EL323//El324 Digital System Laboratory II

LAB 11: Complete Simple Processor

Objective:

1. Can use the Quartus II and create the new project using VHDL
2. To know the assign pin of DE-0 in Quartus.
3. Design using VHDL
4. To know the Processor.

LAB 11.1

1. Create new Project name “sControl”.
2. Create new entity for sControl in this below.

   A block diagram of the control circuit of the processor is Figure 11.1 below.

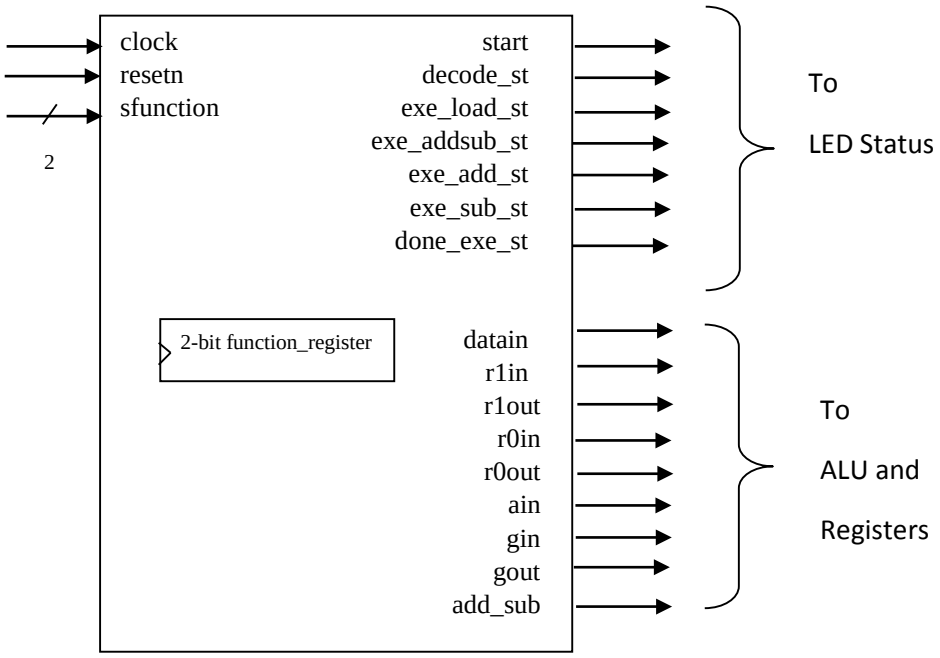


Figure 11.1 The Control Unit

The specific operation of the processor to be performed at any given time is indicated using the control circuit input name “sfunction” as shown in Figure 11.2 below. The control circuit also contains a 2-bit function register.

3. Use the DE0 User Manual to define the DE0 pin.
4. Compile the code and program to DE0.

sfunction	Operation	Function Performed
00	Load R0, data	R0 ← Data
01	Load R1, data	R1 ← Data
10	Add R1, R0	R1 ← R1 + R0
11	Sub R1, R0	R1 ← R1 – R0

Figure 11.2 Operations and functions of **sfunction**

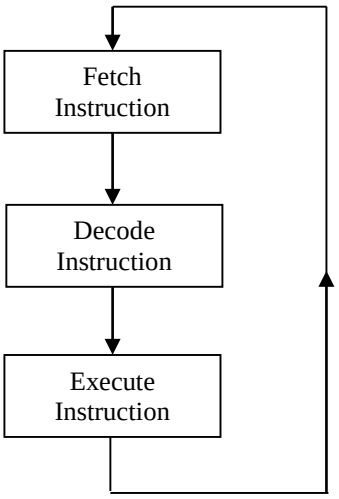


Figure 11.3 Flow chart of the instruction cycle

State	RTL	Start	Datain	R1in	R1out	R0in	R0out	Ain	Gin	Gout	Add_Sub
Fetch	function_register ← sfunction	0	0	0	0	0	0	0	0	0	0
Decode		1	0	0	0	0	0	0	0	0	0
Execute_load_r0	R0 ← Data	1	1	0	0	1	0	0	0	0	0
Execute_load_r1	R1 ← Data	1									
Execute_addsub	A ← R1	1									
Execute_add	G ← A + R0	1									0
Execute_sub	G ← A – R0	1									1
Done_execute_addsub	R1 ← G	1									

