TEIS
Teknisk rapport

Författare Lasse Karagiannis
Uppgift 7

TEIS ECS – Embedded Computer System

Kontrollerad JA
Version 1

Fil Lasse_Karagiannis_vhdl_uppgift_10.pdf

TEIS AB

TEIS ECS - Embedded Computer System -

Lasse Karagiannis

16-11-20

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.

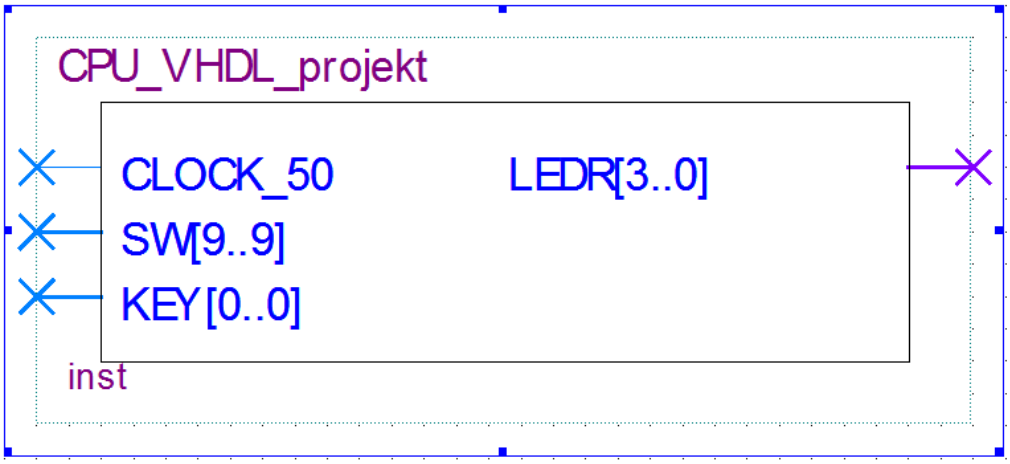
2 KRAVSPECIFIKATION

Denna Uppgiften är att till kund leverera en rapport innehållande analys av givet datorsystem TEIS-ECS, att genomföra en verfiering av tillhandahållen VHDL-kod, där dokumentation av verifieringsprocessen ingår i ovan nämnd rapport, att göra en validering på ett lämpligt FPGA-kort med dokumentation av resultatet i ovan nämnd rapport, samt att ge förslag till förbättringar, också denna presenterad i samma rapport. Vidare krav från kunden är att utrymmet på FPGA:n som använts s.k. footprint anges i rapporten samt att tidsåtgången för arbetet dokumenteras i rapporten.

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



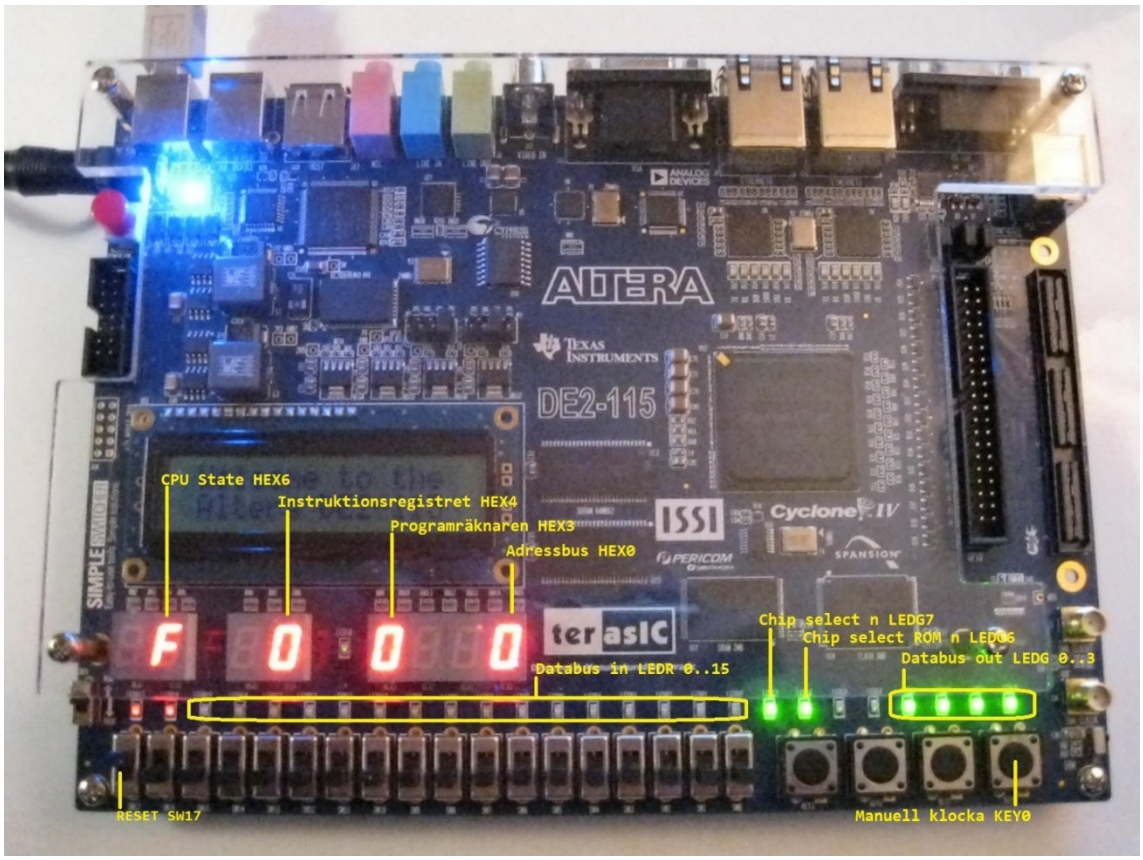
Figur 1. Toppnivån för TEIS mikrodatorsystem

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:

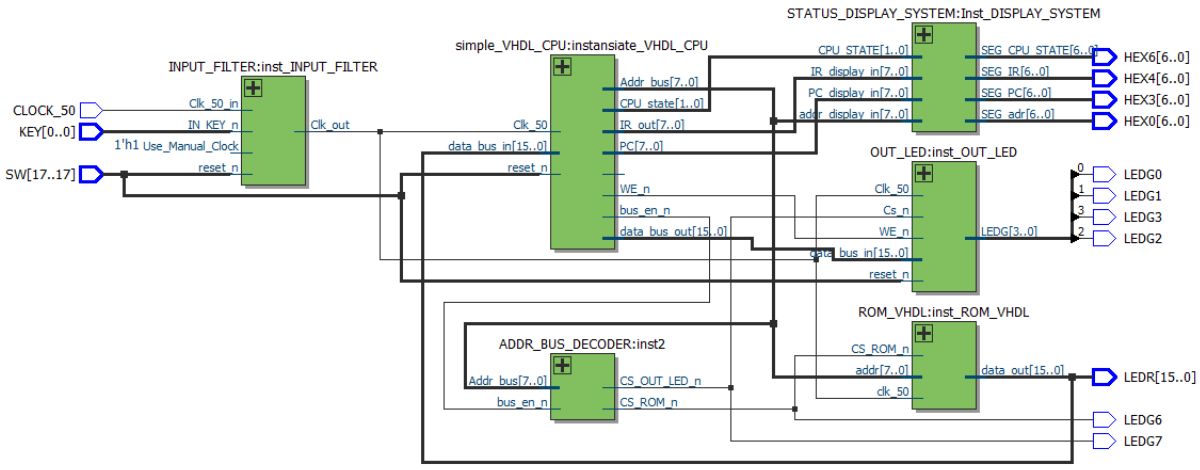
Node Name	Direction	Location
CLOCK_50	Input	PIN_AF14
KEY[0]	Input	PIN_AA14
LEDR[3]	Output	PIN_V18
LEDR[2]	Output	PIN_V17
LEDR[1]	Output	PIN_W16
LEDR[0]	Output	PIN_V16
SW[9]	Input	PIN_AE12

Figur 2. Pinn planner vyn för TEIS mikrodatorsystem

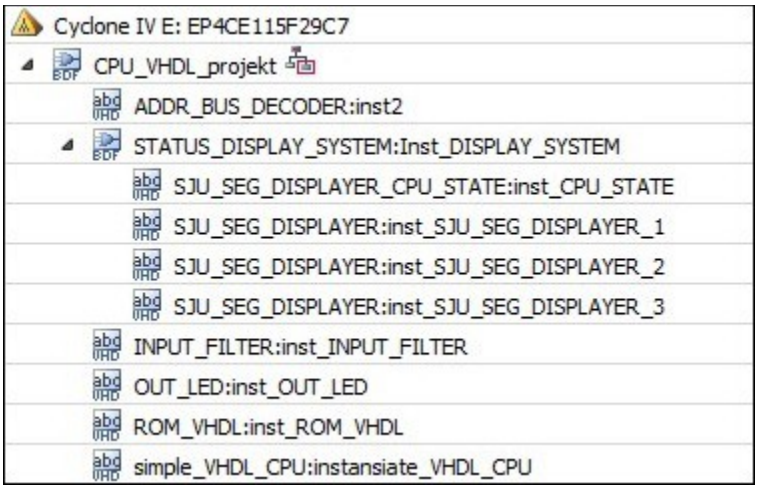


Figur 3. TEIS datorsystem på DE2-115

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 1. Resultat lagras i systemets register.

