APPENDIX

VHDL CODE FOR THE TL/OSL SYSTEM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.All;
library UNISIM;
use UNISIM.VComponents.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity top is
   Port ( clk50 : in  STD_LOGIC;--100MHz

                              --RS232 pins-----------------
                                serial_out : out  STD_LOGIC;
                                serial_in : in std_logic;
                              --ADC pins------------------
                              SDATA1 : IN std_logic;
                              SDATA2 : IN std_logic;
                              SCLK : OUT std_logic;
                              nCS : OUT std_logic;
                              blue: OUT std_logic;
                              ir: OUT std_logic;
                              ck_dac: OUT std_logic;
                              n_sync: OUT std_logic;
                              --H-bridge signals---
                                pwm_c : out  STD_LOGIC;
                                pwm_d : out  STD_LOGIC;
                                en : out std_logic;
                              emccd : OUT std_logic
                                        );
end top;

architecture Behavioral of top is


COMPONENT main--rs232
        PORT(
                clk : IN std_logic;
                serial_in : IN std_logic;
                serial_out : OUT std_logic;
                new_comd: out std_logic;
                new_comd_o: out std_logic;
```

```
            over: in std_logic;
            over_o: in std_logic;
            ramp_in : IN integer range 0 to 16383;
            result : IN std_logic_vector(15 downto 0);
            f_temp : OUT integer range 0 to 16383;
            rate : OUT integer range 0 to 1048575;--66000;
            holdtime : OUT integer range 0 to 3600;
            feedbck: out integer range 0 to 16383;
            result2: in std_logic_vector(15 downto 0);--adc o/p 2; OSL:
            light_final : out integer range 0 to 16383;
            light_rate: out integer range 0 to 1048575;--66000;
            lgt_holdtime: out integer range 0 to 3600;
            ramp_lig_in: in integer range 0 to 16383;
            filter_fb: in integer range 0 to 16383;
            emccd: out std_logic
            );
    END COMPONENT;


COMPONENT ADC
    PORT(
            CLK : IN std_logic;
            SDATA1 : IN std_logic;
            SDATA2 : IN std_logic;
            debug : OUT std_logic;
            SCLK : OUT std_logic;
            nCS : OUT std_logic;
            temp1 : OUT std_logic_vector(15 downto 0);
            light2 : out std_logic_vector(15 downto 0);
            light_opti : out std_logic_vector(15 downto 0)
                        );
    END COMPONENT;

COMPONENT DA2RefComp
    PORT(
            CLK : IN std_logic;
            RST : IN std_logic;
            DATA1 : IN std_logic_vector(11 downto 0);
            DATA2 : IN std_logic_vector(11 downto 0);
            START : IN std_logic;
            D1 : OUT std_logic;
            D2 : OUT std_logic;
            CLK_OUT : OUT std_logic;
            nSYNC : OUT std_logic;
            DONE : OUT std_logic
            );
    END COMPONENT;

COMPONENT thermal
    PORT(
```

```
		clk : IN std_logic;
		ck_adc: in std_logic;
		en : out std_logic;
		final_temp : IN integer range 0 to 16383;
		start : IN std_logic;
		rate : IN integer range 0 to 1048575;--66000;
		hold_time : IN integer range 0 to 3600;
		feedback : IN integer range 0 to 16383;
		filter_fb: out integer range 0 to 16383;-------------------------
		new_comd : IN std_logic;
		ramp_rs : OUT integer range 0 to 16383;
		ramp_t : out  integer range 0 to 16383;
		err : out  integer range 0 to 16383;--------ramp_dac: OUT integer range 0 to
4095;
		adc_no: out std_logic;
		pwm1 : OUT std_logic;
		pwm2 : OUT std_logic;
		st : out std_logic;
		over : OUT std_logic;
		debug1 : out std_logic
		);
	END COMPONENT;

	COMPONENT optical
	PORT(
		clk : IN std_logic;
		final_intensity : IN integer range 0 to 16383;
		rate : IN integer range 0 to 1048575;--66000;
		hold_time : IN integer range 0 to 3600;
		adc_sp: in std_logic;--start of conv for adc
		--feedback : IN integer range 0 to 16383;
		new_comd : IN std_logic;
		ramp_rs : OUT integer range 0 to 16383;
		ramp_dac: OUT integer range 0 to 4095;
		adc : in integer range 0 to 65535;
		--dac_pw: OUT std_logic;--power on-off for dac
		over : OUT std_logic
		);
END COMPONENT;

signal clk_50, ck50 : std_logic;
signal clk_spi,ck_50,start_ck,new_comd_s,over_s, ad_conv_s, pwm_s,adc_ck: std_logic;
signal fl,f_temp_s,ramp_in_s, feedback_s,light_final_i,lig_ramp_rs,rt,filter_fbs: integer range
0 to 16383;
signal holdtime_s, htl,lgt_holdtime_i : integer range 0 to 3600;
signal adc_temp,rampt, adc_lgt,lgt_opti: std_logic_vector(15 downto 0);
signal ramp_dac,cnt: integer range 0 to 4095;
signal mosi_adc, mosi_dac, amp_cs_c, cs_s: std_logic;
signal n_cmd_o, ovr_o,st, reached: std_logic;
signal adc_i: integer range 0 to 65535;
```