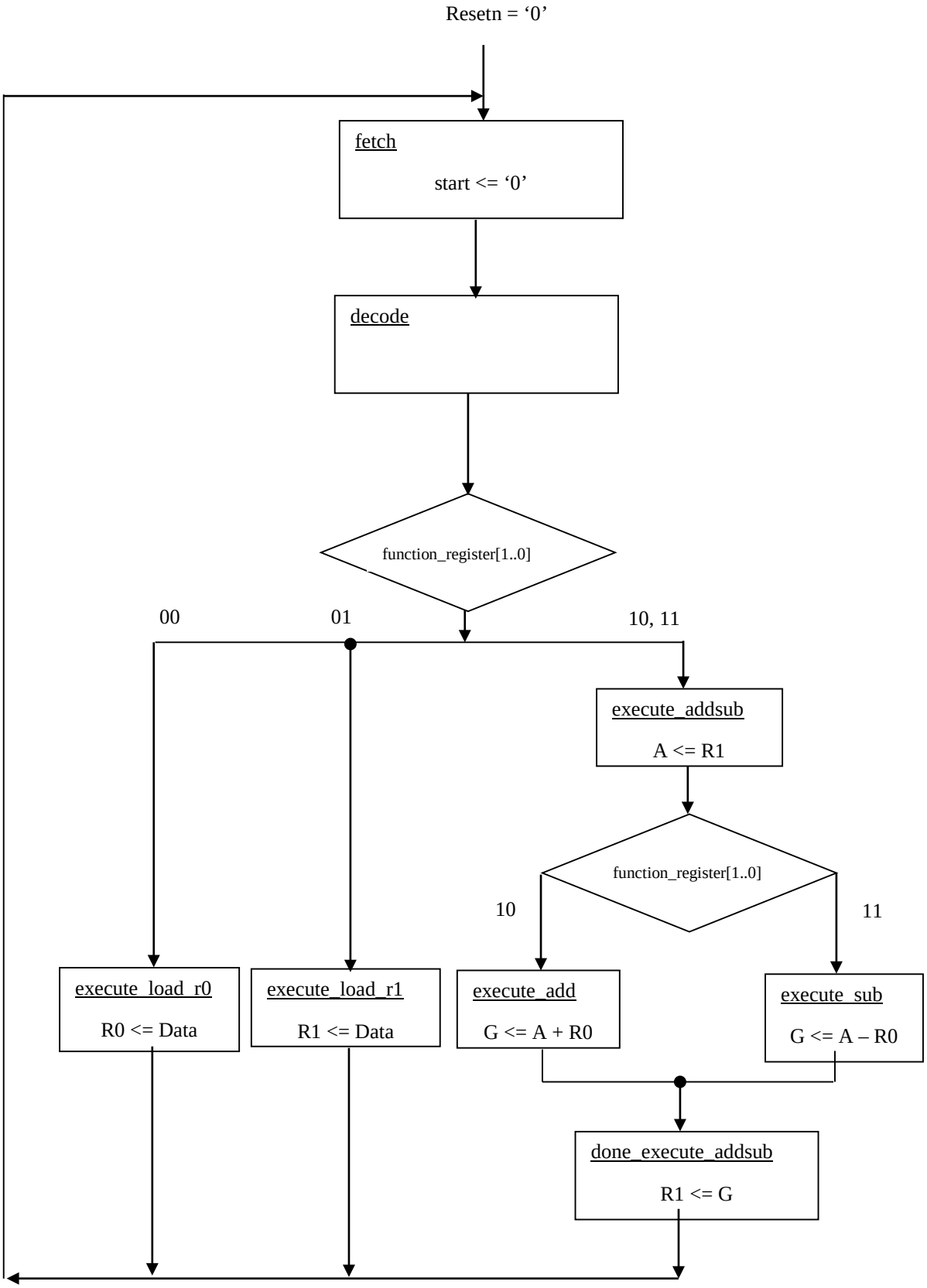


Figure 11.4 State diagram of the control unit of the processor

CODE of "sControl"

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

entity scontrol is
        port( clock, resetn         : in      std_logic;
                sfunction           : in      std_logic_vector(1 downto 0);
                start               : out     std_logic;
                decode_st           : out std_logic;
                exe_load_st         : out std_logic;
                exe_addsub_st       : out std_logic;
                exe_add_st          : out std_logic;
                exe_sub_st          : out std_logic;
                done_exe_st         : out std_logic;
                datain              : out     std_logic;
                r1in, r1out         : out     std_logic;
                r0in, r0out         : out     std_logic;
                ain, gin, gout      : out     std_logic;
                add_sub             : out     std_logic);
end scontrol;

architecture behavior of scontrol is

type state_type is (fetch, decode, execute_addsub, execute_add, execute_sub,
                done_execute_addsub, execute_load_r0, execute_load_r1);

signal state:       state_type;
signal function_register:std_logic_vector (1 downto 0);

begin

states:
process (resetn, clock)
        begin
        if resetn = '0' then state <= fetch;
        elsif clock'event and clock = '1' then
                case state is
                when fetch =>
                        function_register <= sfunction;
                        state <= decode;
                when decode =>
                        case function_register is
                        when "00" =>
                                state <= (_____);
                        when "01" =>
                                state <= (_____);
                        when "10" =>
                                state <= (_____);
```

```
                        when “11” =>
                                state <= (_____);
                        when others =>
                                state <= fetch;