Tabell 1. Minnesmapp

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

3.4 General package and library

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

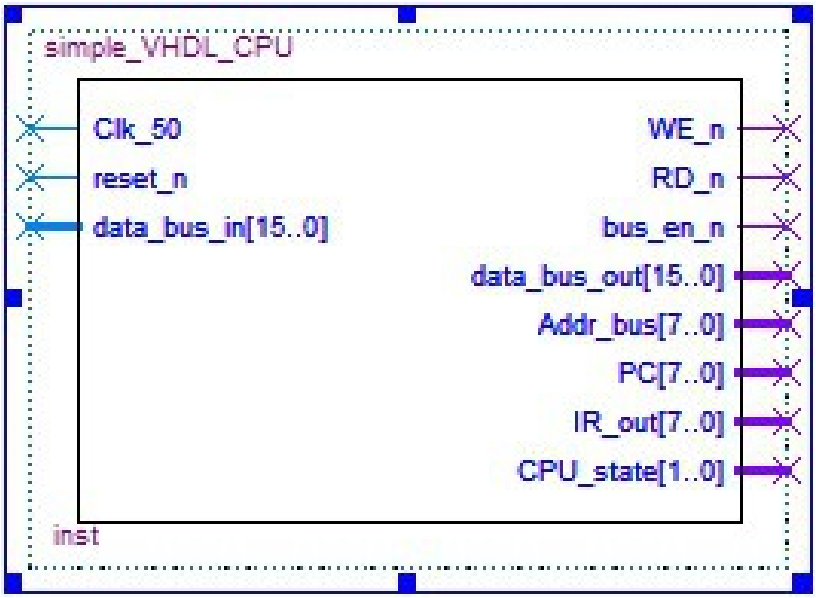
Komponent namn: simple_VHDL_CPU

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

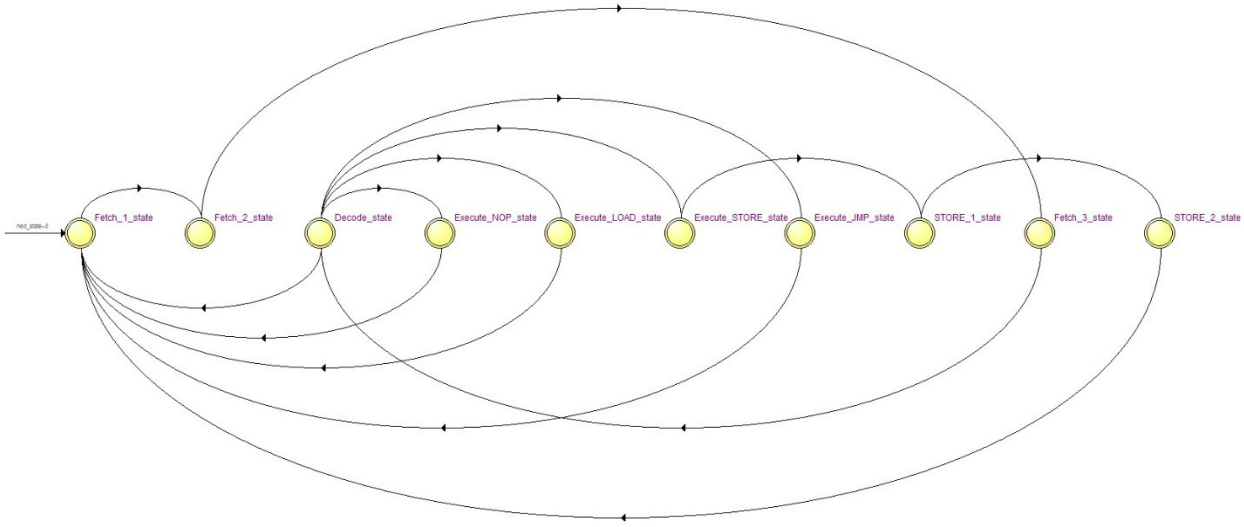
CPU:ns in och utsignaler visas i Tabell 2.

Tabell 2. 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

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 3 för detaljerad beskrivning.

Tabell 3. 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 4, som beskriver de tillgängliga assemblerinstruktionerna.

Tabell 4: Assembler instruktioner, OP-kod och kodning

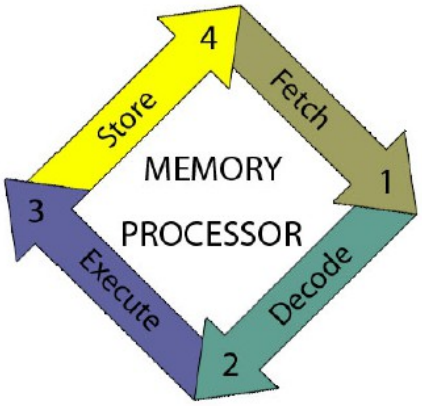
Instruktion	OP-kod	Exempel och förklaring
NOP	0x0---	No operation
LOAD_R0 #imm	0x1nnn	Load R0 with immediate
STORE_R0#ADDR	0x2-nn	Store R0 at 8-bit adress
JMP #ADDR	0x3-nn	Load IR content of code memory at adress nn

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

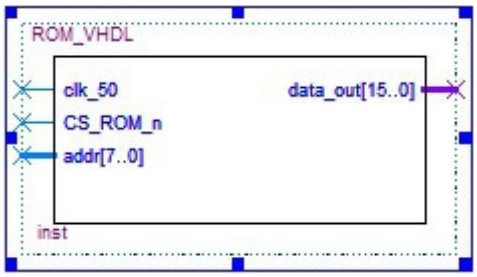
4.2 ROM – komponent

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 5. RTL-nivån på ROM visas i Figur 9.



Figur 9. ROM Symbol

Tabell 5. 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

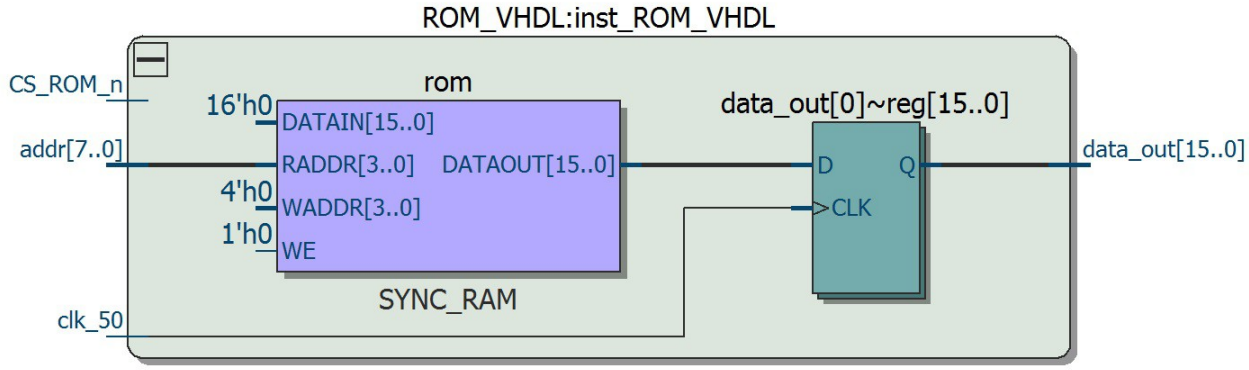
4.2.2 In/utgångar

In- och utsignalerna till ROM enligt Tabell 6.

