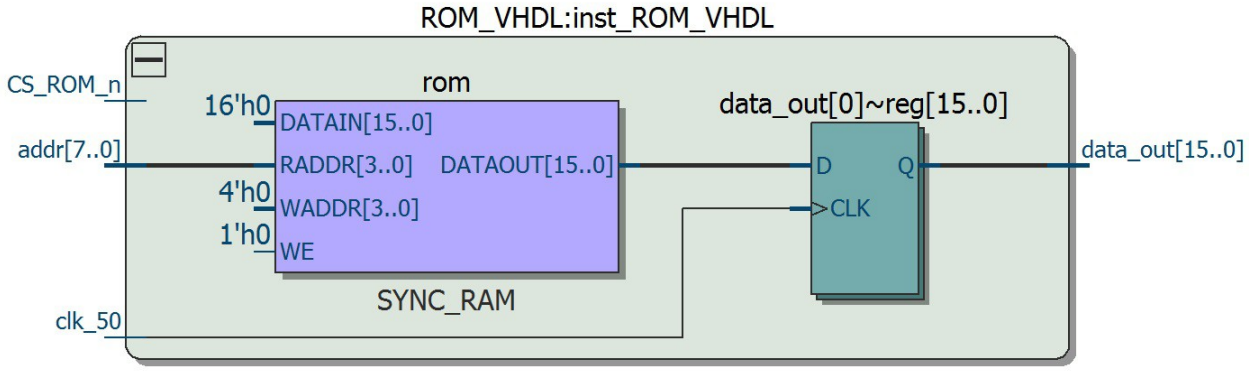
4.2.2 In/utgångar

In- och utsignalerna till ROM enligt Tabell 7.

Tabell 7. In- och utsignalerna till ROM

Signal	Namn	Riktning	Typ
Klocksignal 50 MHz	clk_50	in	std_logic
Chipselect	CS_ROM_n	in	std_logic
Adressbuss	addr	in	std_logic_vector(7 downto 0);
Data ut	data_out	ut	out std_logic_vector(15 downto 0)

4.2.3 RTL-nivå



Figur 10. RTL-nivån på ROM

4.2.4 VHDL-nivå

```
entity ROM_VHDL is
	port
	(
		clk_50, CS_ROM_n			: in std_logic;
		addr					: in std_logic_vector(7 downto 0);
		data_out				: out std_logic_vector(15 downto 0)
	);
end entity;

architecture rtl of ROM_VHDL is

	-- Build a 2-D array type for the RoM
	subtype word_t is std_logic_vector(15 downto 0);
	type memory_t is array(0 to 15) of word_t;

	signal rom : memory_t := memory_t'(
	X"0000", -- Adress 0; NOP
	X"100A", -- Adress 1; LOAD_R0 #A
	X"2010", -- Adress 2; STORE_R0 #10
	X"1001", -- Adress 3; LOAD_R0 #1
	X"2010", -- Adress 4; STORE_R0 #10
	X"3001", -- Adress 5; JMP #1
	X"0000",
	X"0000",
	X"0000", -- Adress 8
	X"0000",
	X"0000",
	X"0000",
	X"0000", -- Adress 12
	X"0000",
	X"0000",
	X"0000"); -- Adress 15


begin

	process(clk_50)
	begin
	if(rising_edge(clk_50)) then
		data_out <= rom(to_integer(unsigned(addr(3 downto 0))));
	end if;
	end process;

end rtl;
```

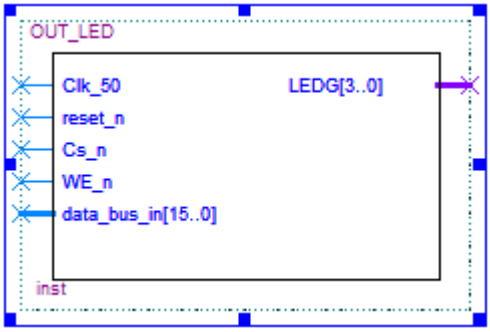
4.3 LED – komponent

Komponent: OUT_LED

Instansnamn: inst_OUT_LED

4.3.1 Funktion och arkitektur

OUT_LED är en periferienheten för hantering av 4 stycken lysdioder.
Utsignalerna är kopplade till LEDR[3..0] enligt Pin-planner
Den har insignalerna Clk_50, reset_n, Cs_n, WE_n och data_bus_in, som är klocka, reset, chip select, write enable och en 16-bitars data-port in LED. Figur 10 ger en schematisk bild av komponentens in och utgångar. Tabell 8 beskriver komponentens in och utgångar typer. Figur 11 visar RTL implementationen av VHDL-koden som listas i avsnitt 4.3.4.