```
signal emccd_f: std_logic := '0';
signal cnt_t: integer range 0 to 2505;
signal light_rate_i,rate_s: integer range 0 to 1048575 := 0;--fffff ~1000 000 ms--tim_s,
signal t1, lig, lig_dac : std_logic_vector(11 downto 0);
signal sta, dne : std_logic;
signal at, al : std_logic_vector(13 downto 0);

begin

process (clk50) --making a 50MHz clk form a 100MHz clk
begin
if clk50'event and clk50 = '1' then
        ck50 <= not ck50;
end if;
end process;

adc_i <= CONV_INTEGER(adc_lgt); --adc_lgt lgt_opti
lig_dac <= CONV_STD_LOGIC_VECTOR(ramp_dac,12);

 BUFG_inst : BUFG
  port map (
    O => clk_50,     -- Clock buffer output
    I => ck50      -- Clock buffer input
  );


        RS232: main PORT MAP(
                clk => clk_50,
                serial_out => serial_out,
                serial_in => serial_in,
                f_temp => f_temp_s,--temp  out 16383
                rate => rate_s, --out 1048575
                holdtime => holdtime_s, --out 3600
                ramp_in => ramp_in_s,-- in 16383
                result => adc_temp,--in log_vec 16 bits, temp
                new_comd => new_comd_s,
                new_comd_o => n_cmd_o,
                over => over_s,
                over_o => ovr_o,
                feedbck => feedback_s,--int of result-- out 16383
                result2 => adc_lgt,--adc_light,--rampt,-- --in log_vec 16 bits, light
                light_final => light_final_i,
                light_rate => light_rate_i,
                lgt_holdtime => lgt_holdtime_i,
                ramp_lig_in => lig_ramp_rs,     --in 16383 osl display       --ramp for osl
        light_rate_i,
                filter_fb => filter_fbs,
                --tim => tim_s,
                emccd => emccd
        );
```

```
	Feedback: ADC PORT MAP(
		CLK => clk_50,
		debug => open,
		SDATA1 => SDATA1,
		SDATA2 => SDATA2,
		SCLK => SCLK,
		nCS => nCS,
		temp1 => adc_temp,
		light2 => adc_lgt,
		light_opti => lgt_opti
	);


	DAC: DA2RefComp PORT MAP(
	--General usage
		CLK => clk_50,                                  -- System Clock (50MHz)
		RST => '0',
	 --Pmod interface signals
		D1 => blue,
		D2 => ir,
		CLK_OUT => ck_dac,
		nSYNC => n_sync,
	--User interface signals
		DATA1 => lig_dac,
		DATA2 => "000000000000",
		START => sta,
		DONE => dne
	);

process (clk_50) --making a 50MHz clk form a 100MHz clk
begin
if clk_50'event and clk_50 = '1' then
	sta <= dne;
end if;
end process;

	TL : thermal PORT MAP(
		clk => clk_50,
		ck_adc => dne,--adc_ck,
		en => en,
		final_temp => f_temp_s,
		start => '0',--tc_open, pwm will stop if it is '1'
		rate => rate_s,
		hold_time => holdtime_s,
		ramp_rs => open,--ramp_in_s,
		ramp_t => ramp_in_s,--open,--rt,
		err => open,--lig_ramp_rs,--   ramp_dac => ramp_dac,
		feedback => feedback_s,
```

```
                filter_fb => filter_fbs,
                adc_no => pwm_s,
                pwm1 => pwm_c,
                pwm2 => pwm_d,
                --e => open,--enable2,
                new_comd => new_comd_s,
                st => open,--dat_out(14),
                over => over_s,
                debug1 => open--debug1
        );

