AGSTU		**Technical report**		
Author Menyar Hees		Task ECS – Embedded Computer System		
Menyar.hees@gmail.com	Controlled	Version 2.0		Thread

AGSTU AB

TEIS ECS - Embedded Computer System -

Menyar Hees

2024-10-20

Summary: The TEIS computer system is a simple but complete computer that runs on an FPGA board. The system includes a CPU, an address bus decoder, ROM, and an input filter that allows for the choice of manual or automatic clocking for the processor. The system displays registers, buses, and chip-select signals on the card's seven-segment displays and LEDs. The functionality has been verified in the simulator and validated on the DE10-Lite board.

TABLE OF CONTENTS

NG

1. INTRODUCTION

This report describes a computer system written in VHDL. The system has been analyzed through simulation and verification in ModelSim and then programmed and validated on an FPGA board (DE10-lite)

2. CONSTRUCTION

TEIS computer system consists of a CPU, address bus decoder, ROM, an input filter to select manual or automatic clocking of the processor and other I/O. Results are presented on various displays. TEIS computer systems are normally clocked with the built-in 50MHz clock or manually to allow the designer to see what is happening at each individual clock cycle. The next figure shows the top level with inputs and outputs.

2.1 Symbol

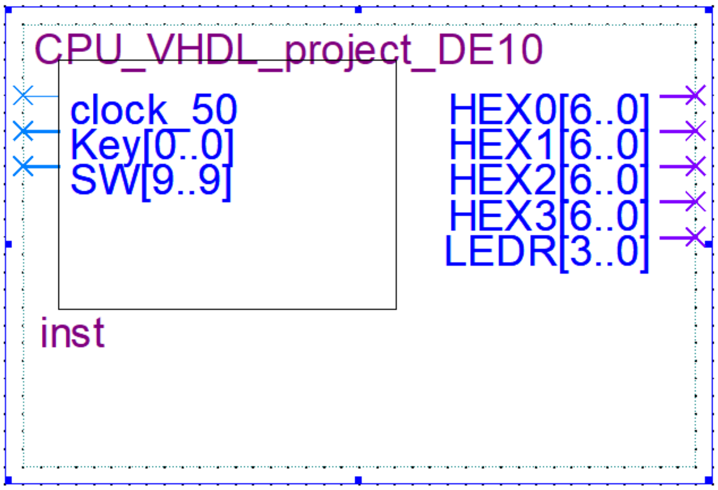


Figure 2. The top level of TEIS microcomputer systems.

2.2 Inputs/Outputs

3. Inputs

- **clock_50**: System clock with a frequency of 50 MHz that controls the entire design.
- **Key**: A simple (1-bit) keypad used as input control for specific events. Here, it is used by the input_filter component.
- **SW**: A 1-bit input to control the reset signal. SW(9) acts as a manual reset and activates when set to low ('0').

4. Outputs

- **HEX0, HEX1, HEX2, HEX3**: Four outputs (7-bit) representing the numerical segments of the 7-segment displays. Each output controls a display that shows different CPU statuses, including address, state, instruction register (IR), and program counter (PC).

- **LEDR**: A 4-bit LED output used to indicate the status and data of the out_led component, which is driven by data from the CPU

.

5. **Signals and Internal Connections**

- The reset_n_t1 and reset_n_t2 signals are used to synchronize the reset signal in two stages.

- LEDR_signal is an intermediate signal attached to the LEDR to control its state.

- The components addr_bus_decoder, status_display_system, input_filter, out_led, rom_vhdl, and simple_vhdl_cpu communicate with each other via different control and data lines such as addresses, clock, and control signals (e.g., WE_n, CS_ROM_n).

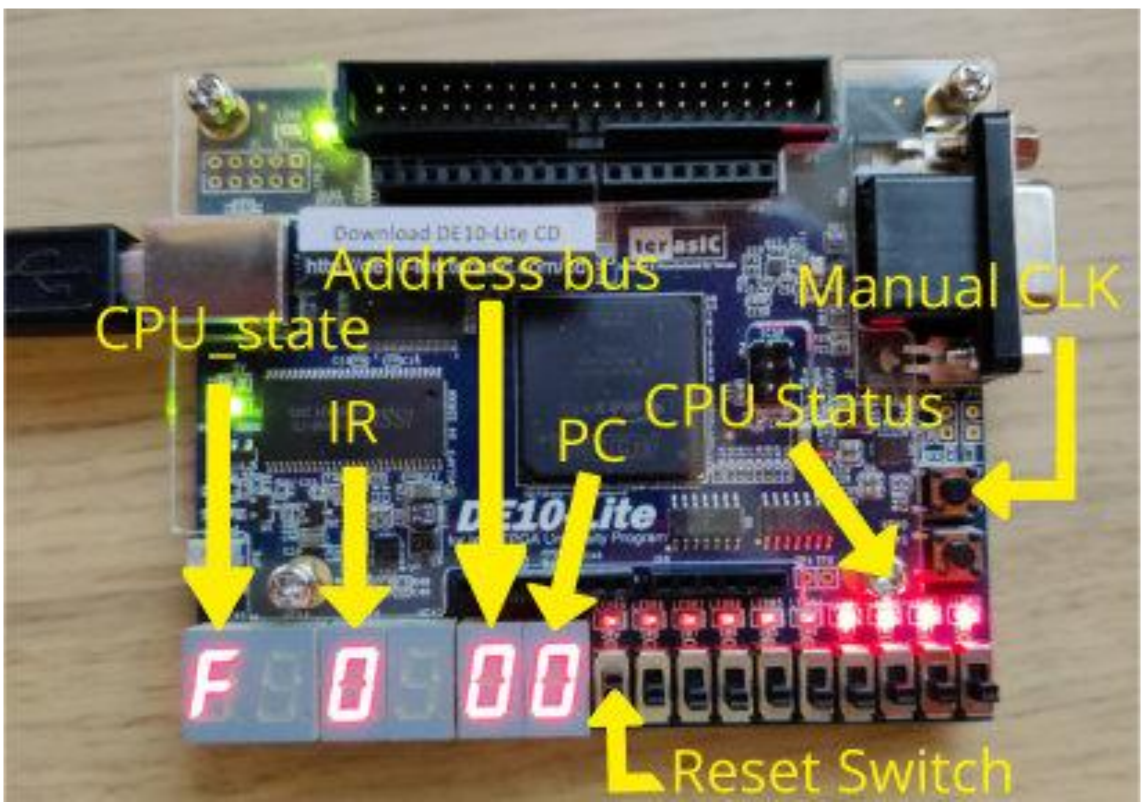


Figure 3. TEIS computer system on DE10-Lite.

5.1 Embedded Computer Systems Architecture (ECS)

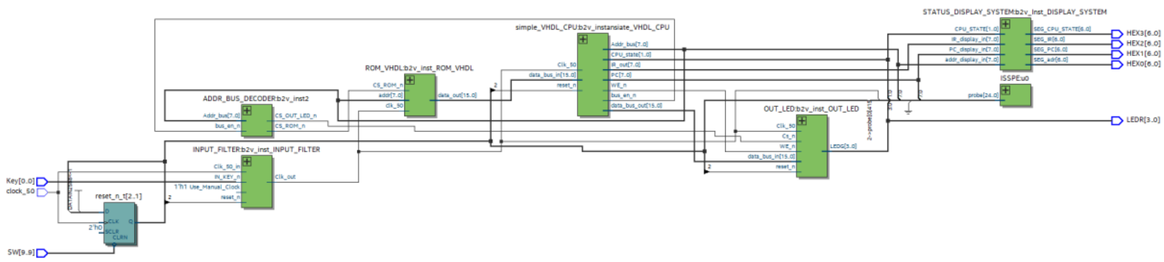


Figure 4. The system architecture of TEIS computer systems.

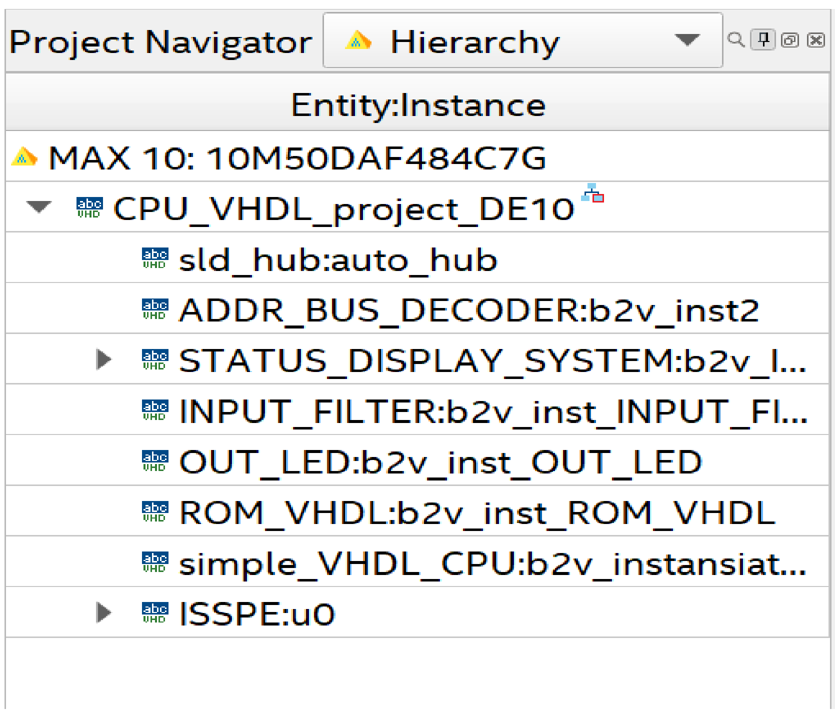


Figure 5. Component hierarchy in TEIS computer systems.