Figur 10 OUT_LED

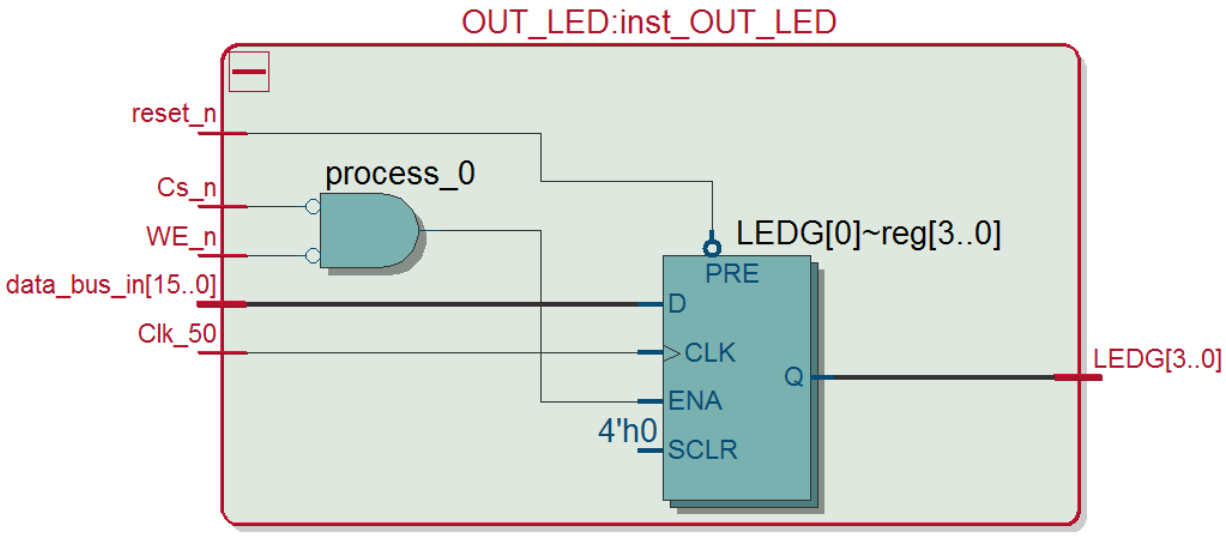
4.3.2 In/utgångar

In och utsignaler till LED_OUT enligt Tabell 8

Tabell 8 In och utsignaler OUT_LED

Signal	Namn	Riktning	Typ
Klocksignal 50 MHz	clk_50	in	std_logic
Reset	reset_n	in	std_logic
Chip select	Cs_n	in	std_logic
Data_in	data_bas_in	in	std_logic_vector(15 downto 0)
LED-signaler	LEDG	ut	std_logic_vector(3 downto 0)

4.3.3 RTL-nivå



Figur 11 RTL-nivån för OUT_LED

4.3.4 VHDL-kod

```
library ieee;
use ieee.std_logic_1164.all;


entity OUT_LED is  -- out to leds
        port
        (
        Clk_50, reset_n        : in std_logic;
        Cs_n, WE_n             : in std_logic;
        data_bus_in            : in std_logic_vector(15 downto 0);
        LEDG                   : out std_logic_vector(3 downto 0));
End entity;


architecture rtl of OUT_LED is


Begin
        process(reset_n, clk_50)
        begin
                if reset_n = '0' then
                        LEDG <= "1111";                 -- LED-signals go high during reset
                elsif(rising_edge(clk_50)) then
                        if CS_n = '0' and WE_n = '0' then   --if CS_n and WE_n lowest nibble
                                LEDG <= data_bus_in(3 downto 0);-- at data_bus_in is sent to LEDs
                        end if;
                end if;
        end process;
end rtl;
```