Tabell 6. 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;
```

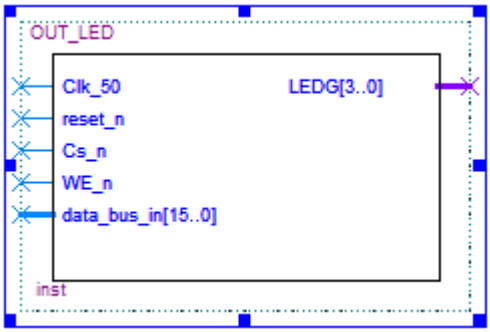
4.3 LED – komponent

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 11 ger en schematisk bild av komponentens in och utgångar. Tabell 7 beskriver komponentens in och utgångar typer. Figur 12 visar RTL implementationen av VHDL-koden som listas i avsnitt 4.3.4.



Figur 11 OUT_LED

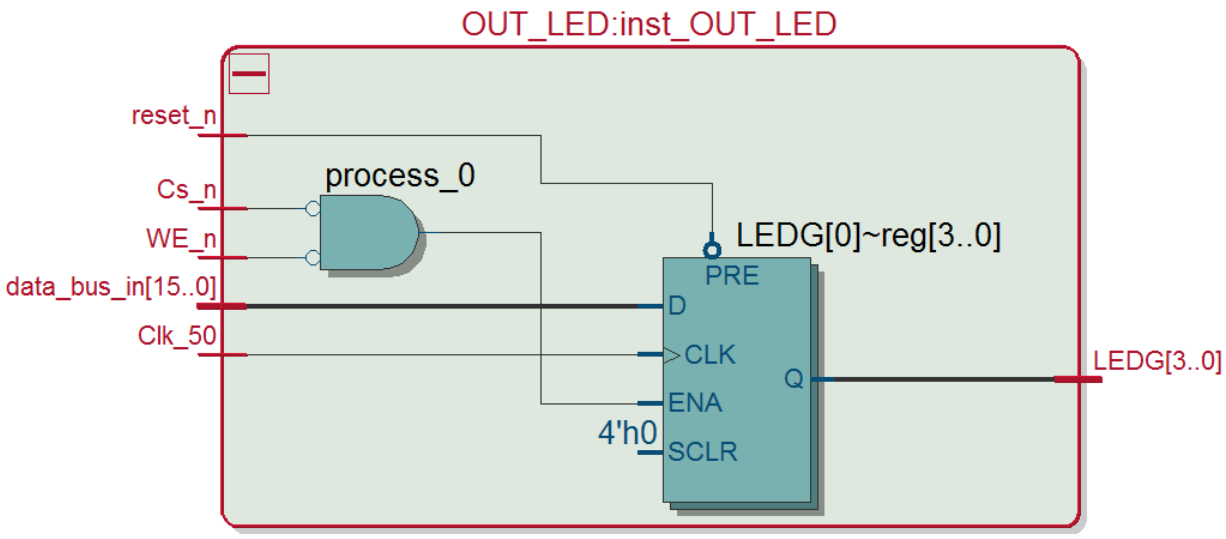
4.3.2 In/utgångar

In och utsignaler till LED_OUT enligt Tabell 7

Tabell 7 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 12 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;
```

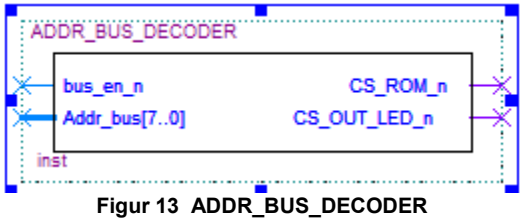
4.4 Adressbussdekoder – komponent

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 13 ger en schematisk bild av komponenten. Tabell 8 innehåller in och utgångarnas typer. Figur 14 visar RTL-nivån.



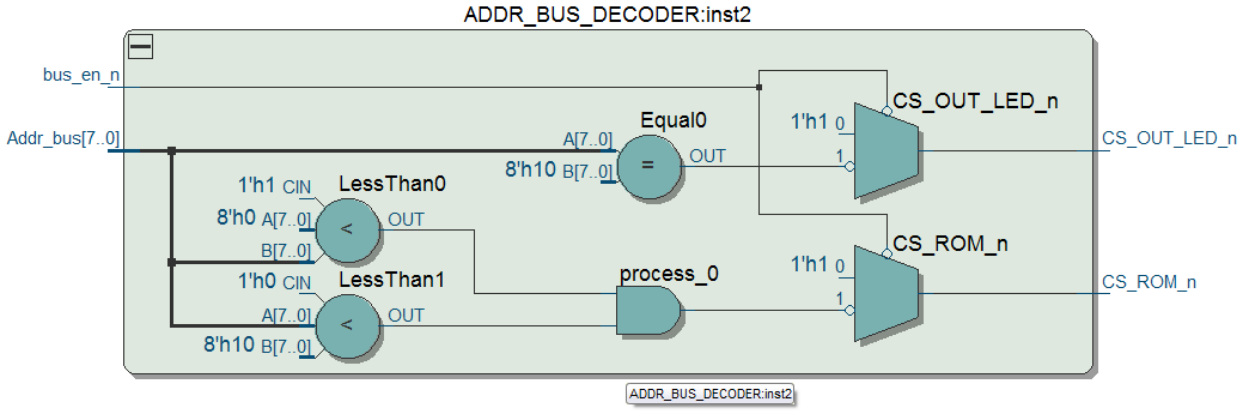
Figur 13 ADDR_BUS_DECODER

4.4.2 In/utgångar

Tabell 8 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 14 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

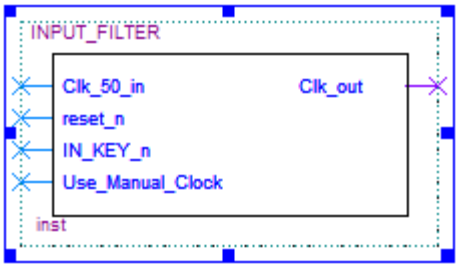
Komponent: INPUT_FILTER

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 15 ger en schematisk vy över in och ut -portarna. Tabell 9 listar typerna för ingångarna och utgångarna.