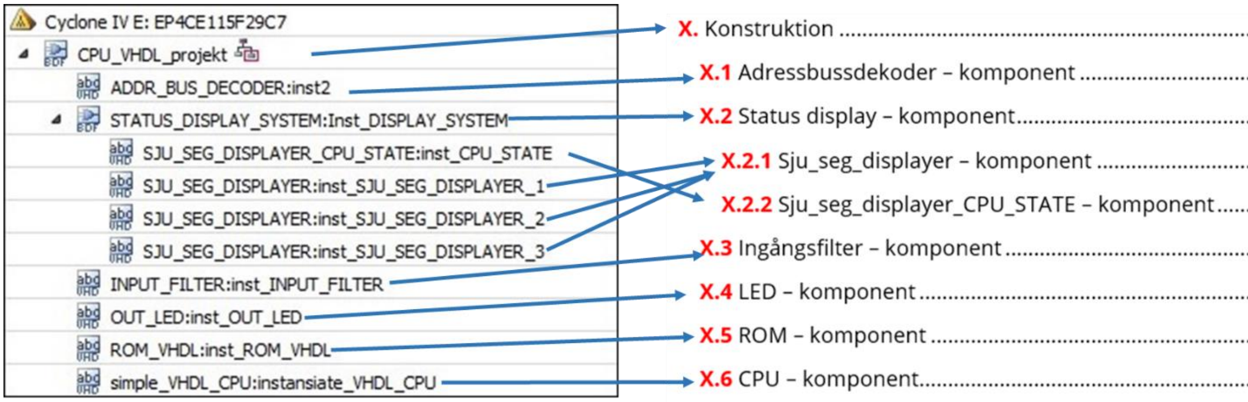


Figure 6. How the component hierarchy determines the design of the technical report.

The system uses a ROM to store the program code in. See the memory folder in Table 5. Results are stored in the system's register.

Table 5. Minnesmapp (supplement, task 7).

Device *	Memorial addresses **	Data Size
ROME	0x0 – 0xF	16
OUT_LED	0x10 – 0x13	4

*) Device (components) that the CPU can read or/and write to

**) Unique addresses of the device

) Data size is in number of bits

5.2 General package and library

Table 6. Packages and libraries.

Component	Std_logic_1164	Std_logic_unsigned	Numeric_std	...
Input Filter	And	No	And	
Out Led	And	No	And	
ROME	And	No	And	
Addr Bus Decoder	And	No	And	
CPU	And	No	And	

5.3 INCLUDED SUBCOMPONENTS

5.3.1 CPU – component

Component Name: simple_VHDL_CPU

Instant: instansiate_VHDL_CPU

5.3.1.1 Function, architecture and permit machine

The CPU component's inputs and outputs are shown in the next figure. The CPU is controlled by reset or clock signal. The CPU works with a state machine that can execute the NOP, LOAD, STORE, and JMP instructions. Internally, the CPU uses the program counter, instruction register, and data register. The program counter points out which instruction is to be retrieved from the ROM.

On reset, the program counter, buses, and registers are initialized to 0 while enable signals are initialized to 1. On positive clock signal, the enable signals are initialized to 1 and then enter the state machine.

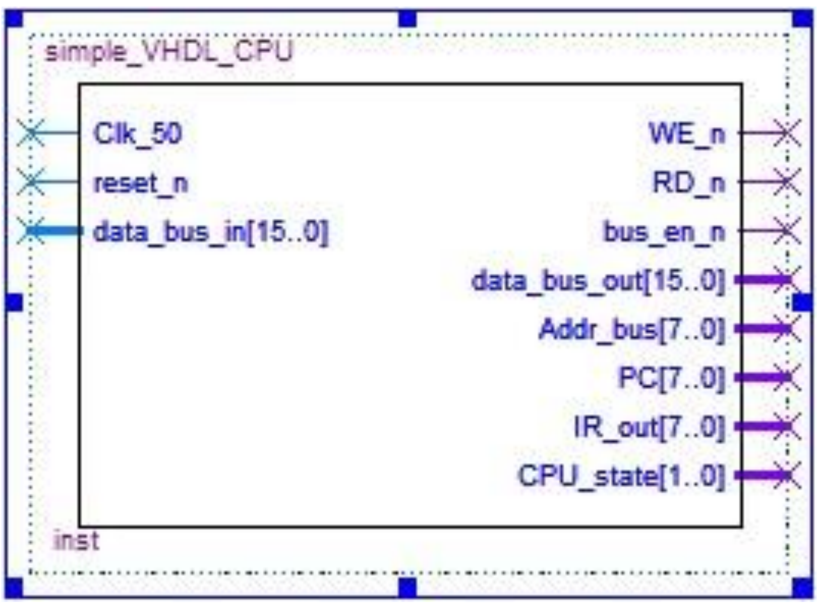


Figure 7. CPU symbol.

5.3.1.2 Inputs/Outputs

CPU input and output signals are displayed in the Table 7.

Table 7. CPU input and output signals.

Signal	**Name**	**Direction**	**Type**
Clock signal 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)
Address bus	Addr_bus	ut	std_logic_vector(7 downto 0)
Program Counter	PC	ut	std_logic_vector(7 downto 0)
Instruction register	IR_out	ut	std_logic_vector(7 downto 0)
State	CPU_state	ut	std_logic_vector(1 downto 0))

5.3.1.3 Permit machine

The state machine works in four state groups, fetch, decode, execute and store. Fetch fetches the next instruction and fetch_1 is started after reset_n. Decode decodes what needs to be done. Execute executes what was decided in the decode phase. Store stores data to memory. The next figure below shows the state machine in detail.

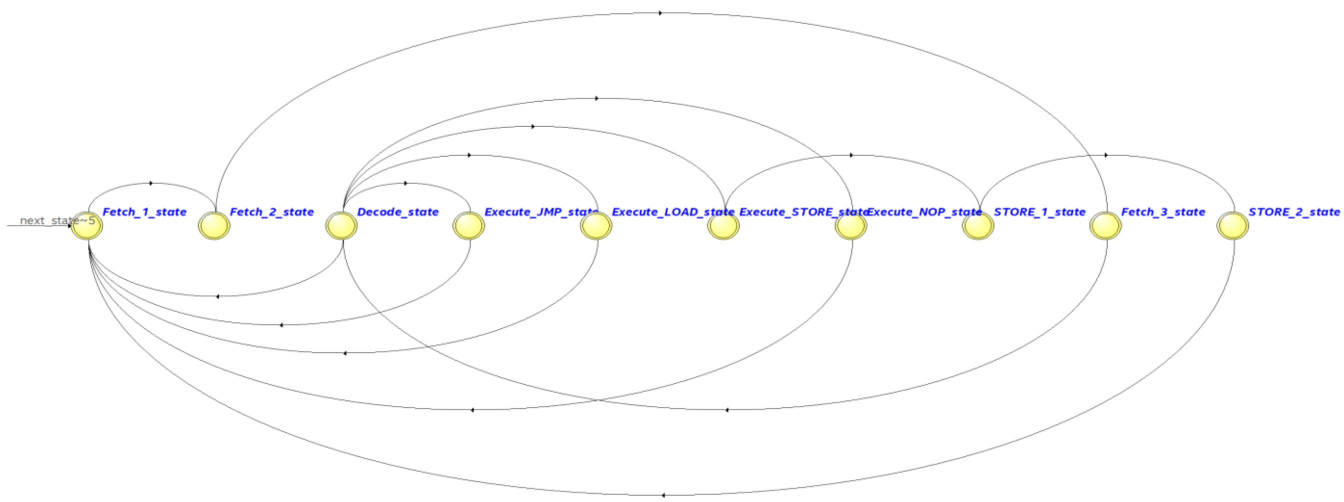


Figure 8. The permit machine

Condition description :

Fetch

CPU_state will receive a value of "00". Data is collected for the instruction register. The value of the program counter is posted on the address bus. RD and Bus Enable are set to 0. In IR, the instruction is entered from the data bus.

- **Decode:**
  First, the program counter is increased by one and CPU_state is given the value "01". Depending on which instruction is in the instruction register, the instruction to be executed is selected in the Execute phase
- **Execute**
  CPU_state get the value "10". If the NOP is executed, nothing is done. If Load is executed, the IR is added to data registers. If Store is executed, the registry is copied to the data bus and the address bus is given the value in IR. If JMP is executed, the program counter gets the value in IR
- **Store**
  CPU_state is given a value of "11". First, write enable and bus enable are set to 0, and then they are set to 1.

5.3.1.4 Description of the CPU Registry, Operations, Data Bus, Address Bus, and Control Signals

The CPU works internally with three different registers. Program counters, instruction registers and data registers. See Table 8 for detailed description.

Table 8. CPUns interna register.

Register	Name	Description
Program Counter	PC_reg	Contains which address the CPU should read from the ROM
Instruction register	AND	Contains what instruction the CPU should execute
Dataregister	CPU_REG_0	Internal register to transfer data

Table 9: Assembler instructions, OP code and coding.

ASM Instruction	# klockcycler	Op-kod	Machine Coding	
			15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
NOP	4	0x0000	0 0 0 0	XXXXXXXXXXX (don't care)
LOAD R0, #imm	4	0x1XXX	0 0 0 1	Imm (data)
STORE R0, (addr)	5	0x2XXX	0 0 1 0	Addr (address)
JMP label	4	0x3xxx	0 0 1 1	Label (hop address)

"LOAD_R0 #DATA" - this assembler instruction loads the R0 registry with data. This means that "DATA" can be a maximum of 12 bits (11-0), because the operation code takes 4 bits.

"STORE_R0 #ADR" - this assembly instruction prints the data saved in RO on the bus to the address "ADR".

5.3.1.5 Description of the operation of the CPU

The principle of how a CPU works is based on a cyclic process in which it retrieves instructions from memory, decodes them to identify which operation to perform, and then executes the instruction by manipulating data, writing to register or memory, or jumping to another instruction. This process is repeated continuously as long as the CPU

is up and running.

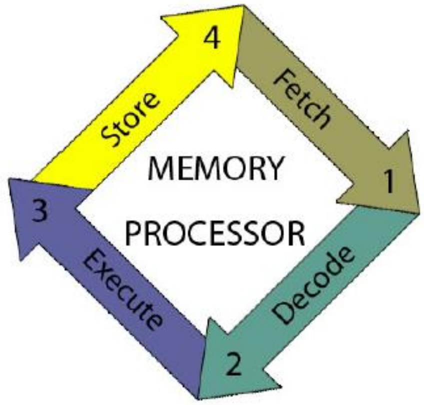


Figure 9. The CPU's way of working.

5.3.2 ROME – Component

Component Name: ROM_VHDL

Instance Name: b2v_inst_ROM_VHDL

5.3.2.1 Function and architecture

ROM is used to store the machine code that the CPU will execute. In the event of a positive clock flank, the data is posted on the data bus at the address ROM has as the input signal. Chip select is not used internally in the ROM. The ROM component's inputs and outputs are shown in the Figure 10 and the contents of the ROM are shown in the next table. The RTL level of the ROM is shown at the end of the chapter.