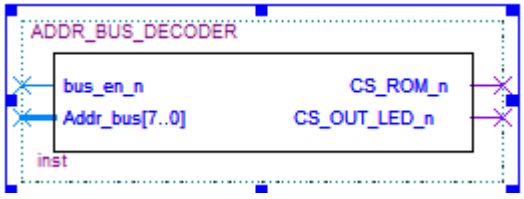
4.4 Adressbussdekoder – komponent

Komponent: ADDR_BUS_DECODER

Instansnamn: inst2

4.4.1 Funktion och arkitektur

Adressbussdekodern beslutar om ROM ska väljas eller I/O. Denna har insignalerna bus_en_n samt 8-bitars vektorn Addr_bus. Dekodern undersöker om adressen är 0-15, i såfall väljer decodern att enabla CS_ROM_n, men om adressen är 16, så väljer dekodern att enabla CS_OUT_LED_n. Figur 12 ger en schematisk bild av komponenten. Tabell 9 innehåller in och utgångarnas typer. Figur 13 visar RTL-nivån.



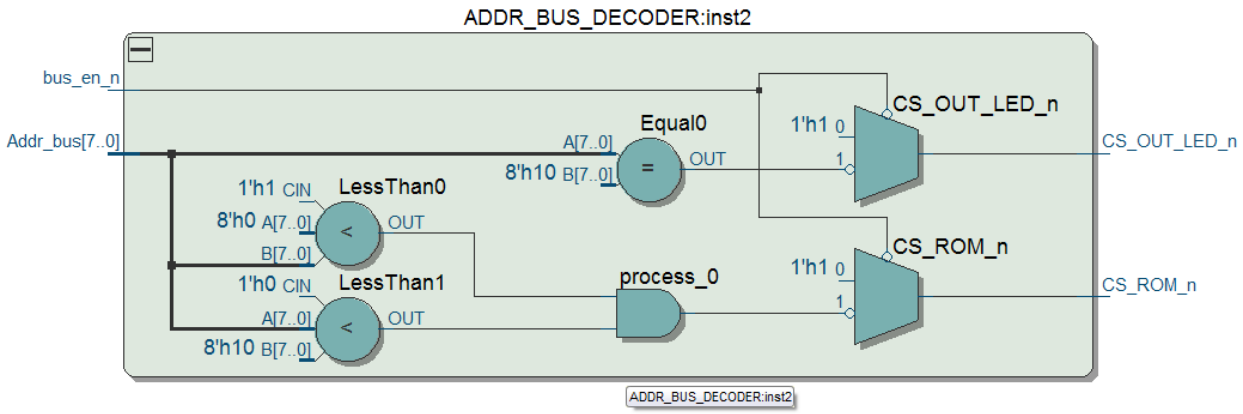
Figur 12 ADDR_BUS_DECODER

4.4.2 In/utgångar

Tabell 9 In och utsignaler för ADDR_BUS_DECODER

Signal	Namn	Riktning	Typ
Buss enable	bus_en_n	in	std_logic
Adress buss	Addr_bus	in	std_logic_vector(7 downto 0)
Chip select ROM	CS_ROM_n	ut	std_logic
Chip select LED	CS_OUT_LED_n	ut	std_logic

4.4.3 RTL-nivå



Figur 13 RTL-nivån för ADDR_BUS_DECODER

4.4.4 VHDL-kod

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ADDR_BUS_DECODER is
	port(
		bus_en_n	: in std_logic;
		Addr_bus	: in std_logic_vector(7 downto 0);
		CS_ROM_n	: out std_logic;
		CS_OUT_LED_n	: out std_logic		);
end entity ADDR_BUS_DECODER;