Figur 14 INPUT_FILTER

4.5.2 In och utgångar

Tabell 9 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;
```

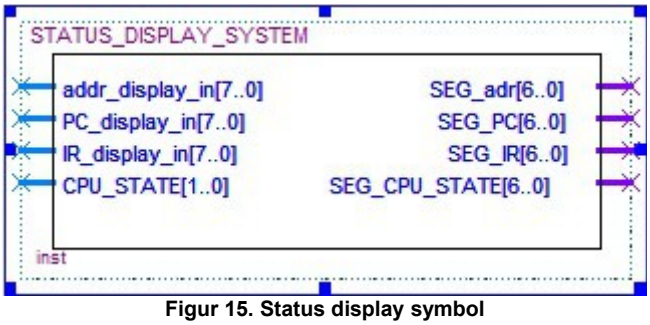
4.6 Status display – komponent

Komponent: STATUS_DISPALY_SYSTEM

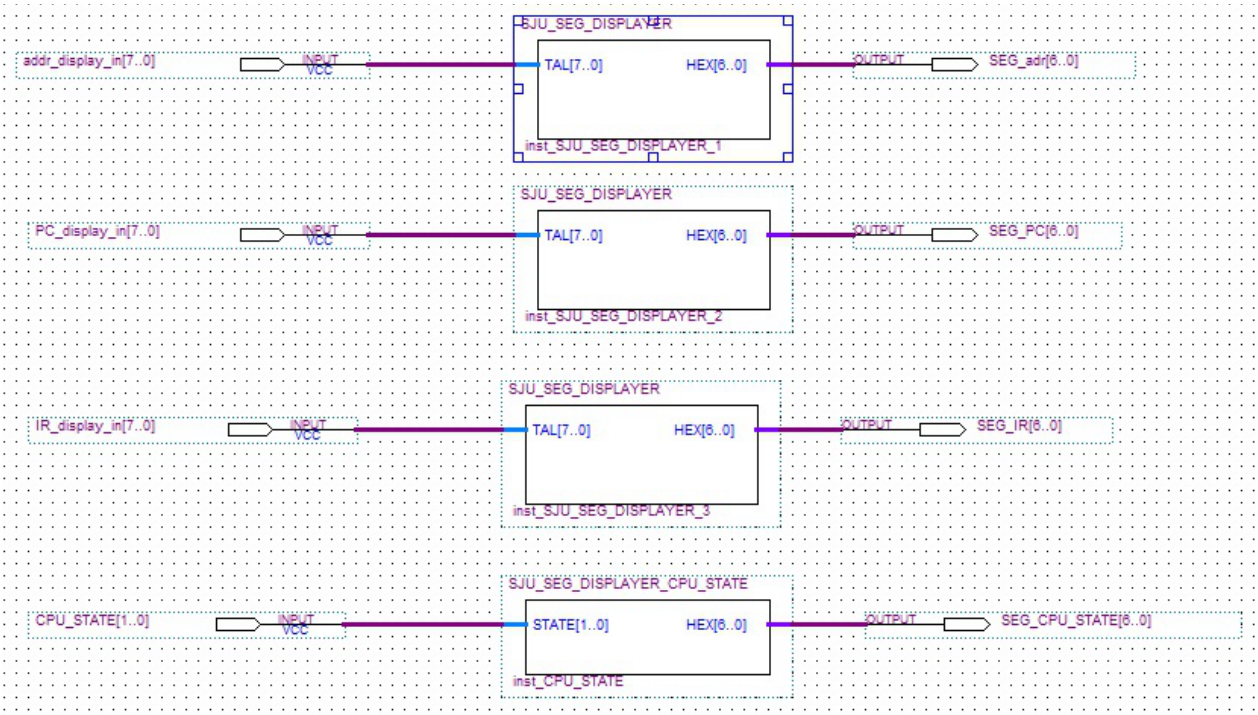
Instansnamn: inst_DISPLAY_SYSTEM

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 10, visar datatyperna för in- och ut-signalerna. Tabell 11 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 10 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 11 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

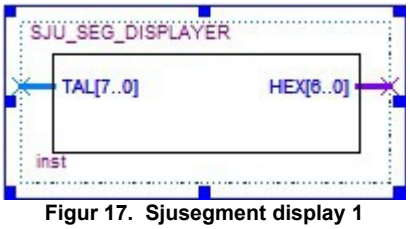
4.7 Sju_seg_displayer – komponent

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.



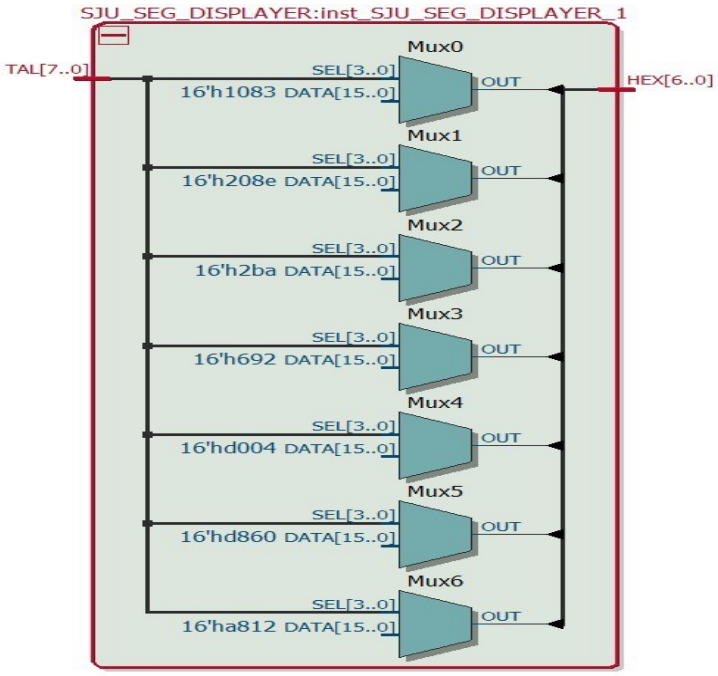
Figur 17. Sjusegment display 1

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

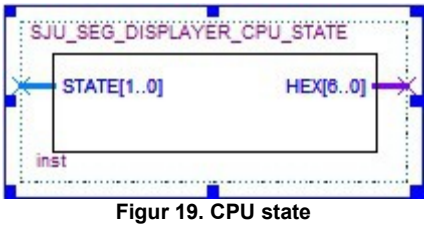
4.8 Sju_seg_displayer_CPU_STATE

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 Figur20



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

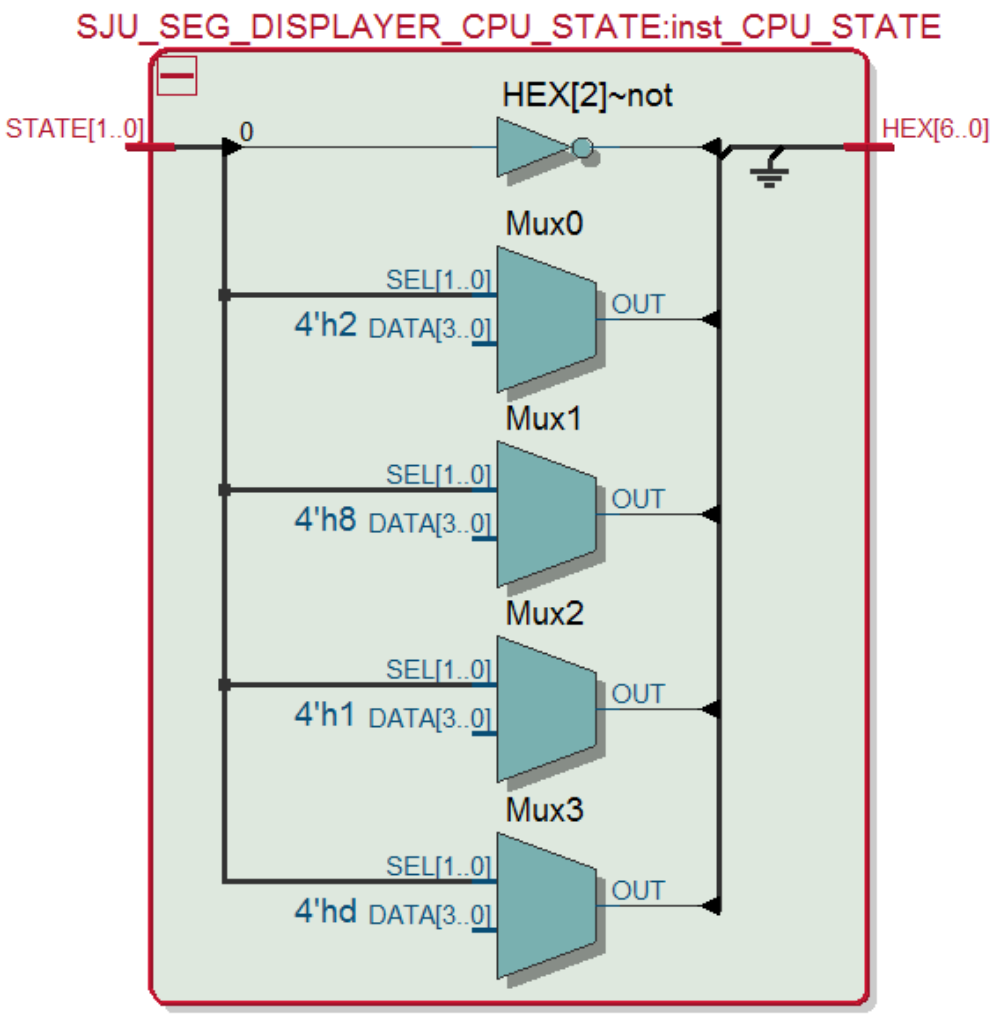
STATE:in std_logic_vector(1 downto 0);-- Tryckknappar

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



Figur 20. RTL-nivån för CPU state

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 Verifiering

Verifiering med Modelsim har gjorts av enkel CPU. 50 instruktionscykel har stegats igenom, och värdet i CPU:ns register, adress och data bussar, samt lysdioderna status har analyserats.
Registervärden och signalvärden på bussar har jämförts med den givna VHDL-koden. Tabell 16 I avsnitt 5.1 Tabell med testfall visar ett antal testfall som undersökts.

5.1 Tabell med testfall

Tabell 16. Tabell med testfall

Testfall	Insignal	Utsignal	OK/Fel
Resetsignal som återställer konstruktionen till utgångsläget Reset_n <= ’0’	reset_n <= '0'	Nollställs PC, IR, R0, adressbuss och datautgång. Kontrollsignalerna sätts i inaktivt läge och tillstånd till Fetch_state	OK
NOP-instruktion	IR<=X"0000"	next_state = Execute_NOP_state	OK
LOAD-instruktion LOAD_R0 #A	IR< =X"100A"	CPU_REG_0 = ”0xA” under Execute_Load-state	OK
STORE-instruktion STORE_R0 #10	IR <= X"2010"	Addr_bus = 0b00010000 WE_n = '0' data_bus_out = REG_0 då next_state = STORE_2	OK
JMP-instruktion JMP #1	IR <= X"3001"	PC_reg <= IR[7:0] next_state <= Fetch_1_state;	OK, då next_state = EXECUTE_JMP

5.2 Beskrivning av testbänken

Testbänken innehåller ingenting annat än en uppdatering av klocksignalen, en reset-nivåändring från låg till hög. Då reset_n går hög kommer processorns state-machine att börja hämta instruktioner från minnet.

```
clock : PROCESS

-- variable declarations

BEGIN

    CLOCK_50 <= '0';

                wait for sys_clk_period/2;

                CLOCK_50 <= '1';

                wait for sys_clk_period/2;

END PROCESS clock;

SW(9) <= '0', '1' after 10*sys_clk_period;
```

5.3 Do-filens beskrivining

Do-filen återfinns under mappen simulation, och skapas då testbänken genereras av verktyget.
Man kan lägga till signaler, genom att klicka på projektstrukturen (fönster med vit bakgrund) och expandera en trädstrukturen för en subkomponent som man vill studera. Då komponenter i hierarkin markeras så presenteras tillgängliga signaler i fönstret med blå bakgrund. Man höger klickar på dessa och väljer ”Add Wave”. Signalen läggs till i simuleringsfönstret, genom att en kodsnutt körs i terminalen. Vill man simulera de manuellt tillagda signalerna från start av simuleringen, tillsammans med de automatiskt valda signalerna, så måste do-filen uppdateras med de kod-snuttar som verktyget genererade och körde genom terminalen då du lade till ytterliggare signaler.