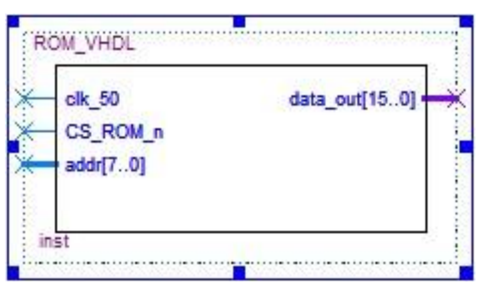


Figure 10. ROM Symbol.

Table 10. Content in ROM.

Address	Machine code [HEX]	Assemblerkod	Instruction [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

5.3.2.2 Inputs/Outputs

The input and output signals to the ROM as shown in the table below.

Table 11. The input and output signals to the ROM.

Signal	Name	Direction	Type
Clock signal 50 MHz	clk_50	in	std_logic
Chipselect	CS_ROM_n	in	std_logic
Address bus	addr	in	std_logic_vector(7 downto 0);
Data ut	data_out	ut	out std_logic_vector(15 downto 0)

5.3.2.3 RTL level

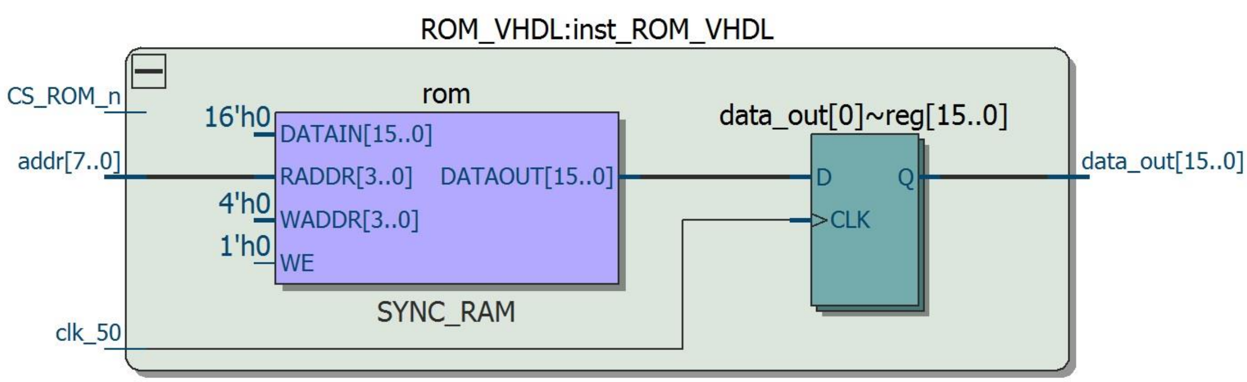


Figure 11. RTL level of ROM.

5.3.2.4 VHDL level

```
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", -- Address 1; LOAD_R0 #A
	X"2010", -- Address 2; STORE_R0 #10
	X"1001", -- Address 3; LOAD_R0 #1
	X"2010", -- Address 4; STORE_R0 #10
	X"3001", -- Adress 5; JMP #1
	X"0000",
	X"0000",
	X"0000", -- Address 8
	X"0000",
	X"0000",
	X"0000",
	X"0000", -- Address 12
	X"0000",
	X"0000",
	X"0000"); -- Address 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;
```

5.3.3 LED Component

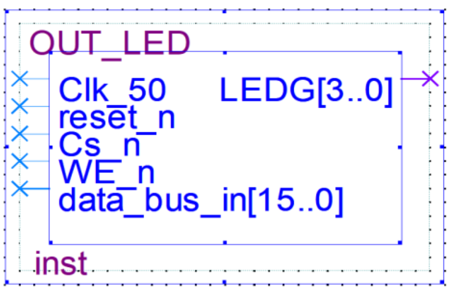
Component: LED_VHDL

Instance Name: b2v_inst_OUT_LED

5.3.3.1 Function and architecture

The LED component is used to control a number of light-emitting diodes (LEDs) based on the data sent from the system. At each positive clock flank, the component checks the value of the data bus and turns the LEDs on or off in accordance with the binary values sent to its inputs. This component is simple and has no internal calculations, but directly controls the output signal to the LED bank.

Figure 12. LED_OUT Symbol



5.3.3.2 Inputs/Outputs

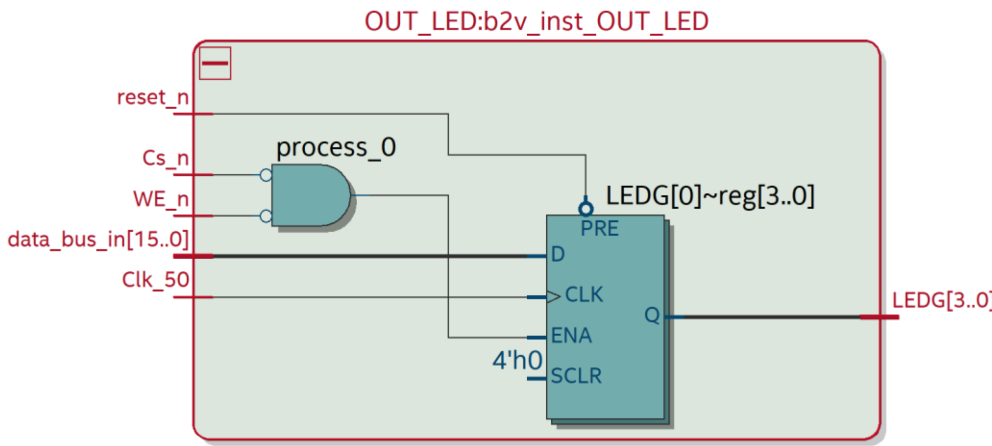
The table below shows the input and output signals of the LED component.

Table 12. In and out signals to the LED component

Signal	Name	Direction	Type
Clock signal 50 MHz	clk_50	in	std_logic
Data in	data_in	in	std_logic_vector(7 downto 0)
LED outputs	led_out	ut	std_logic_vector(7 downto 0)

5.3.3.3 RTL level

Figure 13. RTL level of LED component



5.3.3.4 VHDL-code

```
entity LED_VHDL is
    port (
        clk_50      : in std_logic;
        data_in     : in std_logic_vector(7 downto 0);
        led_out     : out std_logic_vector(7 downto 0)
    );
end entity;

architecture rtl of LED_VHDL is
begin
    process(clk_50)
    begin
        if rising_edge(clk_50) then
            led_out <= data_in;
        end if;
    end process;
end rtl;
```

Can be advantageously sometimes added to attachments. But it's ok to put it here.

5.3.4 Address Bus Decoder – Component

Komponent: Address_Bus_Decoder_VHDL

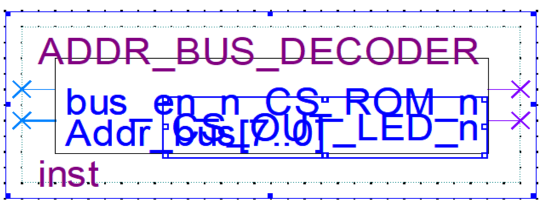
Instance Name: ADDR_BUS_DECODER:b2v_inst2

5.3.4.1 Function and architecture

The address bus decoder is used to control and route the correct signals to the correct components depending on the address the CPU provides on the address bus. The decoder determines which device to activate depending on the addressing range used. The decoder uses a combination of the higher bits in the address bus to identify if the signal is destined for a specific device, and sends the corresponding activation signal to that device.

At each positive clock flank, the decoder reviews the contents of the address bus and then controls the outputs depending on the component requested. This is done by comparing parts of the address with predefined patterns assigned to specific devices. If a matching pattern is found, the correct chip select signal is activated.

Figure 14. Symbol on Address Bus Decoder



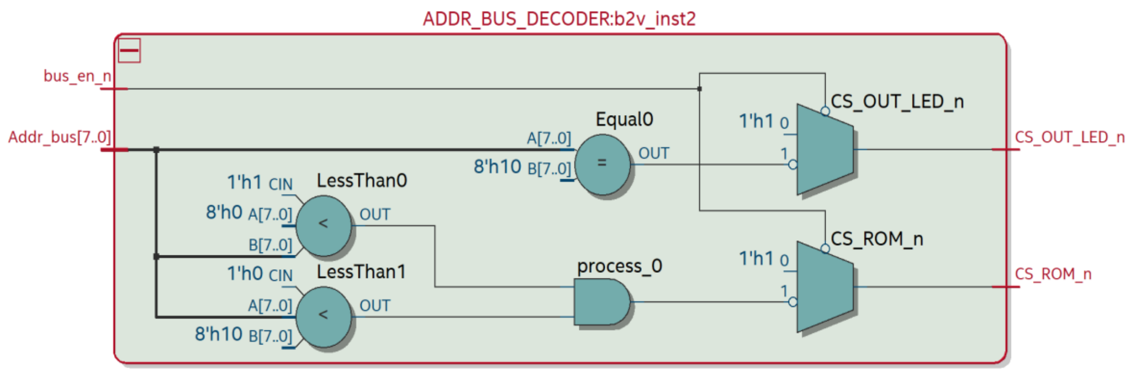
5.3.4.2 Inputs/Outputs

Table 13. In and out signals to Address Bus Decoder

Signal	Name	Direction	Type
Bus Enable	bus_en_n	in	std_logic
Address bus	Addr_bus	in	std_logic_vector(7 downto 0)
Chip Select	CS_ROM_n	ut	std_logic
Chip Select	CS_OUT_LED_n	ut	std_logic

5.3.4.3 RTL level

Figure 15. RTL level of Address Bus Decoder



5.3.4.4 VHDL-code

```
entity ADDR_BUS_DECODER is
    port
    (
    CS_ROM_n                : out std_logic;
    CS_OUT_LED_n            : out std_logic;
    bus_en_n                : in std_logic;
    Addr_bus                : in std_logic_vector(7 downto 0)
    );
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
        if unsigned(addr_bus) >= 0 AND unsigned(addr_bus) < 16 then  -- 0-15;
ROM adress
            CS_ROM_n <= '0';
        else
            CS_ROM_n <= '1';
        end if;

        if addr_bus = "00010000" then  -- 16 ; OUT_LED adress
            CS_OUT_LED_n <= '0';
        else
            CS_OUT_LED_n <= '1';
        end if;
    else
        CS_ROM_n <= '1';
        CS_OUT_LED_n <= '1';
    end if;

end process;

 end rtl;
```

5.3.5 Input Filter – Component

Component: INPUT_FILTER

Instant: inst_INPUT_FILTER

Generic Parameter:

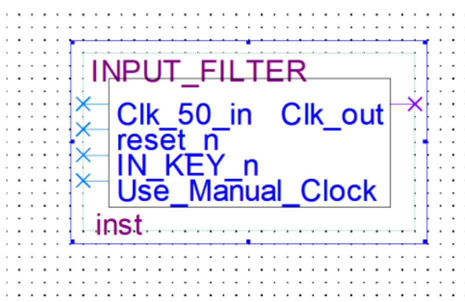
cnt_high: integer := 20

This value determines how many clock pulses the button must be stable for it to register.