                        end case;
                when execute_addsub =>
                        case function_register is
                        when “10” =>
                                state <= (_____);
                        when “11” =>
                                state <= (_____);
                        when others =>
                                state <= fetch;
                        end case;
                when execute_add =>
                        state <= (_____);
                when execute_sub =>
                        state <= (_____);
                when execute_load_r0 =>
                        state <= (_____);
                when execute_load_r1 =>
                        state <= (_____);
                when done_execute_addsub =>
                        state <= (_____);
                when others =>
                        state <= fetch;
                end case;
        end if;
end process;



controlsignals:

  --Status LED
        start<= '1' when (state = fetch) else '0';
        decode_st <=    '1' when (state = decode) else '0';
        exe_load_st     <=      '1' when (state = execute_load_r0) or (state =
execute_load_r1) else '0';
        exe_addsub_st <=        '1' when (state = execute_addsub) else '0';
        exe_add_st      <=      '1' when (state = execute_add) else '0';
        exe_sub_st      <=      '1' when (state = execute_sub) else '0';
        done_exe_st     <=      '1' when (state = done_execute_addsub) else '0';
```

```
-- Control signal
        datain <= '1' when (state = execute_load_r0) or (state = execute_load_r1) else '0';
        r1in <= '1' when (state = _____) or (state = _____) else '0';
        r1out <= '1' when (state = _____) else '0';
        r0in <= '1' when (state = _____) else '0';
        r0out <= '1' when (state = _____) or (state = _____) else '0';
        ain <= '1' when (state = _____) else '0';
        gin <= '1' when (state = _____) or (state = _____) else '0';
        gout <= '1' when (state = done_execute_addsub) else '0';
        add_sub <= '1' when (state = _____ ) else '0';