```
add wave -position insertpoint  \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/reset_n

add wave -position insertpoint  \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/PC_reg

add wave -position insertpoint  \
```

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/Addr_bus

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/bus_en_n

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/RD_n

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/data_bus_in

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/IR

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/CPU_REG_0

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/next_state

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/WE_n

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_instansiate_VHDL_CPU/data_bus_out

add wave -position insertpoint \

sim:/cpu_vhdl_projekt_vhd_tst/i1/b2v_inst_OUT_LED/LEDG

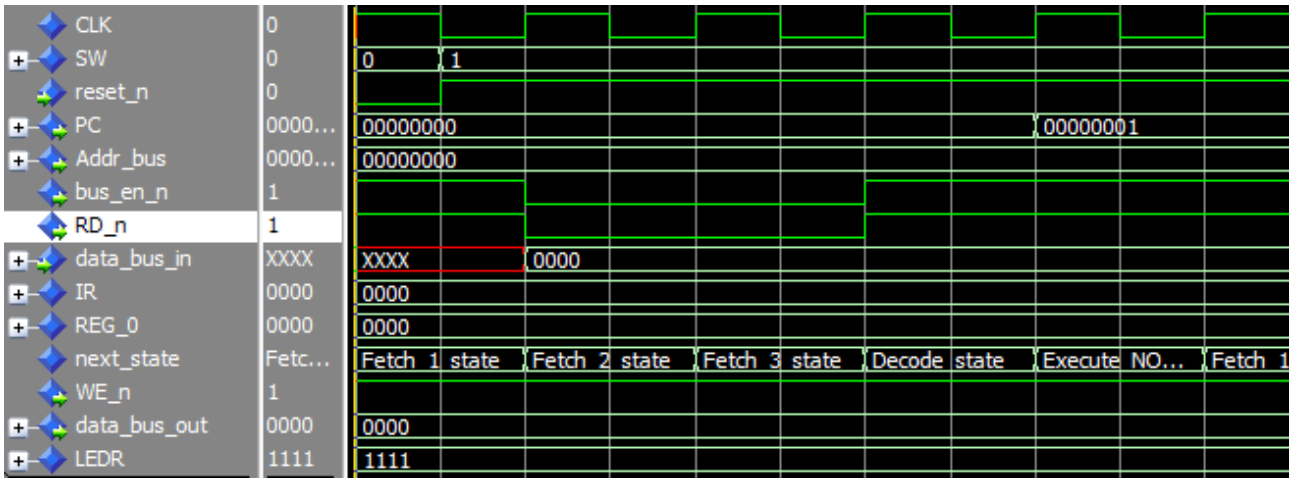
Dessa rader lades till efter raden med instruktionen *add wave **.
För att få ett kortare signalnamn så högerklickar man på signalen i fönstret för simuleringen och väljer ”Properties”. Där kan man skriva in ett alternativt kortare namn som visas i simuleringsfönstret.

5.4 Resultatet från test

Resultatet från testerna visas i kapitel 5.5 Tidsanalys nedan

5.5 Tidsanalys

5.5.1 Cykler 1-14



Figur 21 Cykler 1-14, simulering Modelsim

Tabell 17. Signaler och registervärden cykler 1-14

CLK_PERIOD	1 to 10	11	12	13	14
SW	0	1	1	1	1
reset_n	0	1	1	1	1
PC	0b00000000	0b00000000	0b00000000	0b00000000	0b00000001
Addr_bus	0b00000000	0b00000000	0b00000000	0b00000000	0b00000000
bus_en_n	1	0	0	1	1
RD_n	1	0	0	1	1
data_bus_in	0xUUUU	0x0000	0x0000	0x0000	0x0000
IR	0x0000	0x0000	0x0000	0x0000	0x0000
REG_0	0x0000	0x0000	0x0000	0x0000	0x0000
next_state	Fetch_1_state	Fetch_2_state	Fetch_3_state	Decode	Execute_NOP
WE_n	1	1	1	1	1
data_bus_out	0x0000	0x0000	0x0000	0x0000	0x0000
LEDR	0b1111	0b01111	0b1111	0b1111	0b1111

Fetch_1_state ligger som next_state fram till att reset_n går går hög, samtliga register är nollställda under reset-fasen, vi har nämligen kod (klistrar in en del av den)

```
process(reset_n, clk_50)

begin

if reset_n = '0' then
        PC_reg <= X"00"; -- after reset, PC is zero (i.e address 0)
        IR <= X"0000"; -- NOP, no operation
        Addr_bus <= X"00" ; -- initialize registers
        CPU_REG_0 <= (others => '0'); -- initialize registers
        next_state <= Fetch_1_state;
        bus_en_n <= '1';
        data_bus_out <= "0000000000000000"; -- initialize registers
        WE_n <= '1';
        RD_n <= '1';
        CPU_state <= "00";
elsif(…...
```

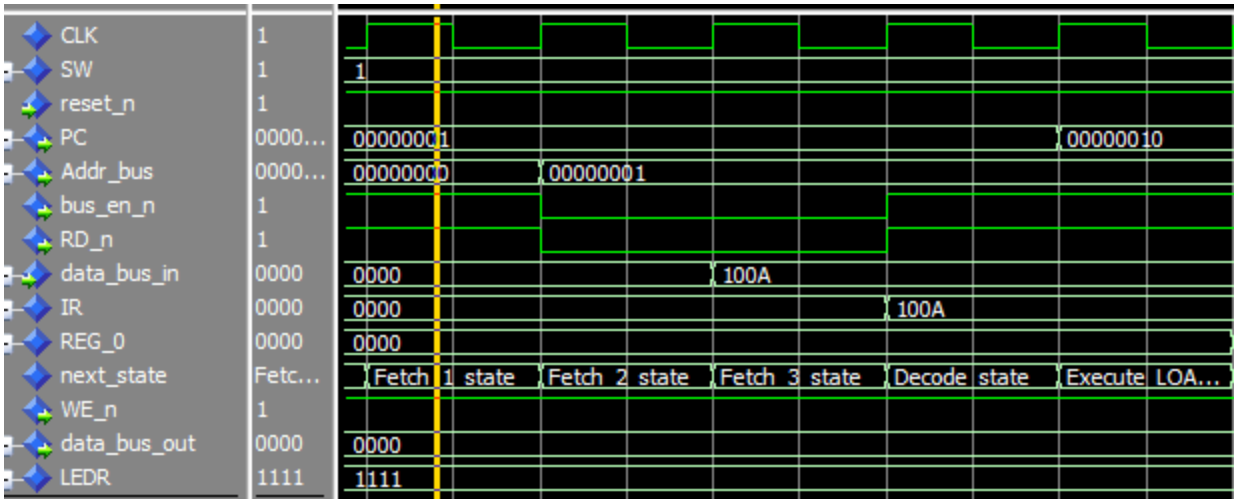
Lysdioderna lyser p.g.a. koden i OUT_LED

```
process(reset_n, clk_50)