Function and architecture

The input filter is used to generate a clock signal to the CPU. The clock signal can be selected as internal or manual clocking with a push button. If Use_Manual_Clock = 0, the card's clock is used, otherwise the KEY0 push button is used on the card. The filter uses a generic, cnt_high to determine how many clock pulses are needed for the signal to be considered stable.

Figure 16. Input Filter Symbol



5.3.5.1 Inputs/Outputs

Table 14. Inputs/outputs on INPUT_FILTER

Inputs/Outputs	Name	Direction	Type
50 MHz clock signal	Clk_50_in	in	std_logic
Push-button	IN_KEY_n	in	std_logic
Reset-signal	reset_n	in	std_logic
Manual clock control	Use_Manual_Clock	in	std_logic
Manual clock control	Clk_out	ut	std_logic

5.3.6 Status display – component

Component:

Instance Name:

5.3.6.1 Function and architecture

The status display system presents the address bus, program counter, instruction register and CPU state on four 7-segment displays. On the DE10-Lite board are these displays. The status display is divided into four different subsystems. CPU state (SJU_SEG_DISPLAYER_CPU) is one part. The other three are instantiations of a component (SJU_SEG_DISPLAYER). The next figure and the next table are shown in and out of the status display system.

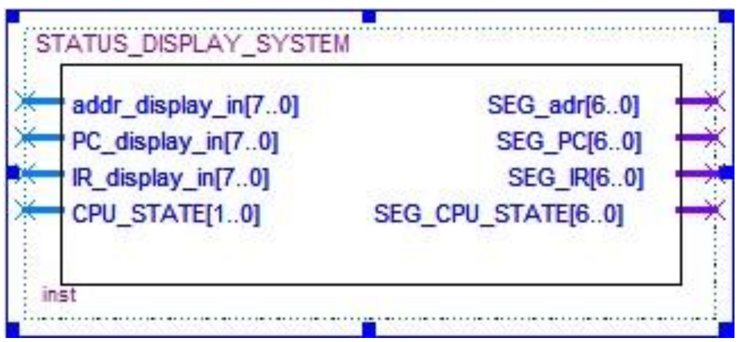


Figure 17. Status display symbol.

The subsystems are shown in the next figure. How the information is presented is shown in the next table.

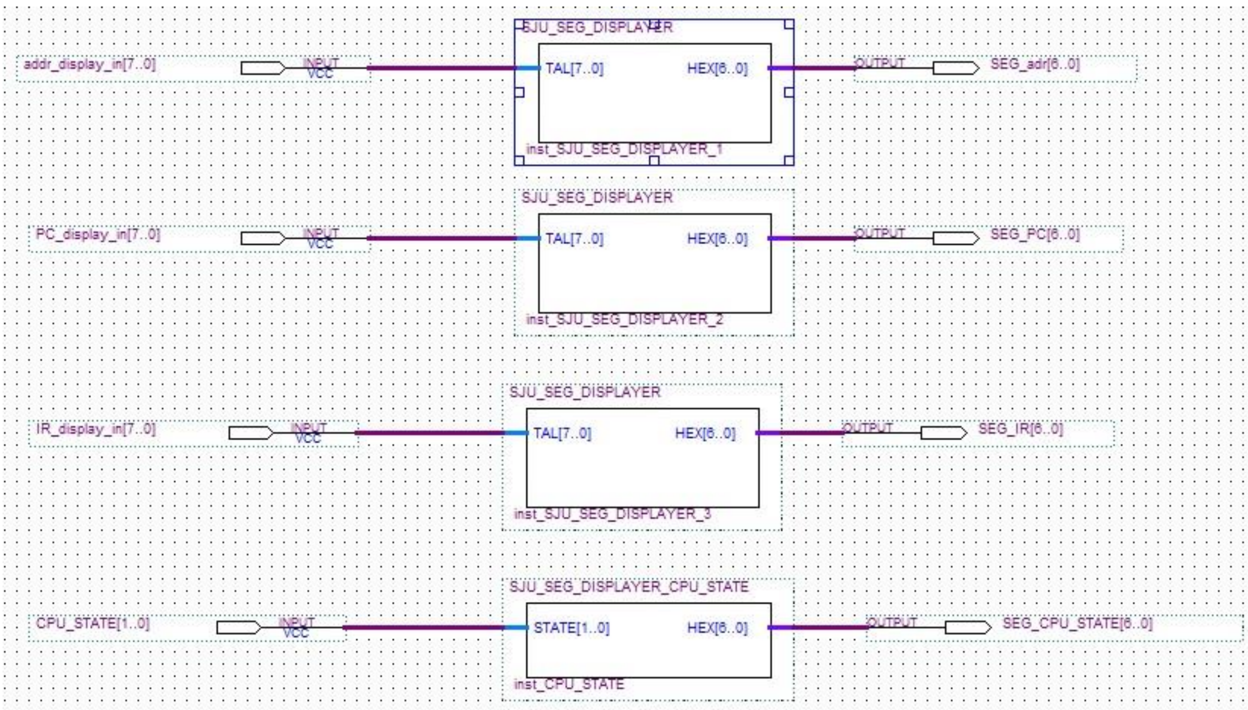


Figure 18. Status displaysystem subsystem (arkitektur).

5.3.6.2 Inputs/Outputs

Table 15. The input and output signals in the status display system.

Signal	Name	Direction	Type
Address bus	Addr_display_in	in	std_logic_vector(7 downto 0)
Program Counter	PC_display_in	in	std_logic_vector(7 downto 0)
Instruction register	IR_display_in	into	std_logic_vector(7 downto 0)
CPU state (STATE)	CPU_state	in	std_logic_vector(1 downto 0)
The value of the address bus reinterpreted to 7-segment information	SEG_adress	ut	std_logic_vector(6 downto 0)
Program Counter Value Reinterpreted to 7-segment information	SEG_PC	ut	std_logic_vector(6 downto 0)
The value of the instruction register reinterpreted to 7-segment information	SEG_IR	ut	std_logic_vector(6 downto 0)

CPU State (STATE) value reinterpreted to 7-segment information	SEG_CPU_STATE	out	std_logic_vector(6 downto 0)

Table 16. Information on the 7-segment displays. (Task 7)

Name	Presented information	Display on DE10-Lite
SJU_SEG_DISPLAYER_1	Address bus	HEX 0
SJU_SEG_DISPLAYER_2	Program Counter	HEX 1
SJU_SEG_DISPLAYER_3	Instruction Register	HEX 2
CPU_STATE	CPU state	HEX 3

5.3.6.3 Sju_seg_displayer – Component

Component: SJU_SEG_DISPLAYER

Instansnamn: inst_SJU_SEG_DISPLAYER_1, inst_SJU_SEG_DISPLAYER_2, inst_SJU_SEG_DISPLAYER_3 och inst_CPU_STATE_DISPLAY _CPU

- Instance Name: inst_SJU_SEG_DISPLAYER_1 (Address Bus) - Associated with HEX0
- Instance Name: inst_SJU_SEG_DISPLAYER_2 (Program Counter) - Associated with HEX1
- Instance Name: inst_SJU_SEG_DISPLAYER_3 (Instruction Register) – Associated with HEX2
- Instance Name: inst_CPU_STATE_DISPLAY (CPU

state) – linked to HEX3

The fourth instance name, inst_CPU_STATE_DISPLAY, can be used to show the CPU state on HEX3.

Function and architecture

The SJU_SEG_DISPLAYER_1 component shows the address bus on the 7-segment display hex0. The input signal TAL is reinterpreted as the output signal HEX so that the 7-segment display shows the value of TAL. The display is updated when TAL changes state. See the next figure.

Figure 19. Sjusegment display 1

Inputs/Outputs

Table below shows ....

Table 17. Signals for the presentation of the address bus.

Signal	External name	Internal name	Direction	Type
Address buses	addr_display_in	SPEECH	into	std_logic_vector(7 downto 0)
Control data for HEX6	SEG_adr	HEX	ut	out std_logic_vector(6 downto 0)

RTL level

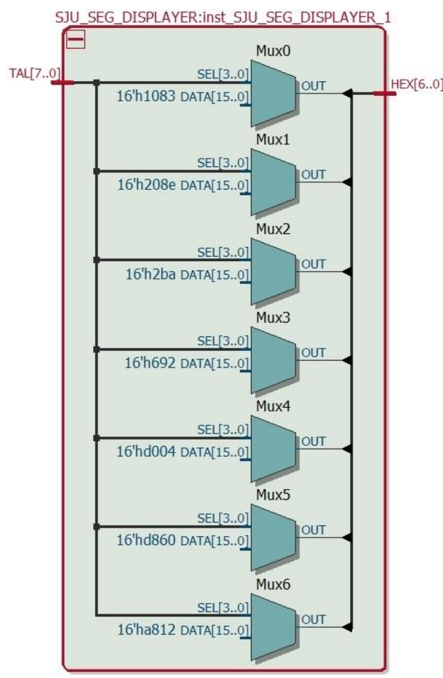


Figure 20. RTL level for seven-segment display 1

5.3.6.4 Sju_seg_displayer_CPU_STATE – component

Komponent: SJU_SEG_DISPLAYER_CPU_STATE

Instance Name: SJU_SEG_DISPLAYER_CPU_STATE:inst_CPU_STATE

Function and architecture

The device CPU_STATE presents the state of the computer on the 7-segment display HEX6. The display will update when STATE changes state. The input and output signals for the CPU_STATE component are shown in the next figure and table.