end behavior;
```

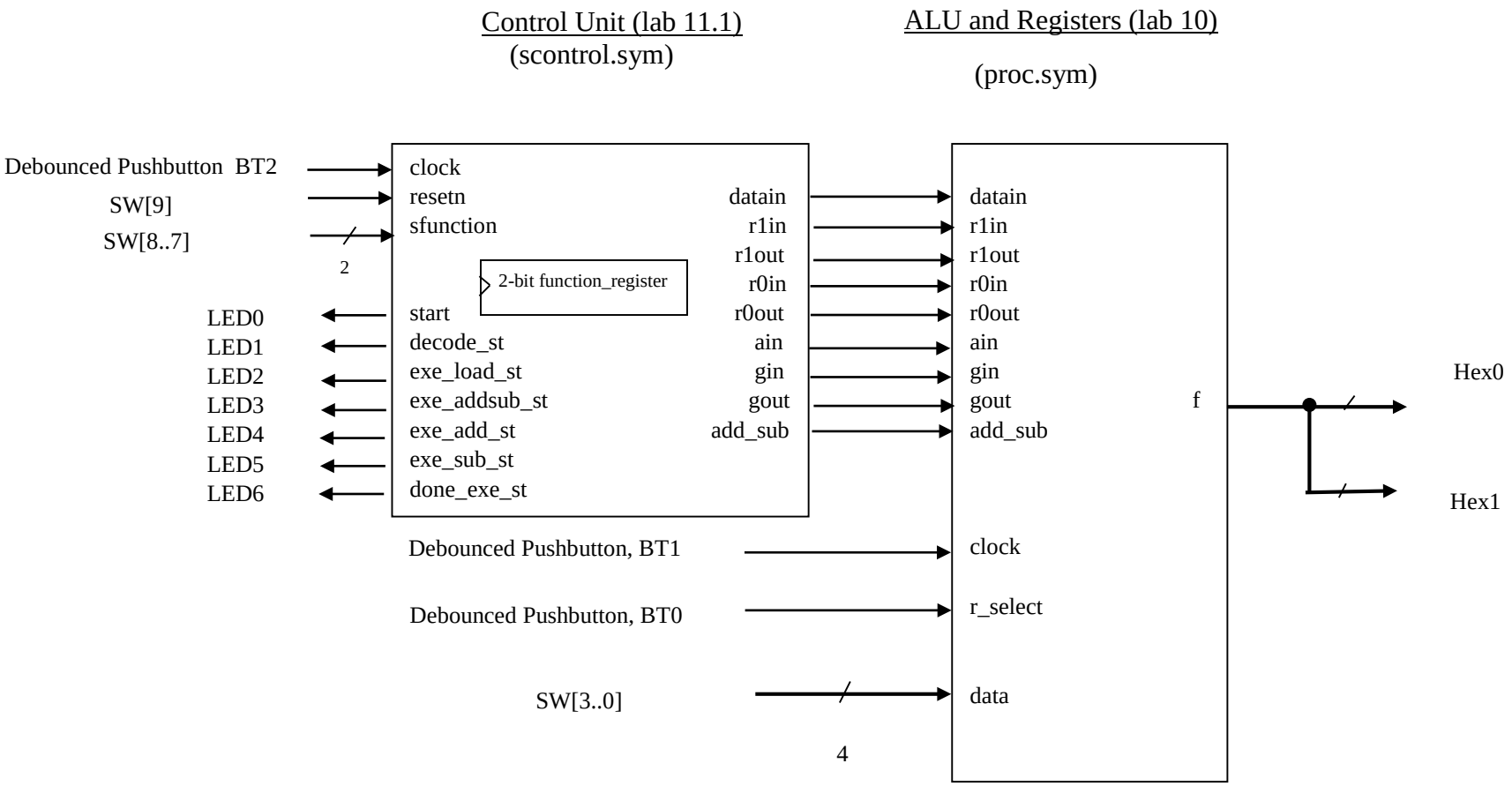


Figure 11.5 The Complete Simple Processor.

LAB 11.2

1. Create new Project name “CompleteCPU”.
2. Create new graphic editor for CompleteCPU as seen in figure 11.5.
3. Use the DE0 User Manual to define the DE0 pin.
4. Compile the code and program to DE0.

Result

Follow the LAB 11.2. Let’s configured the sFunction and data to support this equations. And put the value of Sfuction and Data in the Table 11.

1. Find the 15-4
2. Find the 8+5

Table 11.1

State	RTL	sFunction	Data	Explain

Conclusion

……

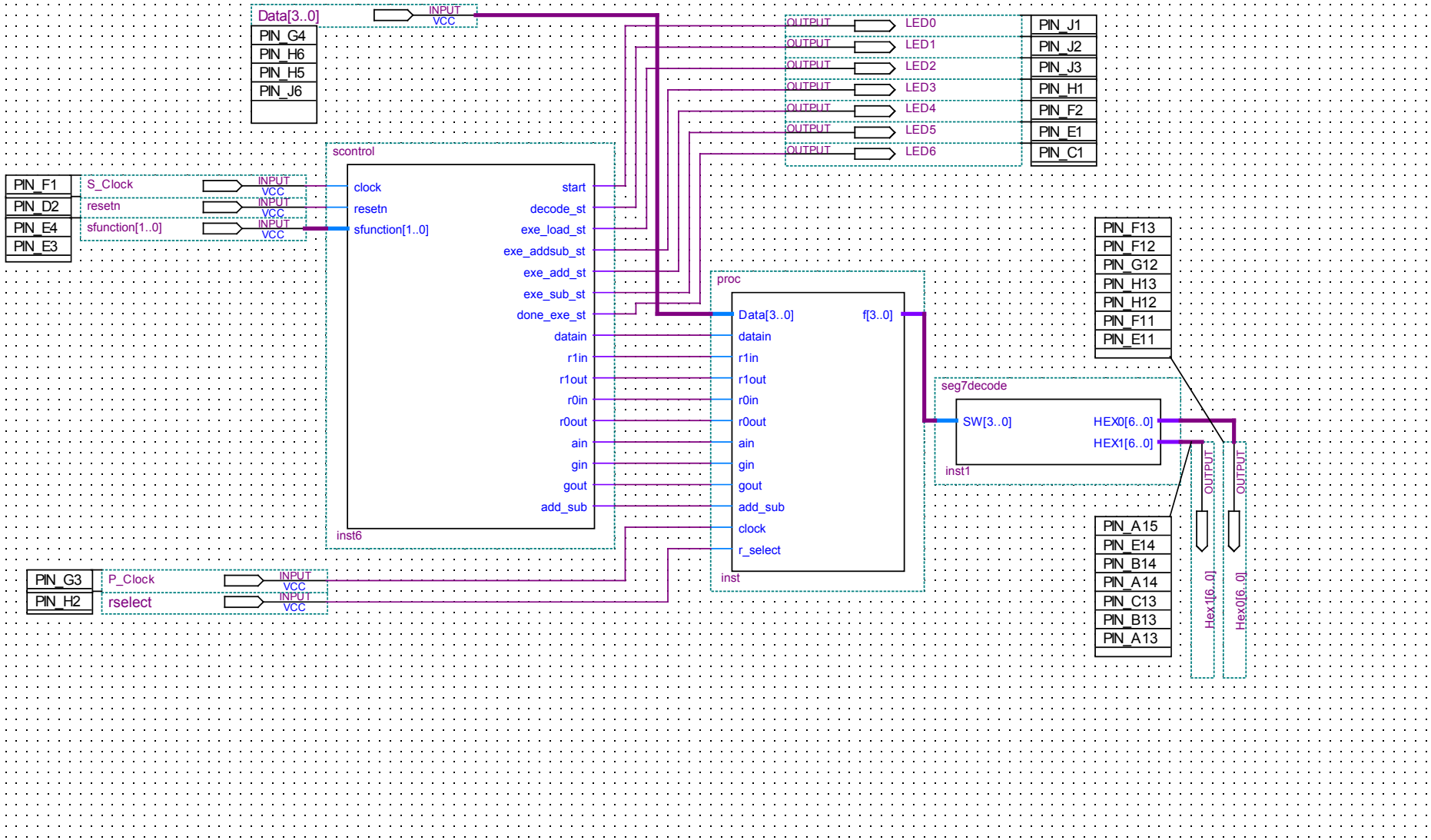
……

……

……

……

……


Data[3..0]
INPUT
VCC
PIN_G4
PIN_H6
PIN_H5
PIN_J6
OUTPUT
LED0
LED1
LED2
LED3
LED4
LED5
LED6
PIN_J1
PIN_J2
PIN_J3
PIN_H1
PIN_F2
PIN_E1
PIN_C1
scontrol
PIN_F1
PIN_D2
PIN_E4
PIN_E3
S_Clock
resetn
sfunction[1..0]
clock
resetn
sfunction[1..0]
start
decode_st
exe_load_st
exe_addsub_st
exe_add_st
exe_sub_st
done_exe_st
datain
r1in
r1out
r0in
r0out
ain
gin
gout
add_sub
inst6
proc
Data[3..0]
f[3..0]
datain
r1in
r1out
r0in
r0out
ain
gin
gout
add_sub
clock
r_select
inst
PIN_F13
PIN_F12
PIN_G12
PIN_H13
PIN_H12
PIN_F11
PIN_E11
seg7decode
SW[3..0]
HEX0[6..0]
HEX1[6..0]
inst1
PIN_A15
PIN_E14
PIN_B14
PIN_A14
PIN_C13
PIN_B13
PIN_A13
Hex1[6..0]
Hex0[6..0]
PIN_G3
PIN_H2
P_Clock
rselect