begin
      if reset_n = '0' then
            LEDG <= "1111";-- LED-signals go high during reset
      elsif(....
```

Instruktionsläsningen kräver 2 cykler, där IR har laddats med instruktionskoden för NOP 0x0000 och är giltlig under cykel 14.

5.5. 2 Cykler 15-19



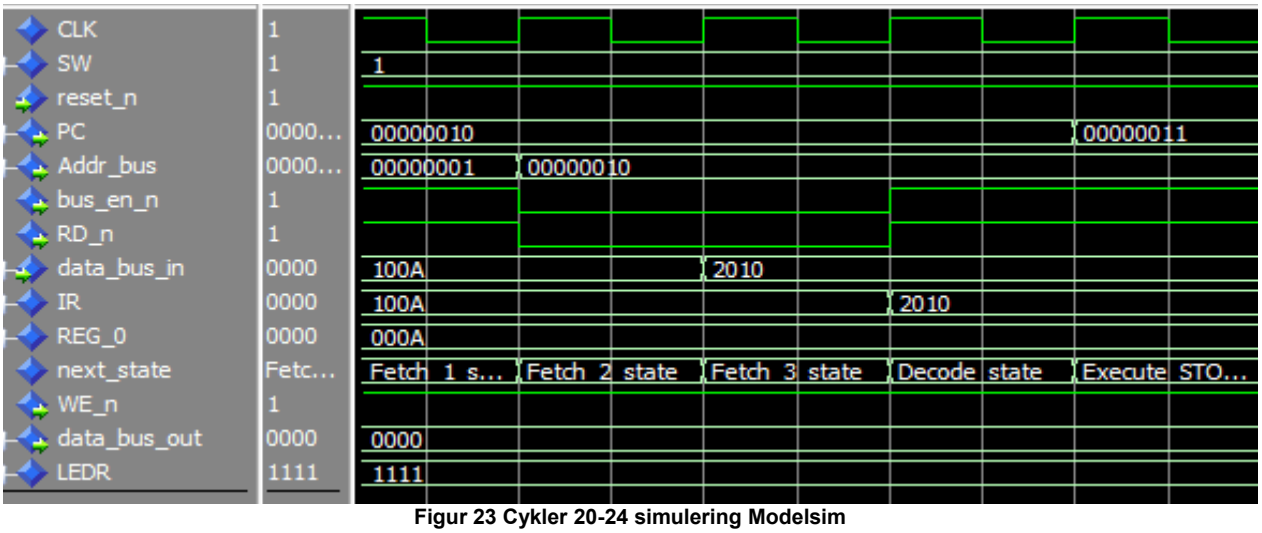
Figur 22 Cykler 15-19 simulering Modelsim

Tabell 18. Signaler och registervärden cykler 15-19

CLK_PERIOD	15	16	17	18	19
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000001	0b00000001	0b00000001	0b00000001	0b00000010
Addr_bus	0b00000000	0b00000001	0b00000001	0b00000001	0b00000001
bus_en_n	1	0	0	1	1
RD_n	1	0	0	1	1
data_bus_in	0x0000	0x0000	0x100A	0x100A	0x100A
IR	0x0000	0x0000	0x0000	0x100A	0x100A
REG_0	0x0000	0x0000	0x0000	0x0000	0x0000
next_state	Fetch_1	Fetch_2	Fetch_3	Decode	Execute_LOAD
WE_n	1	1	1	1	1
data_bus_out	0x0000	0x0000	0x0000	0x0000	0x0000
LEDR	0b1111	0b1111	0b1111	0b1111	0b1111

Här hämtas och exekveras intruktionen LOAD_R0 #A med op-kod 0x100A på adressposition 1 i kod-minnet. PC inkrementerades redan under exekveringsfasen av NOP, under cykel 14. Addr_bus kan inte adressera kod-minnet under decode-fasen av NOP, eftersom samma adressbuss används för både kod och data.
En hel cykel används (16) till att låta RD_n och bus_en_n bli stabil, och koppla PC till adressbussen. Nästa cykel (17) liger kod-data på kod-bussen, för att läsas in till IR cykeln därpå. Man borde kunna slå ihop cykel 16 och 17, dvs. att RD_n och bus_en_n bara får vara låga under en cykel och IR läses samtidigt som kod ligger stabil på kodbussen utan något mellansteg.

5.5.3 Cykler 20 – 24



Figur 23 Cykler 20-24 simulering Modelsim

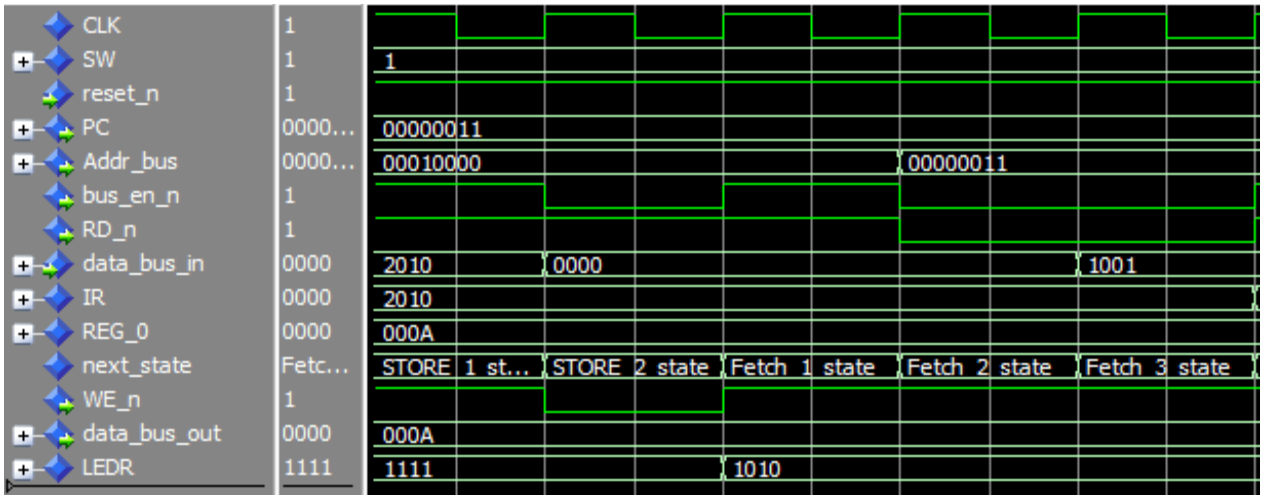
Tabell 19. Signaler och registervärden cykler 20-24

CLK_PERIOD	20	21	22	23	24
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000010	0b00000010	0b00000010	0b00000010	0b00000011
Addr_bus	0b00000001	0b00000010	0b00000010	0b00000010	0b00000010
bus_en_n	1	0	0	1	1
RD_n	1	0	0	1	1
data_bus_in	0x100A	0x100A	0x2010	0x2010	0x2010
IR	0x100A	0x100A	0x100A	0x2010	0x2010
REG_0	0x000A	0x000A	0x000A	0x000A	0x000A
next_state	Fetch_1	Fetch_2	Fetch_3	Decode:xecute_STORE	
WE_n	1	1	1	1	1
data_bus_out	0x0000	0x0000	0x0000	0x0000	0x0000
LEDR	0b1111	0b1111	0b1111	0b1111	0b1111

Under cykel 20 sker exekvering av LOAD instruktionen där R0 laddas med argumentet 0xA. PC inkrementerades under decode-fasen av LOAD_R0 #A. Nästa cykel (21) är Fetch_1, och då är PC kopplad till adressbusesen, och då bus_en_n och RD_n går från låg till hög laddas IR med op-koden

0x2010, vars mnemonic är STORE_R0 #10. Under decode (cykel 24) inkrementeras PC återigen.

5.5.4 Cykler 25-29



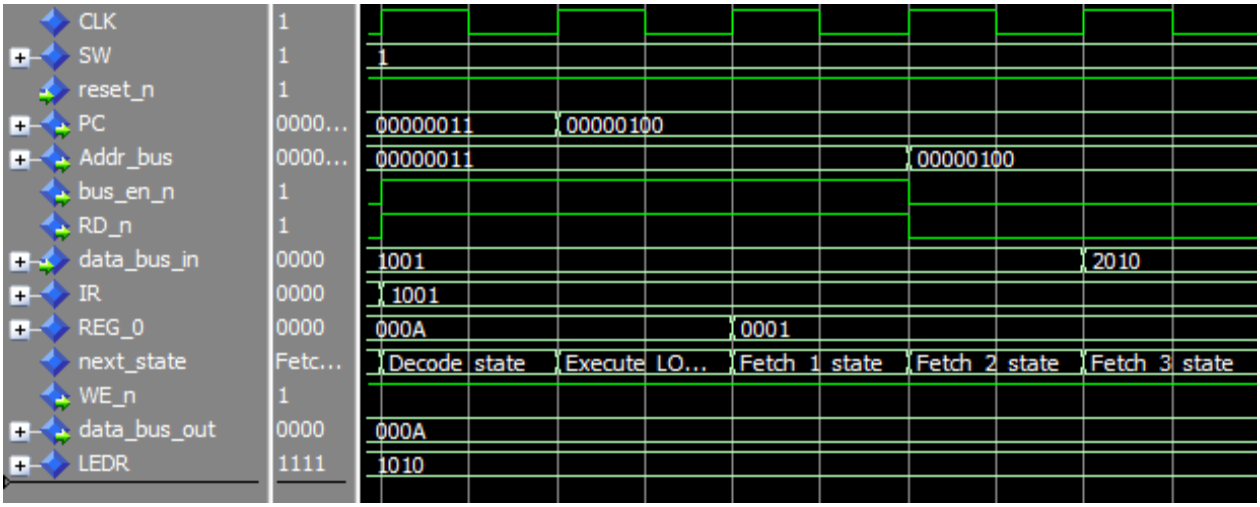
Figur 24 Cykler 25-29 simulering Modelsim

Tabell 20. Signaler och registervärden cykler 25-29

CLK_PERIOD	25	26	27	28	29
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000011	0b00000011	0b00000011	0b00000011	0b00000011
Addr_bus	0b00010000	0b00010000	0b00010000	0b00000011	0b00000011
bus_en_n	1	0	1	0	0
RD_n	1	1	1	0	0
data_bus_in	0x2010	0x0000	0x0000	0x0000	0x1001
IR	0x2010	0x2010	0x2010	0x2010	0x2010
REG_0	0x000A	0x000A	0x000A	0x000A	0x000A
next_state	STORE_1	STORE_2	Fetch_1	Fetch_2	Fetch_3
WE_n	1	0	1	1	1
data_bus_out	0x000A	0x000A	0x000A	0x000A	0x000A
LEDR	0b1111	0b1111	0b1010	0b1010	0b1010

Under Execute_STORE (cykel 25) kopplas IR:s argument 0x10 till Adressbussen och Reg_0 läggs ut på data_bussen. Under cykel 26 görs data giltlig genom att WE_n går låg. Senast cykel 27 lyser dioderna enligt bitmönstret 0b1010 (0xA).
PC har varit inkrementerad sedan flera klockcykler tidigare. Under Fetch_1 (cykel 28) koppas PC ut till Adressbssen och bus_en_n och RD_n går då återigen samtidigt låga. IR kommer laddas under Fetch_3 (cykel 30, se nästa avsnitt).

5.5.5 Cykler 30 – 34



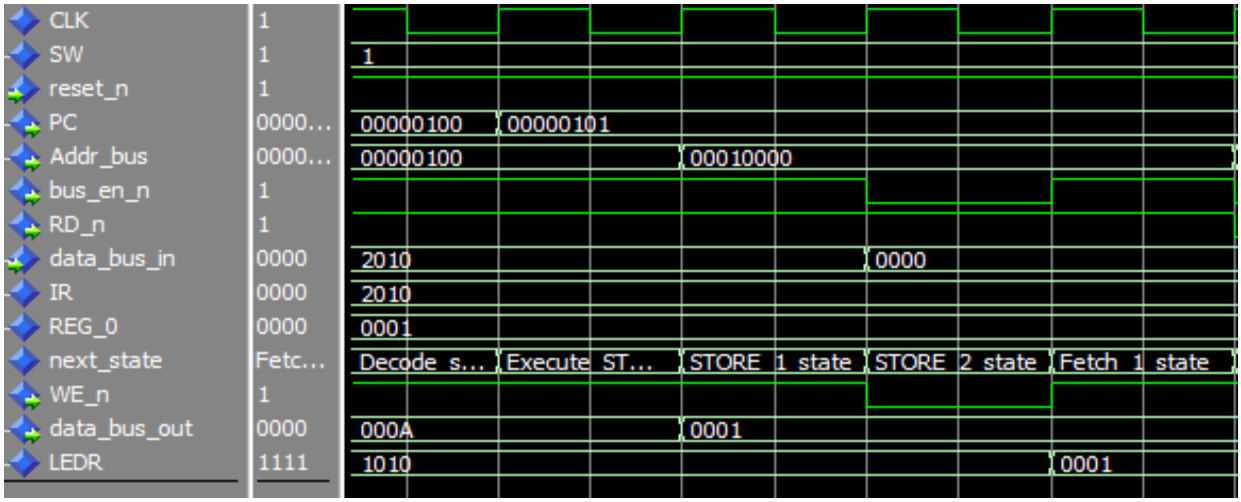
Figur 25 Cykler 30-34 simulering Modelsim

Tabell 21. Signaler och registervärden cykler 30-34

CLK_PERIOD	30	31	32	33	34
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000011	0b00000100	0b00000100	0b00000100	0b000001000
Addr_bus	0b00000011	0b00000011	0b00000011	0b00000100	0b000001000
bus_en_n	1	1	1	0	0
RD_n	1	1	1	0	0
data_bus_in	0x1001	0x1001	0x1001	0x1001	0x2010
IR	0x1001	0x1001	0x1001	0x1001	0x1001
REG_0	0x000A	0x000A	0x0001	0x0001	0x0001
next_state	Decode	Execute_LOAD	Fetch_1	Fetch_2	Fetch_3
WE_n	1	1	1	1	1
data_bus_out	0x000A	0x000A	0x000A	0x000A	0x000A
LEDR	0b1010	0b1010	0b1010	0b1010	0b1010

Op-koden som IR håller är 0x1001 vars mnemonic är LOAD_R0 #1. Under decode (cykel 31) inkrementeras PC. Under Execute_LOAD (cykel 32) laddas Reg_0 med OP-kodens argument. Under Fetch_1 (cykel 33) kopplas PC till adressbussen bus_en_n och RD_n går då låga. Nästa instruktion som är resultatet av adresseringen av kod-minnet läggs ut på kod-bussen under cykel 34, och läses in till IR under cykel 36.

5.5.6 Cykler 35-39



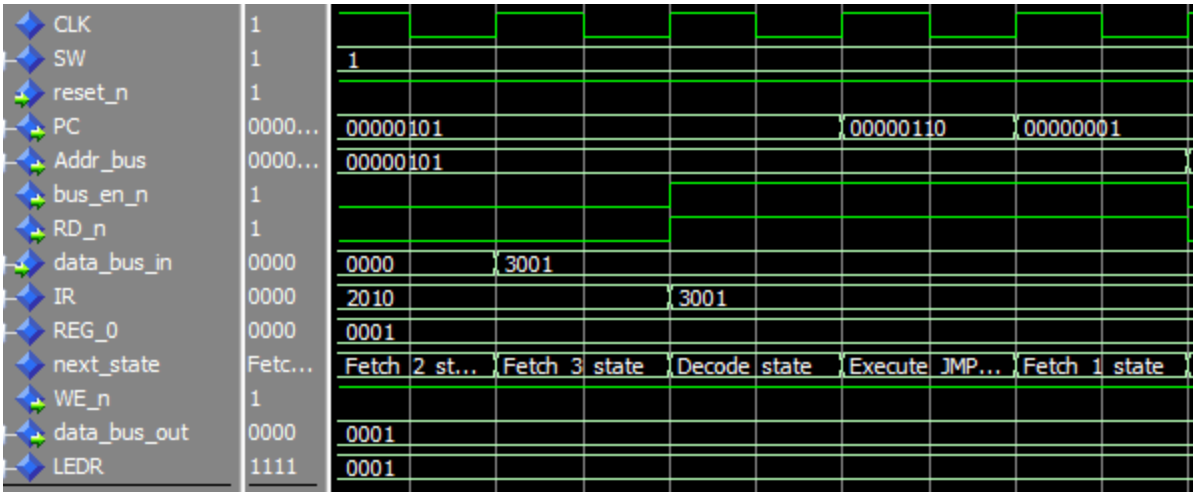
Figur 25 Cykler 30-34 simulering Modelsim

Tabell 22. Signaler och registervärden cykler 35-39

CLK_PERIOD	35	36	37	38	39
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000100	0b00000101	0b00000101	0b00000101	0b00000101