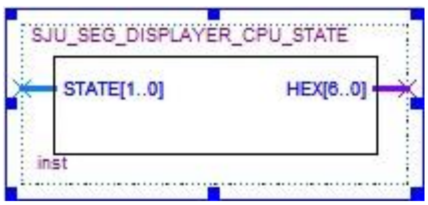


Figure 21. CPU state

Table 18. CPU state

State	HEX6
FETCH	F
DECODE	D
EXECUTE	And
STORE	S
ERROR	8

Inputs/Outputs

6. TEST LOG

Table 19: Sample Test Protocol

Test case	LED	Condition	Correct outcome	Simulated outcome correct?	ISSP validation correct?	Validation with the displays correctly?
1	1111	SW(9) (reset) = 0	PC, IR, R0, Addr_bus, data_bus_out = 0, next state = Fetch_1_state	OK	OK	OK
2		POP (IR = 0x0000)	PC+= 1, no other side effects	OK	OK	OK
3		LOAD_R0 #A (IR = 0x100A)	R0[11..0] = IR[11..0]	OK	OK	OK
4	1010	STORE_R0 #10 (IR = 0x2010)	Addr_bus = IR [7..0], data_bus_out [7..0] = cpu_reg_0 [7..0]	OK	OK	OK
5		LOAD_RO #1 (IR = 0x1001)	R0[11..0] = IR[11..0]	OK	OK	OK
6	0001	STORE_RO #10 (IR = 0x2010)	Addr_bus = IR [7..0], data_bus_out [7..0] = cpu_reg_0 [7..0]	OK	OK	OK
7		JMP #1 (IR = 0x3001)	PC_reg = unsigned(IR[7..0])	OK	OK	OK

7. VERIFICATION WITH MODELSIM

1. **Compile the VHDL code**: Before the simulation begins, all VHDL code must be complied with to ensure that no syntax errors are present. This is usually done in

ModelSim by selecting "Compile" and then specific design files. The compilation also verifies that all dependencies are correctly linked.

2. **Create the test bench**: Before the simulation can run, a test bench must be created to test the design. The test bench is a separate VHDL file that generates specific input signals and checks the expected outputs from the design. The test bench defines the inputs (e.g. reset, clock, and other control signals) as well as test scenarios to which the design should react. The test bench is written to automatically apply these stimuli to the design and capture the outputs.
3. **Running the Simulation**: When the test bench is loaded, you can start the simulation in ModelSim. This executes the test bench, and the ModelSim starts generating and displaying signal values according to the scenarios specified in the test bench.
4. **Analyzing the plusdaiagram (Waveform):** The wave chart in the image shows a timeline of different signals. Here, you can observe input signals such as reset, clock, and any control or data lines, as well as output signals that show the design's response. The wave chart makes it possible to see if the design responds correctly to different inputs over time.

7.1 TEST BENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Testbench for CPU_VHDL_project_DE10
ENTITY CPU_VHDL_project_DE10_vhd_tst IS
END CPU_VHDL_project_DE10_vhd_tst;

ARCHITECTURE CPU_VHDL_project_DE10_arch OF CPU_VHDL_project_DE10_vhd_tst IS
 -- Signals to simulate the external connections
 SIGNAL clock_50         : STD_LOGIC;
   SIGNAL HEX0             : STD_LOGIC_VECTOR(6 DOWNTO 0);
   SIGNAL HEX1             : STD_LOGIC_VECTOR(6 DOWNTO 0);
   SIGNAL HEX2             : STD_LOGIC_VECTOR(6 DOWNTO 0);
   SIGNAL HEX3             : STD_LOGIC_VECTOR(6 DOWNTO 0);
   SIGNAL Key              : STD_LOGIC_VECTOR(0 DOWNTO 0);
   SIGNAL LEDR             : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL SW               : STD_LOGIC_VECTOR(9 DOWNTO 9);

 -- System master clock period
   CONSTANT sys_clk_period : TIME := 20 ns;

 -- Component declaration that matches the entity of the main design exactly
 COMPONENT CPU_VHDL_project_DE10
      PORT (
 clock_50 : IN STD_LOGIC;
         Key      : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
         SW       : IN STD_LOGIC_VECTOR(9 DOWNTO 9);
```

```
         HEX0      : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
         HEX1      : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
         HEX2      : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
         HEX3      : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
         LEDR      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
 );
 END COMPONENT;

BEGIN
 -- Instantiate the main design for testing purposes
 i1 : CPU_VHDL_project_DE10
      PORT MAP(
 clock_50 => clock_50,
 HEX0     => HEX0,
 HEX1     => HEX1,
 HEX2     => HEX2,
 HEX3     => HEX3,
 LEDR     => LEDR,
 Key      => Key,
 SW       => SW
 );

 -- Process for generating a clock signal
 clock : PROCESS
 BEGIN
 clock_50 <= '0';  -- Set the clock low
      WAIT FOR sys_clk_period / 2;
 clock_50 <= '1';  -- Set the clock high
      WAIT FOR sys_clk_period / 2;
 END PROCESS clock;

 -- Simulate button presses by changing SW(9)
   SW(9) <= '0', '1' AFTER 10 * sys_clk_period;

END CPU_VHDL_project_DE10_arch;
```

This test bench generates a clock signal with a period of 20 ns, connects inputs and outputs to the CPU_VHDL_project_DE10 component, and simulates a sequence of the SW(9) signal by switching it from '0' to '1' after a certain amount of time.

7.2 DO-FIL

```
# Enable print logging to get a transcript of all the commands that are
executed.
transcript on

# Check if the directory 'vhdl_libs' exists, if not, create it.
if ![ file isdirectory vhdl_libs] {
 file mkdir vhdl_libs
}

# Create and map VHDL libraries for Altera.
vlib vhdl_libs/altera
vmap altera ./vhdl_libs/altera
```

```
# Compile the necessary VHDL libraries for Altera.
vcom -93 -work altera
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_syn_attributes.vhd}
vcom -93 -work altera
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_standard_functions.vhd}
vcom -93 -work altera
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/alt_dspbuilder_package.vhd}
vcom -93 -work altera
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_europa_support_lib.vhd}
vcom -93 -work altera
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_primitives_components.vh
d}
vcom -93 -work altera
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_primitives.vhd}

# Create and map VHDL libraries for  Logic Programmable Modules (LPM).
vlib vhdl_libs/lpm
vmap lpm ./vhdl_libs/lpm

# Compile the LPM libraries.
vcom -93 -work lpm {c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/220pack.vhd}
vcom -93 -work lpm
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/220model.vhd}

# Create and map VHDL libraries for sgate (standardgate).
vlib vhdl_libs/sgate
vmap sgate ./vhdl_libs/sgate

# Compile the sgate libraries.
vcom -93 -work sgate
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/sgate_pack.vhd}
vcom -93 -work sgate {c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/sgate.vhd}

# Create and map VHDL libraries for Altera-mf (megafunctions).
vlib vhdl_libs/altera_mf
vmap altera_mf ./vhdl_libs/altera_mf

# Compile the Altera-mf libraries.
vcom -93 -work altera_mf
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_mf_components.vhd}
vcom -93 -work altera_mf
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_mf.vhd}

# Create and map VHDL libraries for Altera lnsim (linear simulation).
vlib vhdl_libs/altera_lnsim
vmap altera_lnsim ./vhdl_libs/altera_lnsim

# Compile the linsim libraries and VHDL components.
vlog -sv -work altera_lnsim
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/mentor/altera_lnsim_for_vhdl.sv
}
vcom -93 -work altera_lnsim
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/altera_lnsim_components.vhd}

# Create and map VHDL libraries for FiftyFiveNM (necessary components for 55nm
technology).
vlib vhdl_libs/fiftyfivenm
```

```
vmap fiftyfivenm ./vhdl_libs/fiftyfivenm

# Compile the FiftyFiveNM libraries.
vlog -vlog01compat -work fiftyfivenm
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/mentor/fiftyfivenm_atoms_ncrypt
.v}
vcom -93 -work fiftyfivenm
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/fiftyfivenm_atoms.vhd}
vcom -93 -work fiftyfivenm
{c:/intelfpga_lite/22.1std/quartus/eda/sim_lib/fiftyfivenm_components.vhd}

# Check if  the 'rtl_work' library already exists, if it does, delete all its
contents.
if {[file exists rtl_work]} {
 vdel -lib rtl_work -all
}

# Create and map the 'rtl_work' working library.
vlib rtl_work
vmap work rtl_work

# Compile the VHDL files included in the project.
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/STATUS_DISPLAY_SYSTEM.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/CPU_VHDL_project_DE10.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/SJU_SEG_DISPLAYER_CPU_STATE.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/SJU_SEG_DISPLAYER.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/simple_VHDL_CPU.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/ROM_VHDL.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/OUT_LED.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/INPUT_FILTER.vhd}
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/ADDR_BUS_DECODER.vhd}

# Compile the VHT file for simulation.
vcom -93 -work work
{C:/AGSTU/Kurs_VHDL_1/VHDL_uppgift_7/Menyar_Hees_vhdl_uppgift_7/CPU_VHDL_proje
ct_DE10_restored/simulation/modelsim/CPU_VHDL_project_DE10.vht}

# Start simulation with specific settings and connection of libraries.
```

```
vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L
fiftyfivenm -L rtl_work -L work -
voptargs="+acc"  CPU_VHDL_project_DE10_vhd_tst