architecture rtl of ADDR_BUS_DECODER is
Begin
	process(Addr_bus, bus_en_n)
	begin
		if bus_en_n = '0' then   -- Om buss enable
			if unsigned(addr_bus) >= 0 AND unsigned(addr_bus) < 16 then
								-- 0-15; ROM adress
				CS_ROM_n <= '0'; -- ROM väljs
			else
				CS_ROM_n <= '1'; -- ROM väljs ej
			end if;
			- Är adressen 16?
			if addr_bus = "00010000" then  -- 16 ; OUT_LED adress
				CS_OUT_LED_n <= '0'; -- LED väljs
			else
				CS_OUT_LED_n <= '1'; -- Led väljs inte
			end if;
		else	-- Om inte bus enable
			CS_ROM_n <= '1';-- välj inte ut någon alls
			CS_OUT_LED_n <= '1';
		end if;
	end process;
end rtl;
```

4.5 Ingångsfilter – komponent

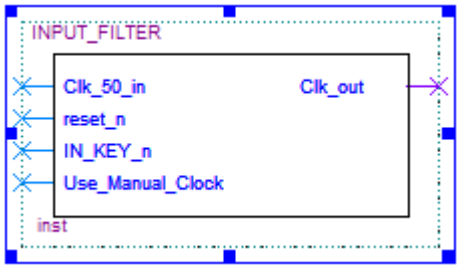
Komponent: INPUT_FILTER

Instansnamn: inst_INPUT_FILTER

Generisk parameter: cnt_high av typen integer, initierad till 20.

4.5.1 Funktion och arkitektur

Ingångsfiltret används för att generera en klocksignal till CPU. Klocksignalen kan väljas som intern eller manuell klockning med tryckknapp. Om Use_Manual_Clock = 0 används kortets klocka annars används tryckknapp KEY0 på kortet. Filtret använder en generic, cnt_high för att bestämma hur många klockpulser som behövs för att signalen ska anses som stabil. Key0 läses, och sparas under tiden för två stigande klockflanker. Om föregående värde för KEY0 skiljer sig från nu inlästa värde har en förändring skett, så nollställer vi räknaren som räknar upp till 50 000. Om högsta bit i räknaren fortfarande noll så fortsätt att klocka räknaren, men om räknarens högsta bit är 1 så skicka ut det värdet från Key0. Figur 14 ger en schematisk vy över in och ut -portarna. Tabell 10 listar typerna för ingångarna och utgångarna.



Figur 14 INPUT_FILTER

4.5.2 In och utgångar

Tabell 10 In och utsignaler INPUT_FILTER

Signal	Namn	Riktning	Typ
Klocksignal 50MHz	Clk_50_in	in	std_logic
Reset	reset_n	in	std_logic
KEY0	IN_KEY_n	in	std_logic
Klock val	Use_Manual_Clock	in	std_logic
Utvald klocka	Clk_out	ut	std_logic

4.5.3 VHDL-koden

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity INPUT_FILTER is
	generic ( cnt_high : integer := 20);
	port (
		Clk_50_in		: in std_logic; -- 50MHz
		reset_n			: in std_logic; -- Reset signal, aktivt låg, SW9 uppåt
		IN_KEY_n		: in std_logic; -- Manuella klockan KEY0 på DE1-SoC
		Use_Manual_Clock	: in std_logic; -- När denna är '1' så IN_KEY_n => Clk_out
	       Clk_out			: out std_logic -- Utgående klockan
	   );

end entity INPUT_FILTER;

architecture rtl of INPUT_FILTER is

	signal flipflops  : std_logic_vector(1 downto 0); -- Detektera förändring på knappen
	signal btn_changed : std_logic;
	signal btn_counter : std_logic_vector(cnt_high downto 0) := (others => '1');
	signal use_Real_Clock : boolean := false;

begin

	btn_changed <= flipflops(0) xor flipflops(1);	-- Har knappen ändrat läge?

	process(Clk_50_in,reset_n)
		variable Clk_Status : std_logic;
	begin
		if (reset_n = '0') then
			btn_counter <= (others => '1');
			use_Real_Clock <= (Use_Manual_Clock = '0');  --Vilken ska användas?
		else
			if rising_edge(Clk_50_in) then
				if not use_Real_Clock then
					flipflops(1) <= flipflops(0);
					flipflops(0) <= IN_KEY_n;
					if (btn_changed = '1') then
						btn_counter <= (others => '0');
					elsif (btn_counter(cnt_high) = '0') then
						btn_counter <= btn_counter + 1;
					else
						Clk_Status := flipflops(1);
						Clk_out <= flipflops(1);
					end if;
				end if;
				use_Real_Clock <= (Use_Manual_Clock = '0');

			end if;
			if use_Real_Clock then
				Clk_Status := Clk_50_in;
			end if;
			Clk_out <= Clk_Status;   -- Lägg ut vår klockstatus på Clk_out

		end if;
	end process;
end rtl;
```