Addr_bus	0b00000100	0b00000100	0b00010000	0b00010000	0b00010000
bus_en_n	1	1	1	0	1
RD_n	1	1	1	1	1
data_bus_in	0x2010	0x2010	0x2010	0x0000	0x0000
IR	0x2010	0x2010	0x2010	0x2010	0x2010
REG_0	0x0001	0x0001	0x0001	0x0001	0x0001
next_state	Decode	Execute_STORE	STORE_1	STORE_2	Fetch_1
WE_n	1	1	1	0	1
data_bus_out	0x000A	0x000A	0x0001	0x0001	0x0001
LEDR	0b1010	0b1010	0b1010	0b1010	0b0001

Inläsningen till IR innebär att op-koden 0x2010, vilket är menomic för STORE_R0 #10.
Under deode inkrementeras PC (cykel 36), och Execute_STORE görs under cykel 37. Då kopplas IR:s argument till adressbussen och REG_0 läggs ut på databussen. WE_n går låg under följande cykel (38) och senast cykel 39 lyser lysdioderna i enlighet med vad som lades ut på databussen dvs. 0b0001.

5.5.7 Cykler 40 – 44



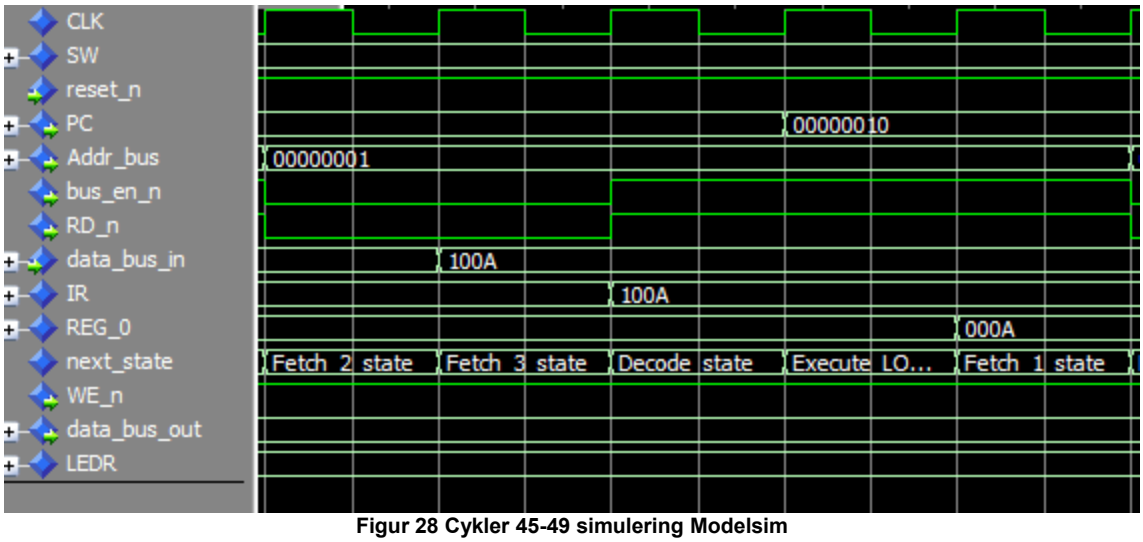
Figur 27 Cykler 40-44 simulering Modelsim

Tabell 23. Signaler och registervärden cykler 40-44

CLK_PERIOD	40	41	42	43	44
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000101	0b00000101	0b00000101	0b00000110	0b00000001
Addr_bus	0b00000101	0b00000101	0b00000101	0b00000101	0b00000101
bus_en_n	0	0	1	1	1
RD_n	0	0	1	1	1
data_bus_in	0x0000	0x3001	0x3001	0x3001	0x3001
IR	0x2010	0x2010	0x3001	0x3001	0x3001
REG_0	0x0001	0x0001	0x0001	0x0001	0x0001
next_state	Fetch_2	Fetch_3	Decode	Execute_JMP	Fetch_1
WE_n	1	1	1	1	1
data_bus_out	0x0001	0x0001	0x0001	0x0001	0x0001
LEDR	0b0001	0b0001	0b0001	0b0001	0b0001

Under Fetch_1 (cykel 40) kopplas PC till adressbussen och bus_en_n och RD_n går båda låga.
Nästa instruktion, dvs. den på adress adresserad av PC finns på kodbussen under cykel 41, och inläsning till IR sker under nästa cykel (42). Op koden är 0x3001 vars mnemonic är JMP #1.
Under decode inkrementeras PC (cykel 43), men under Execute_JMP (cykel 44) laddas PC med instruktionens argument dvs. IR:s argument-del kopplas till PC.

5.5.8 Cykler 45-49



Figur 28 Cykler 45-49 simulering Modelsim

Tabell 24. Signaler och registervärden cykler 45-49

CLK_PERIOD	45	46	47	48	49
SW	1	1	1	1	1
reset_n	1	1	1	1	1
PC	0b00000001	0b00000001	0b00000001	0b00000010	0b00000010
Addr_bus	0b00000001	0b00000001	0b00000001	0b00000001	0b00000001
bus_en_n	0	0	1	1	1
RD_n	0	0	1	1	1
data_bus_in	0x3001	0x100A	0x100A	0x100A	0x100A
IR	0x3001	0x3001	0x100A	0x100A	0x0000
REG_0	0x0001	0x0001	0x0001	0x0001	0x000A
next_state	Fetch_2	Fetch_3	Decode	Execute_LOAD	Fetch_1
WE_n	1	1	1	1	1
data_bus_out	0x0001	0x0001	0x0001	0x0001	0x0001
LEDR	0b0001	0b0001	0b0001	0b0001	0b0001

Under Fetch_1 (cykel 45) kopplas PC till Adressbussen och bus_en_n och RD_n går båda låga samtidigt. Instrutkionen som läses in till IR (cykel 47) är från adress 1 i kodminnet LOAD_R0 #A med OP-kod 0x100A, som exekverats förut.
Under decode (cykel 48) inkrementeras PC, och under Execute_LOAD (cykel 49) har Reg_0 laddats med argument delen av IR.

6 Validering

Validering har gjorts i två delar såsom kravspecifikationen föreskrivit. Enligt kravspecifikationen skall man

”Verifiera konstruktionen med out LED och ett varv i programmet. Endast ett testfall och det visar antal klocktryckningar och out_led. Detta är kundens acceptans på att konstruktionen fungerar.”

Denna första del utökades med att anteckna sjusegmentsdisplayernas värden, som fördes upp i en tabell tillsammans med ledsignalernas värden, som funktion av tangenttryckningar för att åstadkomma manuell klockning av CPU:n.

6.1 Test Cases Tabellen test cases med loggade data

I Tabell 25 nedan finns kortets förevisade värden avseende de fyra sjusegmentsdisplayerna samt de 4 röda lysdioderna rad för rad såsom för respektive tangentnedtryckning, där totalt 38 tangentnedtryckningar görs. I tabellen finns också uppfört JTAG-modulens förevisade värde.

Tabell 25. Verifieringsutfall som funktion av antalet tangentnedtryckningar

Knapptryckning, efter reset	7-segment CPU_STATE,IR,PC, Addr	LED	JTAG CPU_state	JTAG IR	JTAG PC	JTAG Addr_bus	JTAG LED
Ingen tryckning	F000	1111	00b	0	0	0	15
1,2,3,4	F000	1111	00b	0	0	0	15
5	D010	1111	01b	0	1	0	15
6	E010	1111	10b	0	1	0	15
7,8	F011	1111	00b	0	1	1	15
9	F111	1111	00b	1	1	1	15
10	D121	1111	01b	1	2	1	15
11	E121	1111	10b	1	2	1	15
12,13	F122	1111	00b	1	2	2	15
14	F222	1111	00b	2	2	2	15
15	D232	1111	01b	2	3	2	15
16	E230	1111	10b	2	3	16	15
17	S230	1111	11b	2	3	16*	15
18	S230	1010	11b	2	3	16*	10

19,20	F233	1010	00b	2	3	3	10
21	F133	1010	00b	1	3	3	10
22	D143	1010	01b	1	4	3	10
23	E143	1010	10b	1	4	3	10
24,25	F144	1010	00b	1	4	4	10
26	F244	1010	00b	2	4	4	10
27	D254	1010	01b	2	5	4	10
28	E250	1010	10b	2	5	16*	10
29	S250	1010	11b	2	5	16*	10
30	S250	0001	11b	2	5	16*	1
31,32	F255	0001	00b	2	5	5	1
33	F355	0001	00b	3	5	5	1
34	D365	0001	01b	3	6	5	1
35	E315	0001	10b	3	1	5	1
36,37	F311	0001	00b	3	1	1	1
38	F111	0001	00b	1	1	1	1

6.2 Konfigurering av ISSPE

Efter lysdiodsvalideringen skapades ett testprojekt där komponenten CPU_VHDL_project, samt en IP-komponent av
typen ”In Systems Sources and Probe”. Referarar till denna senare komponent såsom JTAG-komponeneten.

JTAG-komponenten source-utgång är reset_n-signalen och dess probe-ingångar är 31 bitar som är fördelade på signalerna key, CPU_state, IR, PC , Adress-buss och Ledr, enligt följande:

```
probe_signal(30 downto 30) <= manual_clock_signal;
probe_signal(29 downto 28) <= CPU_STATE_signal;
probe_signal(27 downto 20) <= Addr_bus_signal;
probe_signal(19 downto 12) <= PC_signal;
probe_signal(11 downto 4)  <= IR_signal;
probe_signal (3 downto 0)  <= LEDR_signal;
```

Testprojektets entitet tar som insignal den manuella klockan som transmitterades vidare till CPU_VHDL_project samt JTAG-komponeneten.
Utsignaler från testprojektets entitet är de 4 sjusegmentssignalerns som går till sjusegmenten HEX3 till HEX0 på kortet, samt ledsignalerna som går till Ledr3 till Ledr0 på kortet.

```
entity isspe_text is
port(
      CLOCK_50 : IN  STD_LOGIC;
      manual_clock : IN std_logic_vector(0 downto 0);
      HEX0  :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);--Adress bus value as seven-
seg
      HEX1  :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);--PC value as seven seg
      HEX2  :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);--IR as seven-seg
      HEX3  :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);--CPU state as sevens eg
      LEDR  :  OUT  STD_LOGIC_VECTOR(3 DOWNTO 0)-- coupled to LEDR of DE1-Soc
      );
end entity;
```

För att komma åt CPU:ns interna signaler i binär form och inte kodade med 7-segmentkodning, lades

följande extra utsignaler till port-beskriviningen av CPU_VHDL_project

```
cpu_state_2_JTAG  :     OUT  STD_LOGIC_VECTOR(1 DOWNTO 0);
IR_reg_2_JTAG     :     OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
PC_reg_2_JTAG     :     OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
Addr_bus_2_JTAG   :     OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
```

6.3 Kommentar kring resultatet från JTAG skanningen

Vi kan se en överenstämmelse mellan kortets resultat och den intern JTAG-probningen. Enda skillanden är under cpu-tillståndet STORE, då CPU:n skriver tilll adress 16. Eftersom endast de fyra minst signifikanta bitarna är avkodade som sju-segment visar sjusegmentet för adress-avkodningen värdet noll, då det egentligen är värdet 16, dvs, hexadecimalt 0x10 som ligger på adressbussen.

7 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

8 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 %)

9 KOSTNAD FÖR PROJEKTET

40 timmar à 350 kr blir 14000kr, moms på 25% tilkommer.