# Add waveforms to monitor all signals.
add wave *

# Add specific signals with dividers for better overview.
add wave -noupdate -divider -height 20 interna_CPU_register
add wave -position insertpoint \
sim:/cpu_vhdl_project_de10_vhd_tst/i1/b2v_instansiate_VHDL_CPU/PC \
Yes:/cpu_vhdl_project_de10_vhd_tst/i1/b2v_instansiate_VHDL_CPU/CPU_state\
Yes:/cpu_vhdl_project_de10_vhd_tst/i1/b2v_instansiate_VHDL_CPU/next_state\
sim:/cpu_vhdl_project_de10_vhd_tst/i1/b2
```

The Do file comes in handy for controlling the simulation automatically and specifying which signals are displayed in the waveform window. Common changes to the .do file include:

- **Adding New Waves Signals**: If new signals have been defined in the test bench that need to be monitored, they can be added to the .do file using commands such as add wave followed by the name of the signal. This way, the signals do not have to be added manually for each simulation.
- **Automate the simulation**: The .do file allows you to set up an automated test flow where the simulation starts, runs for a certain amount of time, and ends. This is especially useful for longer simulations or if many tests need to be rerun.

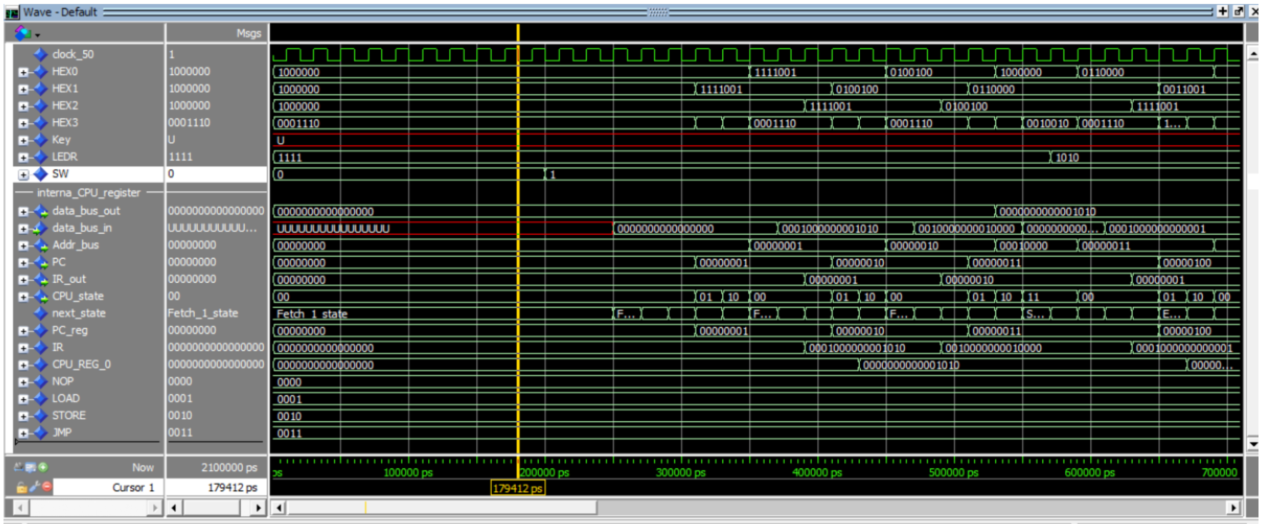
7.3 TEST CASE RESULTS (ACCEPTANCE)

Test-case 1

LED: 1111
Condition: SW(9) (reset) = 0
Correct outcome: When reset, set the program counter (PC), instruction register (IR), register R0, address bus (Addr_bus) and data output bus (data_bus_out) to 0. The expected next state is Fetch_1_state.
Description: This test case verifies that the system resets correctly after activating the reset button, ensuring that all registers and buses return to a defined state.

presented unfilled under the heading TEST PROTOCOL) after each test case that works, if it does not work debug to find the errors. See the next figure.

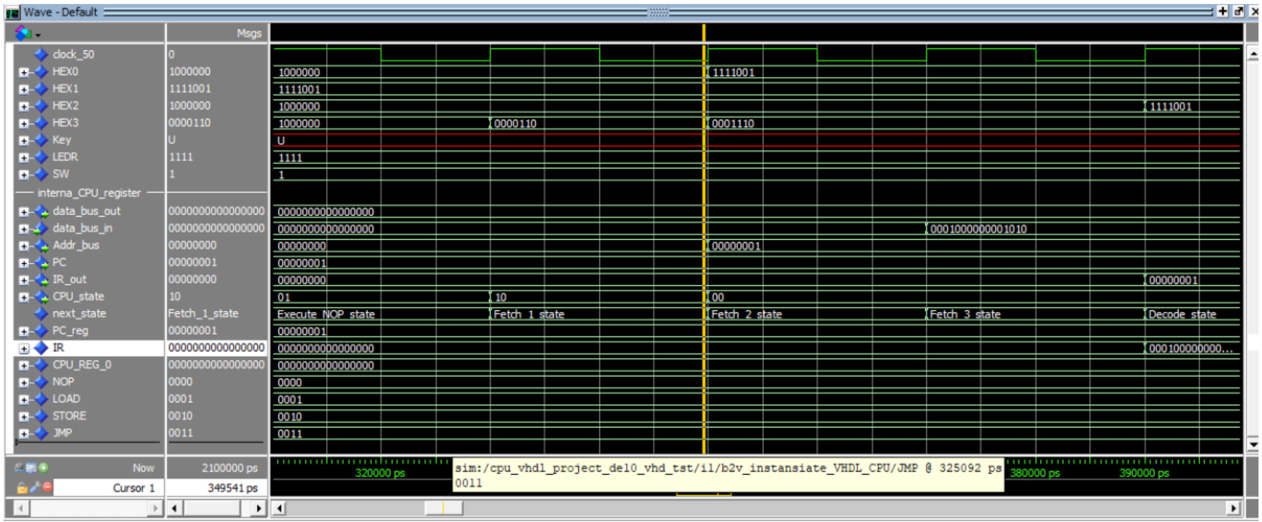
Figure 22. Test-case 1



Test-case 2

LED: -
Condition: NOP (IR = 0x0000)
Correct Outcome: The Program Counter (PC) increases by 1 without any other side effects.
Description: This test case tests the NOP statement, which does not perform any operation except to increase PC by 1. It is used to verify that the processor handles idle cycles correctly. See the next figure.

Figure 23. Test-case 2



Test-case 3

LED: -

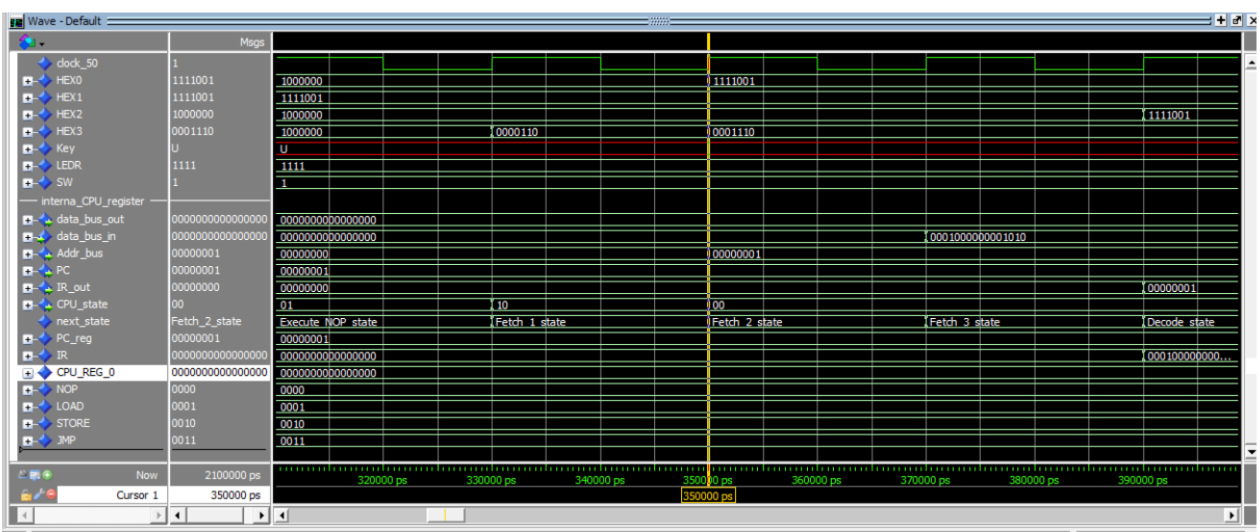
Condition: LOAD_R0 #A (IR = 0x100A)

Correct Outcome: Register R0 should be updated with the value from IR

lower 12 bits (R0[11..0] = IR[11..0]).

Description: This test case verifies that the LOAD statement is working as intended by loading the value from the record's record into R0, which is fundamental to ensuring proper record handling. See the next figure.

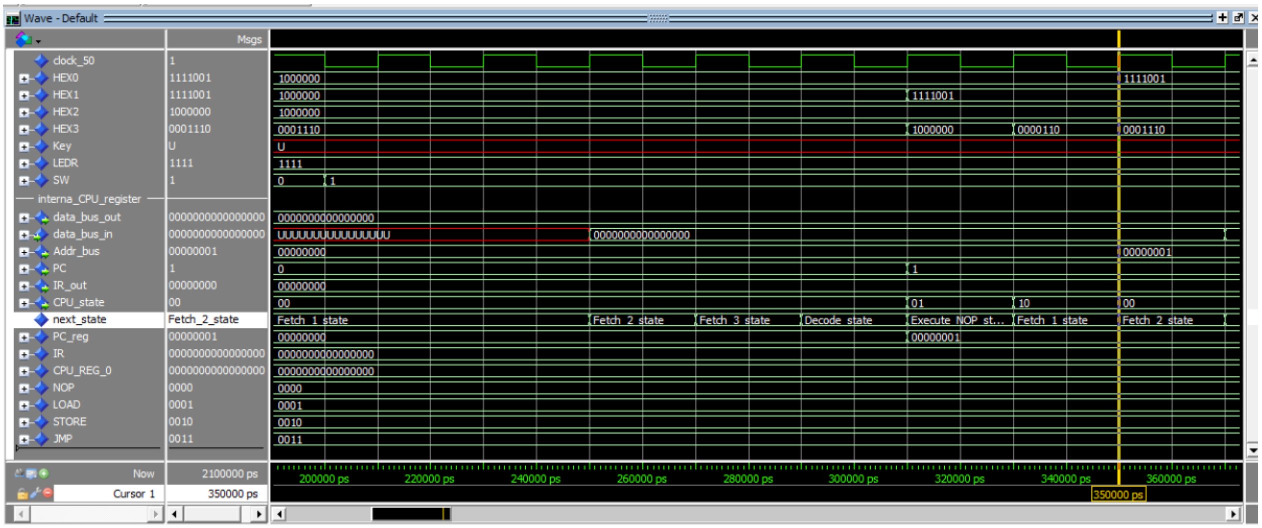
Figure 24. Test-case 3



Test-case 4

LED: 1010
Condition: STORE_R0 #10 (IR = 0x2010)
Correct Outcome: The address bus (Addr_bus) should be set to the lower 8 bits of IR (Addr_bus = IR [7..0]), and the data output bus (data_bus_out) should contain the lower 8 bits of register R0 (data_bus_out [7..0] = cpu_reg_0 [7..0]).
Description: This test case verifies that the STORE statement is working correctly by ensuring that the value in R0 is saved at the correct address, which is critical for in-memory data management. See the next figure.

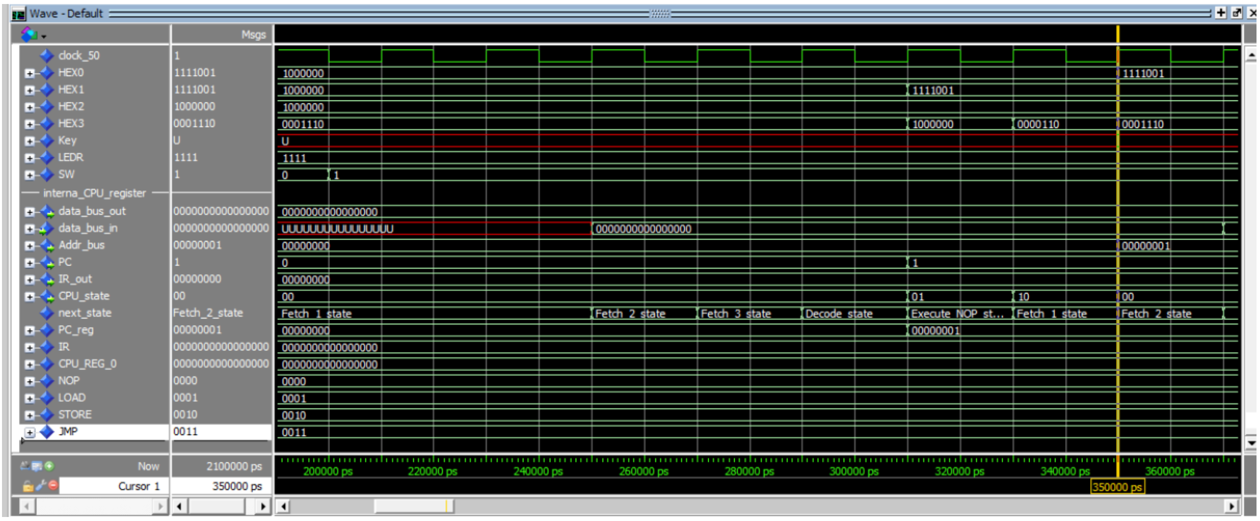
Figure 25. Test-case 4