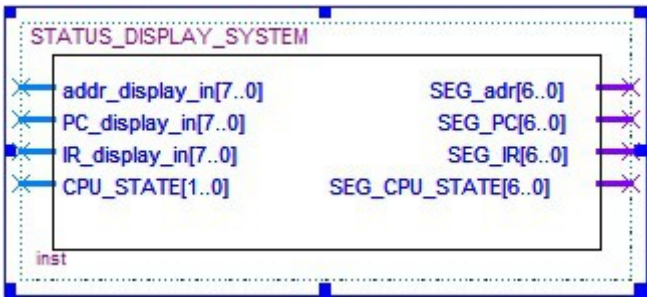
4.6 Status display – komponent

Komponent: STATUS_DISPALY_SYSTEM

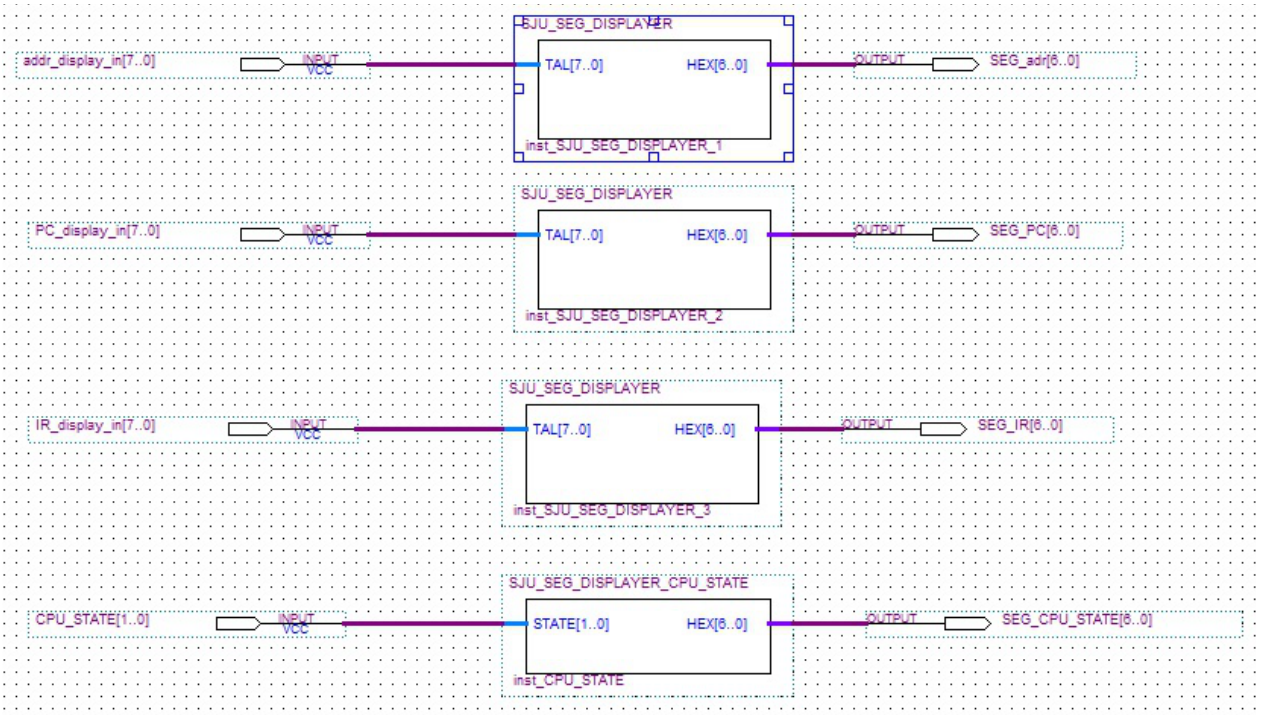
Instansnamn: inst_DISPLAY_SYSTEM

4.6.1 Funktion och arkitektur

Status displaysystemet presenterar adressbussen, programräknaren, instruktionsregistret och CPU:ns state på fyra 7-segmentdisplayer. På DE2-115 kortet finns det, om det inte finns på ett kort så går de inte att koppla in. Status display är uppdelad i fyra olika delsystem. CPU state (SJU_SEG_DISPLAYER_CPU) är en del. De övriga tre är instansieringar av en komponent (SJU_SEG_DISPLAYER). I figur 15 och Figur 16 visas in- och ut-signalerna i status display system. Tabell 11, visar datatyperna för in- och ut-signalerna. Tabell 12 visar vilka 7-segments displayer som är drivna av vilka 7-segments-signaler.