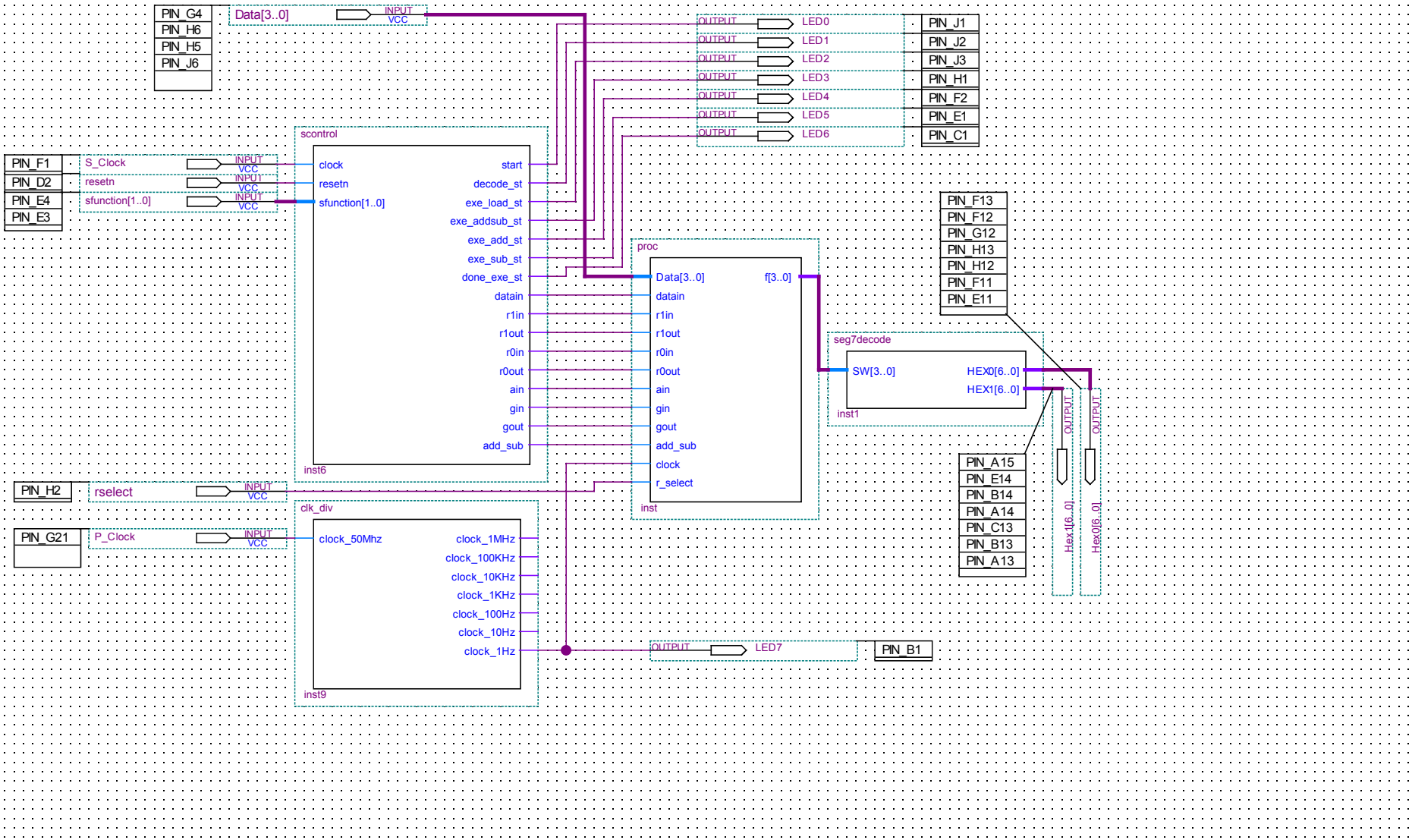
Optional: Can use this VHDL code for Clock_div.

Code Clk_div

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY clk_div IS

	PORT
	(
		clock_50Mhz			: IN	STD_LOGIC;
		clock_1MHz			: OUT	STD_LOGIC;
		clock_100KHz		: OUT	STD_LOGIC;
		clock_10KHz			: OUT	STD_LOGIC;
		clock_1KHz			: OUT	STD_LOGIC;
		clock_100Hz			: OUT	STD_LOGIC;
		clock_10Hz			: OUT	STD_LOGIC;
		clock_1Hz			: OUT	STD_LOGIC);

END clk_div;

ARCHITECTURE a OF clk_div IS

	SIGNAL count_1Mhz: STD_LOGIC_VECTOR(6 DOWNTO 0);
	SIGNAL count_100Khz, count_10Khz, count_1Khz : STD_LOGIC_VECTOR(2 DOWNTO 0);
	SIGNAL count_100hz, count_10hz, count_1hz : STD_LOGIC_VECTOR(2 DOWNTO 0);
	SIGNAL  clock_1Mhz_int, clock_100Khz_int, clock_10Khz_int, clock_1Khz_int: STD_LOGIC;
	SIGNAL clock_100hz_int, clock_10Hz_int, clock_1Hz_int : STD_LOGIC;
BEGIN
	PROCESS
	BEGIN
-- Divide by 50
		WAIT UNTIL clock_50Mhz'EVENT and clock_50Mhz = '1';
			IF count_1Mhz < 50 THEN
				count_1Mhz <= count_1Mhz + 1;
			ELSE
				count_1Mhz <= "0000000";
			END IF;
			IF count_1Mhz < 25 THEN
				clock_1Mhz_int <= '0';
			ELSE
				clock_1Mhz_int <= '1';
			END IF;

		-- Ripple clocks are used in this code to save prescalar hardware
		-- Sync all clock prescalar outputs back to master clock signal
			clock_1Mhz <= clock_1Mhz_int;
			clock_100Khz <= clock_100Khz_int;