Test-case 5

LED: -
Conditions : JMP #1 (IR = 0x3001)
Correct Outcome: The Program Counter (PC_reg) should be set to the unsigned value of the lower 8 bits of IR (PC_reg = unsigned(IR[7..0])).
Description: This test case verifies that the JMP statement is working correctly by ensuring that the program counter is updated to the specified address (#1). This is important for testing linguistic instructions that control the flow of the program, which is central to the program being able to navigate between different blocks of code correctly. See the next figure.

Figure 26. Test-case 5



8. VALIDATE WITH ISSPE (Task 9)

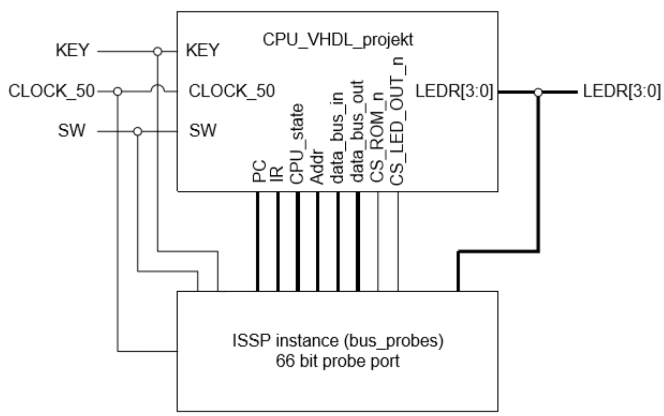


Figure 27: Example image of the architecture with ISSP

8.1 Configuring ISSPE

The ISSPE component serves as a tool to keep track of what's going on inside the VHDL design. It has two main parts, source and probe, which help to get important information about different signals.

- **How the ISSPE component is structured:**

Source: An output that sends a signal (1-bit) that can be used to tell other parts of the design (or external tools) that something special is happening.

Probe: An input (25 bits) where you can send in which signals need to be followed. In this case, the probe is connected to some important internal signals to see what is going on.

2- **What signals are linked to the probe?**

Probe is used as follows:

- Bit 0: reset_n_t2, which shows if the reset is active.
- Bit 1-8: PC, program counter, good for seeing where the CPU
- is in the code.
- Bit 9-10: CPU_state, two bits to see CPU
- state.
- Bit 11-18: Addr_bus, shows which address CPU
- talk to.
- Bit 19: clk_out, the clock signal that syncs the rest.
- Bit 20-23: LEDR_signal, shows the status of the LEDs indicating different states outward.

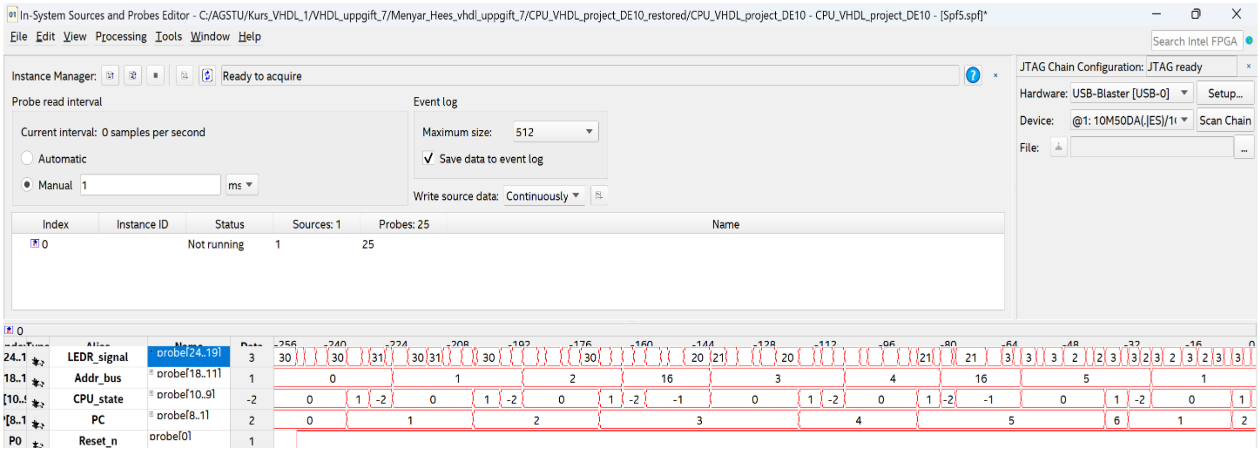
- **What does ISSPE do?**

With ISSPE connected in this way, it is possible to follow all these signals in real time if it is connected to debug tools. ISSPE displays details such as PC counter movements, address switching on Addr_bus, and status on CPU_state. If the source is connected to an important signal, external tools can also be triggered to measure when something specific happens.

8.2 Validate with ISSPE (acceptance)

Validation of the Design with Test Case and Step-by-Step Push Button Control

The figure set below shows how the design has been validated by running test cases one turn and observing the signals out_led, PC, and CPU_State in the ISSPE as well as 7-segment displays and LED lights. The filter has been configured so that the step function is enabled with the push button, which makes it possible to check and observe signal changes step by step. This ensures that the design works as required and that each step can be visually observed for state and output signals. See the next figure.



9. VALIDATE WITH DISPLAYS on the card

validation takes place on a DE10-Lite development board, which is used to validate designs. The circuit shows details of the validation using the 7-segment displays:

- 7-segment displays used to show different statuses and values during validation. And in this case, "F0 0 0" is displayed representing :CPU_state, IR, PC, Adress_bus. These labels point to different components or signals on the board. These are CPU state, instruction register, program counter and address bus.