Figur 15. Status display symbol



Figur 16. Status display system subsystem (arkitektur)

4.6.2 In/utgångar

Tabell 11 In- och utsignalerna i status display system

Signal	Namn	Riktning	Typ
Adressbuss	Addr_display_in	in	std_logic_vector(7 downto 0)
Programräknare	PC_display_in	in	std_logic_vector(7 downto 0)
Instruktionsregistret	IR_display_in	in	std_logic_vector(7 downto 0)
CPU-tillstånd (STATE)	CPU_state	in	std_logic_vector(1 downto 0)
Adressbussens värde omtolkat till 7-segmentinformation	SEG_adress	ut	std_logic_vector(6 downto 0)
Programräknarens värde omtolkat till 7-segmentinformation	SEG_PC	ut	std_logic_vector(6 downto 0)
Instruktionsregistrets värde omtolkat till 7-segmentinformation	SEG_IR	ut	std_logic_vector(6 downto 0)
CPU-tillstånd (STATE) värde omtolkat till 7-segmentinformation	SEG_CPU_STATE	ut	std_logic_vector(6 downto 0)

Tabell 12 Information på 7-segmentdisplayerna

Namn	Presenterad information	Display på DE1-SoC
SJU_SEG_DISPLAYER_1	Adressbuss	HEX 0
SJU_SEG_DISPLAYER_2	Programräknare	HEX 1
SJU_SEG_DISPLAYER_3	Instruktionsregistret	HEX 2
CPU_STATE	CPUns tillstånd	HEX 3

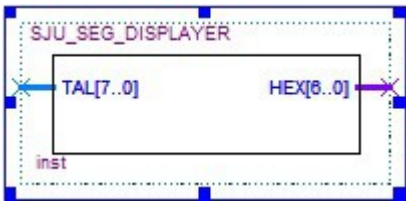
4.7 Sju_seg_displayer – komponent

Komponent: SJU_SEG_DISPLAYER

Instansnamn: inst_SJU_SEG_DISPLAYER_1, inst_SJU_SEG_DISPLAYER_2 och inst_SJU_SEG_DISPLAYER_3

4.7.1 Funktion arkitektur

Komponenten SJU_SEG_DISPLAYER_1 visar adressbussen på 7-segmentdisplayen hex0. Insignalen TAL omtolkas till utsignalen HEX så att 7-segementdisplayen visar TALs värde. Displayen uppdateras då TAL ändrar tillstånd. In- och utsignalerna i sjusegment display 1 visas i figur 17 och tabell 13. RTL-nivån för sjusegmentdisplayen visas i figur 18.



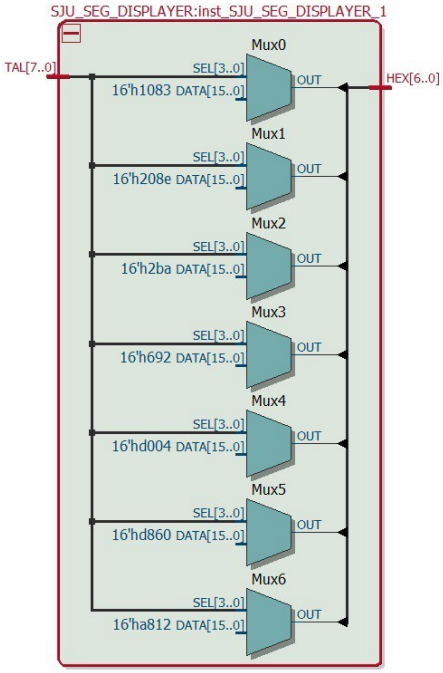
Figur 17. Sjusegment display 1

4.7.2 In/Utgångar

Tabell 13. Signaler för presentation av adressbussen

Signal	Extern namn	Internt namn	Riktning	Typ
Adressbussen	addr_display_in	TAL	in	std_logic_vector(7 downto 0)
Styrdata till HEX6	SEG_adr	HEX	ut	out std_logic_vector(6 downto 0)