			clock_10Khz <= clock_10Khz_int;
```

```
                clock_1Khz <= clock_1Khz_int;
                clock_100hz <= clock_100hz_int;
                clock_10hz <= clock_10hz_int;
                clock_1hz <= clock_1hz_int;
END PROCESS;

        -- Divide by 10
PROCESS
BEGIN
        WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
                IF count_100Khz /= 4 THEN
                        count_100Khz <= count_100Khz + 1;
                ELSE
                        count_100khz <= "000";
                        clock_100Khz_int <= NOT clock_100Khz_int;
                END IF;
END PROCESS;

        -- Divide by 10
PROCESS
BEGIN
        WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
                IF count_10Khz /= 4 THEN
                        count_10Khz <= count_10Khz + 1;
                ELSE
                        count_10khz <= "000";
                        clock_10Khz_int <= NOT clock_10Khz_int;
                END IF;
END PROCESS;

        -- Divide by 10
PROCESS
BEGIN
        WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
                IF count_1Khz /= 4 THEN
                        count_1Khz <= count_1Khz + 1;
                ELSE
                        count_1khz <= "000";
                        clock_1Khz_int <= NOT clock_1Khz_int;
                END IF;
END PROCESS;

        -- Divide by 10
PROCESS
BEGIN
        WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
                IF count_100hz /= 4 THEN
                        count_100hz <= count_100hz + 1;
                ELSE
                        count_100hz <= "000";
                        clock_100hz_int <= NOT clock_100hz_int;
```

```
                        END IF;
        END PROCESS;

                -- Divide by 10
        PROCESS
        BEGIN
                WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
                        IF count_10hz /= 4 THEN
                                count_10hz <= count_10hz + 1;
                        ELSE
                                count_10hz <= "000";
                                clock_10hz_int <= NOT clock_10hz_int;
                        END IF;
        END PROCESS;

                -- Divide by 10
        PROCESS
        BEGIN
                WAIT UNTIL clock_10hz_int'EVENT and clock_10hz_int = '1';
                        IF count_1hz /= 4 THEN
                                count_1hz <= count_1hz + 1;
                        ELSE
                                count_1hz <= "000";
                                clock_1hz_int <= NOT clock_1hz_int;
                        END IF;
        END PROCESS;

END a;
```