- LEDR: LED lights are used to provide additional visual feedback. The validation on the target system means that these displays and indicators are used to show and check that the system is working as intended. The displays can show results of operations or

statuses, which helps in troubleshooting and verifying the design.

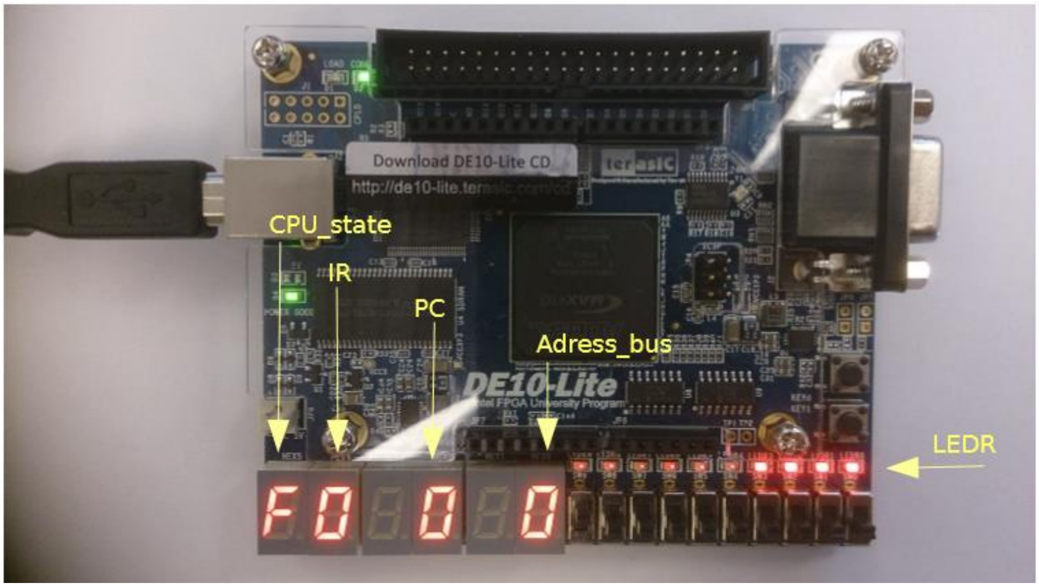
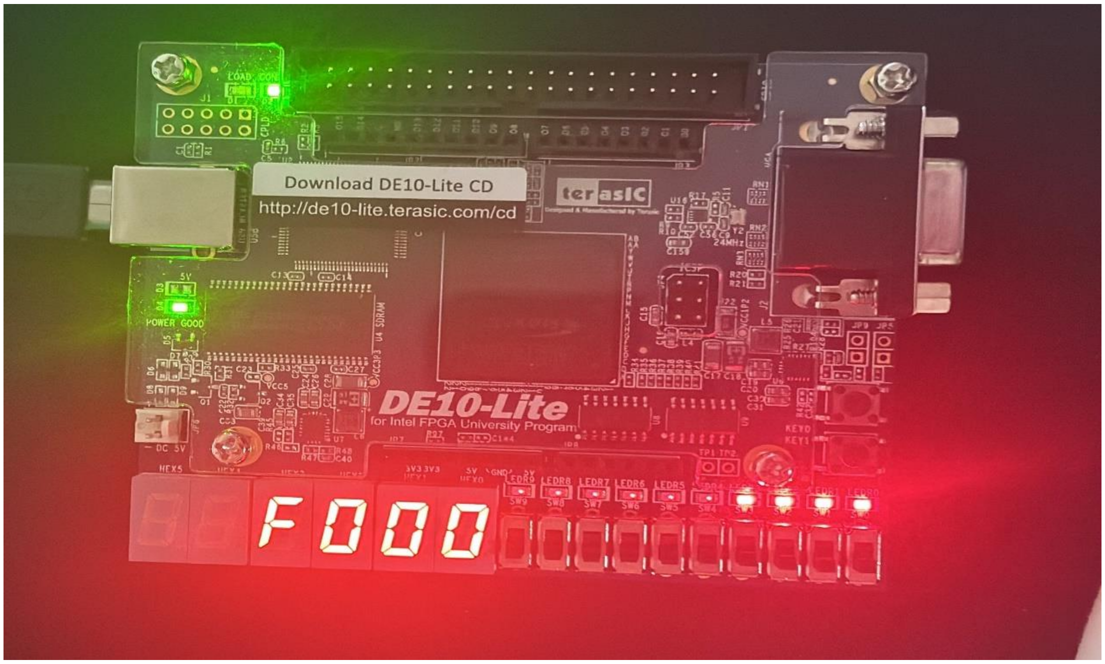


Figure 28: Example image of the validation of the design on the card

9.1 Validate with the Displays (Acceptance)

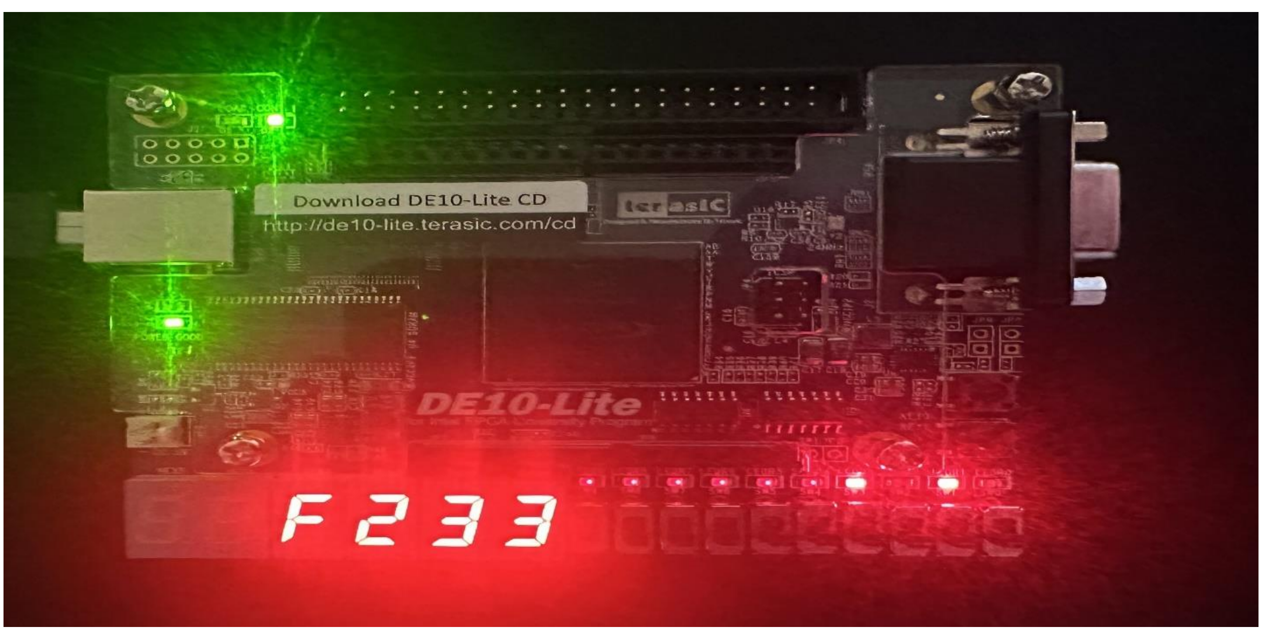
• **Test Case 1**: Activating the reset (SW(9) = 0) resets all critical registers and signals in the system, including program counter (PC), instruction register (IR), register R0, address bus, and data output bus, ensuring that the system restarts in a defined boot state. The next permit is set to Fetch_1_state. See the next figure.

Figure 29. Test-Case (1)



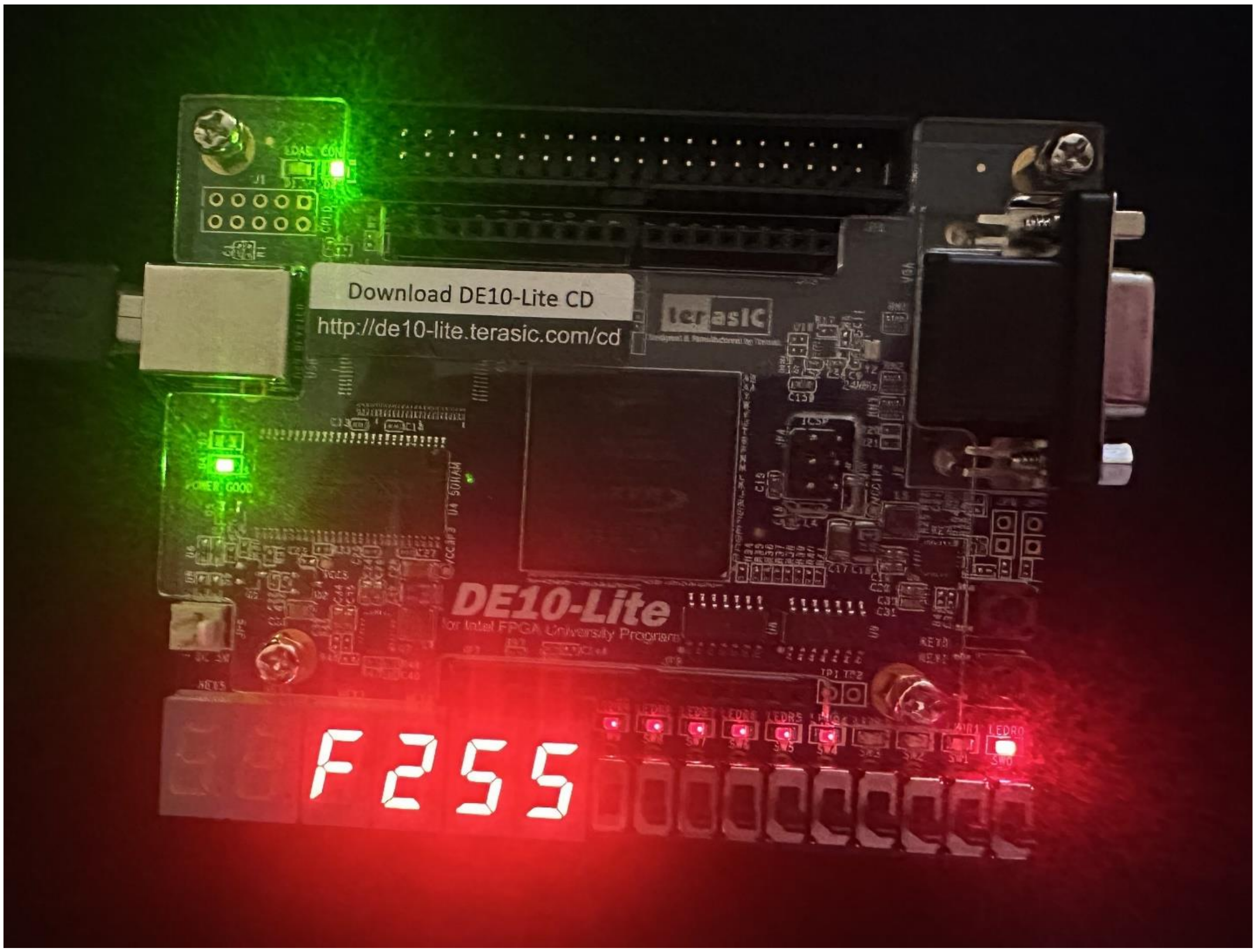
Test case 4: The instruction STORE_R0 #10 (with instruction code IR = 0x2010) is executed. This stores the value in register R0 to the memory address specified in the bottom 8 bits of IR (address 0x10), by setting Addr_bus and data_bus_out with the values from IR and cpu_reg_0. See the next figure.

Figure 30. Test Case (4)



Test case 6: The same instruction (STORE_R0 #10, IR = 0x2010) is executed again, which means that R0 is stored to address 0x10 via Addr_bus and data_bus_out to confirm that the storage operation can be performed consistently. See the next figure.

Figure 31. Test-Case (6)



Summary:

These test cases cover system reset and basic memory storage, verifying both reset functionality and memory operations.

10. FOOT PRINT

Print the size and how much space it takes up on the FPGA chip (in the final report)

The result from the Quartus tool shows the size of the construction which is: 287 logical elements, 183 rockers and 35 pins are used on the circuit. The logic takes up less than 1% of the circuit's capacity and the number of pins 10%. See the next figure.

Figure 32. Footprint info from Quartus.

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Oct 31 19:40:28 2024
Quartus Prime Version	22.1std.2 Build 922 07...0/2023 SC Lite Edition
Revision Name	CPU_VHDL_project_DE10
Top-level Entity Name	CPU_VHDL_project_DE10
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	287 / 49,760 (< 1 %)
Total registers	183
Total pins	35 / 360 (10 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

11. COST OF THE PROJECT

Hourly cost: 350 SEK

Part 1: 14 hours

Part-time 2: 16 hours

Part 3: 8 hours

38 hours Cost: 38 hours x 350 SEK/hour = 13,300 SEK

12. CONCLUSIONS AND FUTURE DEVELOPMENT OPPORTUNITIES

This project has resulted in a working implementation of a computer system on the FPGA, with components such as CPU, ROM, and address interpretation being verified and validated on the DE10-Lite board. With a stable foundation system in place, the following improvements can be considered to increase functionality and robustness in future iterations:

- **Code Optimization** – Refine your VHDL code to reduce resource usage and increase performance.
- **Advanced CPU** – Develop the CPU for more instructions and better performance.
- **Next step**: Focus on optimization and diagnostics for a more robust and usable system.