4.7.3 RTL-nivå



Figur 18. RTL-nivån för sjusegmentdisplay 1

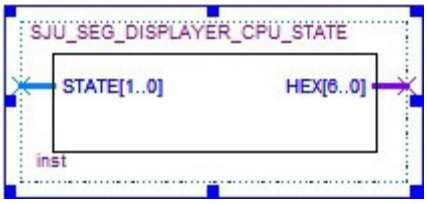
4.8 Sju_seg_displayer_CPU_STATE

Komponent: SJU_SEG_DISPLAYER_CPU_STATE

Instansnamn: inst_CPU_STATE

4.8.1 Funktion arkitektur

Enheten CPU_STATE presenterar datorns tillstånd på 7-segment display HEX3. Displayen uppdateras då STATE ändrar tillstånd. In- och utsignaler för komponenten CPU_STATE visas i figur 19 och tabell 14. Förklaring av text på HEX3 visas i tabell 14. RTL-nivån på CPU_STATE visas i figur?



Figur 19. CPU state

Tabell 14. CPUns tillstånd

Tillstånd	HEX3
FETCH	F
DECODE	D
EXECUTE	E
STORE	S
ERROR	8

4.8.2 In/Utgångar

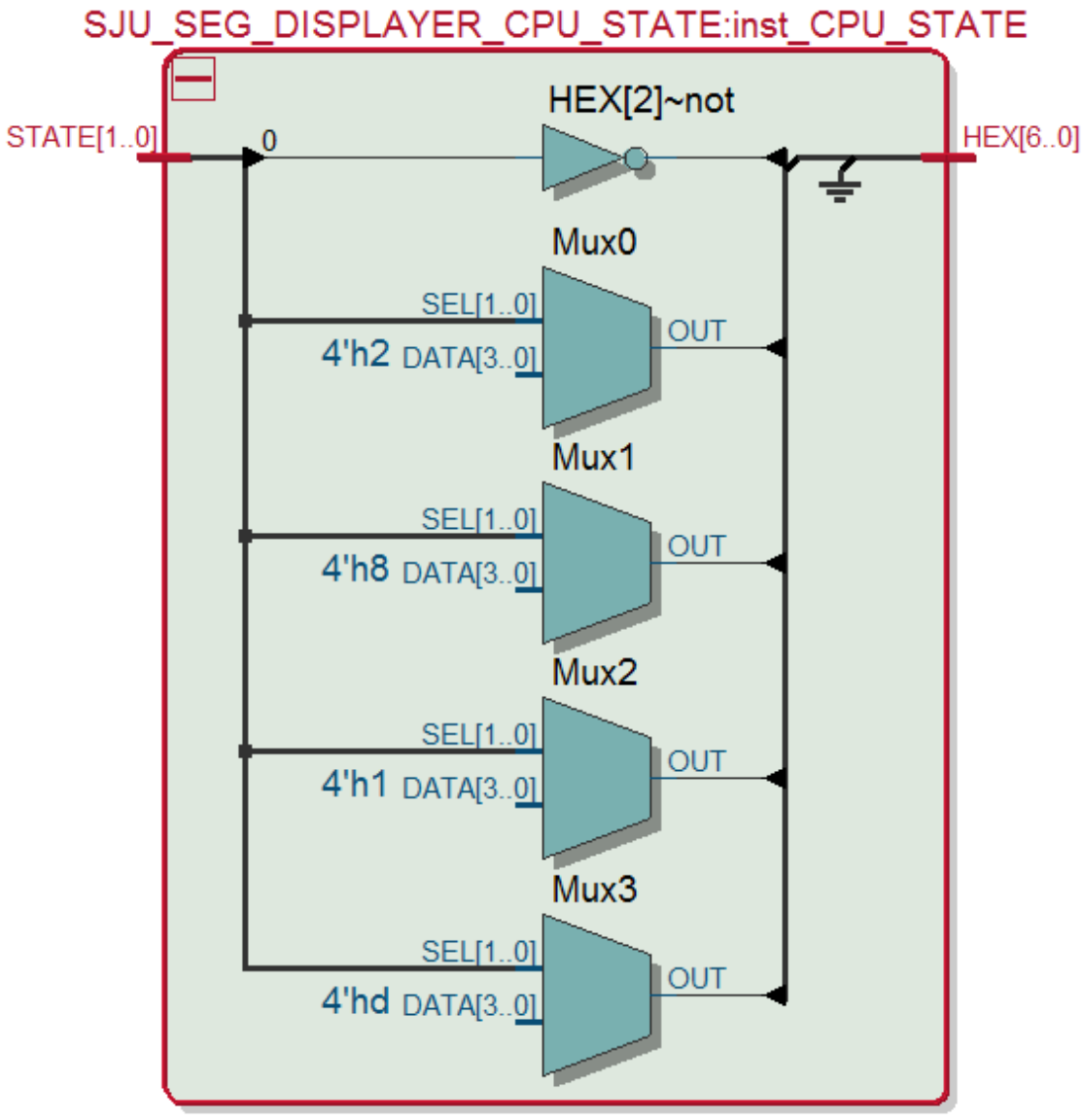
STATE:in std_logic_vector(1 downto 0);-- Tryckknappar