PIN_G4
PIN_H6
PIN_H5
PIN_J6
Data[3..0]
INPUT
VCC
OUTPUT
LED0
LED1
LED2
LED3
LED4
LED5
LED6
PIN_J1
PIN_J2
PIN_J3
PIN_H1
PIN_F2
PIN_E1
PIN_C1
scontrol
PIN_F1
PIN_D2
PIN_E4
PIN_E3
S_Clock
resetn
sfunction[1..0]
clock
resetn
sfunction[1..0]
start
decode_st
exe_load_st
exe_addsub_st
exe_add_st
exe_sub_st
done_exe_st
datain
r1in
r1out
r0in
r0out
ain
gin
gout
add_sub
inst6
proc
Data[3..0]
datain
r1in
r1out
r0in
r0out
ain
gin
gout
add_sub
clock
r_select
f[3..0]
inst
PIN_F13
PIN_F12
PIN_G12
PIN_H13
PIN_H12
PIN_F11
PIN_E11
seg7decode
SW[3..0]
HEX0[6..0]
HEX1[6..0]
inst1
PIN_A15
PIN_E14
PIN_B14
PIN_A14
PIN_C13
PIN_B13
PIN_A13
Hex1[6..0]
Hex0[6..0]
PIN_H2
rselect
PIN_G21
P_Clock
clk_div
clock_50Mhz
clock_1MHz
clock_100KHz
clock_10KHz
clock_1KHz
clock_100Hz
clock_10Hz
clock_1Hz
inst9
LED7
PIN_B1