HEX:out std_logic_vector(6 downto 0)

Tabell 15. In och utgångar för SJU_SEG_DISPLAYER_CPU_STATE

Signal	Extern namn	Internt namn	Riktning	Typ
CPU tillstånd	CPU_STATE[1..0]	STATE[1..0]	in	std_logic_vector(1 downto 0)
Styrdata till HEX3	SEG_CPU_STATE	HEX[6..0]	ut	out std_logic_vector(6 downto 0)

4.8.3 RTL-nivå


SJU_SEG_DISPLAYER_CPU_STATE:inst_CPU_STATE
STATE[1..0]
0
HEX[2]~not
HEX[6..0]
Mux0
SEL[1..0]
4'h2 DATA[3..0]
OUT
Mux1
SEL[1..0]
4'h8 DATA[3..0]
OUT
Mux2
SEL[1..0]
4'h1 DATA[3..0]
OUT
Mux3
SEL[1..0]
4'hd DATA[3..0]
OUT

4.8.4 VHDL-kod

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

entity SJU_SEG_DISPLAYER_CPU_STATE is

port( STATE:   in      std_logic_vector(1 downto 0);        -- Tryckknappar
       HEX:    out std_logic_vector(6 downto 0)      -- 7-segment display

               );
end SJU_SEG_DISPLAYER_CPU_STATE;



architecture rtl of SJU_SEG_DISPLAYER_CPU_STATE is

       constant FETCH:              std_logic_vector(6 downto 0)  :=     "0001110";
       constant DECODE:             std_logic_vector(6 downto 0)  :=     "1000000";
       constant EXECUTE:            std_logic_vector(6 downto 0)  :=     "0000110";
       constant STORE:              std_logic_vector(6 downto 0)  :=     "0010010";
       constant ERROR:              std_logic_vector(6 downto 0)  :=     "0000000"; -- alla

-- En etta släcker segmentet en nolla tänder det

begin

       with STATE(1 downto 0) select

       HEX <=  FETCH   when    "00",

                               DECODE  when    "01",

                               EXECUTE when    "10",

                               STORE   when    "11",

                               ERROR   when    others;



end rtl;
```

5 GRANSKNING OCH FÖRSLAG PÅ FÖRBÄTTRINGAR

Följande förslag på förbättringar eller övervägande till förbättringar av VHDL koden.

Komponent:

Simple_VHDL_CPU

1. Slå ihop Execute_Fetch_state, Fetch1, Fetch2, Fetch3 till ett tillstånd
2. Slå ihop Execute_Store_State, Store1_State, Store2_State till ett enda tillstånd

6 FOOT PRINT

Följande meddelar Quartus under ”Fitter Summary”

Family Cyclone V

Device 5CSEMA5F31C6

Timing Models Final

Logic utilization (in ALMs) 55 / 32,070 (< 1 %)

Total registers 83

Total pins 35 / 457 (8 %)

Total virtual pins 0

Total block memory bits 0 / 4,065,280 (0 %)

Total RAM Blocks 0 / 397 (0 %)

Total DSP Blocks 0 / 87 (0 %)

Total HSSI RX PCSs 0

Total HSSI PMA RX Deserializers 0

Total HSSI TX PCSs 0

Total HSSI PMA TX Serializers 0

Total PLLs 0 / 6 (0 %)

Total DLLs 0 / 4 (0 %)

7 KOSTNAD FÖR PROJEKTET

12 timmar à 350 kr blir 4200kr, moms på 25% tilkommer