        OSL : optical PORT MAP(
                clk => clk_50,
                final_intensity => light_final_i,
                rate => light_rate_i,----------
                hold_time => lgt_holdtime_i,----------
                ramp_rs => lig_ramp_rs,
                ramp_dac => ramp_dac,
                adc_sp => dne,
                new_comd => n_cmd_o,--new_comd_s,--n_cmd_o,--
                adc => adc_i,
                over => ovr_o
        );
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.All;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity main is
   Port ( clk : in  STD_LOGIC;--50MHz
          serial_out : out  STD_LOGIC;
                        serial_in : in std_logic;
                        new_comd: out std_logic;
                        new_comd_o: out std_logic;
                        over: in std_logic;
                        over_o: in std_logic;
                       -- data_cnt: in integer range 0 to 39;
                        f_temp : out integer range 0 to 16383;
                        rate: out integer range 0 to 1048575;--66000;
                        holdtime: out integer range 0 to 3600;
                        ramp_in: in integer range 0 to 16383;
                        result: in std_logic_vector(15 downto 0);--adc_temp
                        result2: in std_logic_vector(15 downto 0);--adc_light
                        ramp_lig_in: in integer range 0 to 16383;--osl ramp display
                        feedbck: out integer range 0 to 16383;
```

```
                        light_final : out integer range 0 to 16383;
                        light_rate: out integer range 0 to 1048575;--66000;
                        lgt_holdtime: out integer range 0 to 3600;
                        filter_fb: in integer range 0 to 16383;
                       -- tim : in integer range 0 to 1048575 := 0;
                        emccd: out std_logic
        );
end main;

architecture Behavioral of main is

signal en_16_x_baud,wr,buffer_full: std_logic;
signal baud_count: integer range 0 to 27 := 0;
signal flag, emccd_f: std_logic;
signal dat_transfer,data_lsb, data_msb,data_lsb0,data_msb0, cs_rp, cs_fd,
rp0,rp1,rp2,rp3,fd0,fd1,fd2,fd3  : std_logic_vector(7 downto 0);
signal cs_lig_r,lig_r0,lig_r1,lig_r2,lig_r3  : std_logic_vector(7 downto 0);--, lig_r4
signal lig_fd0,lig_fd1,lig_fd2,lig_fd3,lig_fd : std_logic_vector(7 downto 0);
signal lsb_msb : std_logic := '0';
signal data3 : std_logic_vector(16 downto 0):= "00000000000000000";
signal data1, data2, adc,light_final_v :  std_logic_vector(15 downto 0);
signal cnt : integer range 0 to 30;--13
signal final_temp_i,ramp_t: integer range 0 to 16383;
signal feedback,fb1, fb2, fb3, fb_final: integer range 0 to 65535;
signal hold_time_i: integer range 0 to 3600;
signal rate_i : integer range 0 to 66000;
signal temp_final_v, ramp,r1: std_logic_vector(15 downto 0);
signal holdtime_v,lgt_holdtime_v: std_logic_vector(11 downto 0);--,light_final_v
signal rp_msb : std_logic_vector(7 downto 0);
signal temp_rate_tv,ramp_lig: std_logic_vector(15 downto 0);
signal feedback_ac0,feedback_ac1,feedback_ac2,feedback_ac3,
ramp_ac0,ramp_ac1,ramp_ac2,ramp_ac3:std_logic_vector(7 downto 0);
signal feedback_lig0,feedback_lig1,feedback_lig2,feedback_lig3,
ramp_lig0,ramp_lig1,ramp_lig2,ramp_lig3:std_logic_vector(7 downto 0);
signal cnt_adc : integer range 0 to 2;
signal cnt_t : integer range 0 to 2508;
--signal tims : integer range 0 to 1048575 := 0;
signal light_rate_v, temp_rate_v  : std_logic_vector(19 downto 0);--timv,


COMPONENT uart_rx
        PORT(
                serial_in : IN std_logic;
                en_16_x_baud : IN std_logic;
                clk : IN std_logic;
                data_out : OUT std_logic_vector(7 downto 0);
                buffer_full : OUT std_logic;
                buffer_half_full : OUT std_logic;
                light_final : OUT std_logic_vector(15 downto 0);
                light_rate : OUT std_logic_vector(19 downto 0);
```

```
                lgt_holdtime : OUT std_logic_vector(11 downto 0);
                temp_final : OUT std_logic_vector(15 downto 0);
                temp_rate : OUT std_logic_vector(19 downto 0);
                holdtime : OUT std_logic_vector(11 downto 0);
                new_comd: out std_logic;
                new_comd_o: out std_logic;
                over : in std_logic;
                over_o : in std_logic;
                fb : in std_logic_vector(15 downto 0)
                );
        END COMPONENT;

COMPONENT uart_tx
        PORT(
                data_in : IN std_logic_vector(7 downto 0);
                write_buffer : IN std_logic;
                reset_buffer : IN std_logic;
                en_16_x_baud : IN std_logic;
                clk : IN std_logic;
                serial_out : OUT std_logic;
                buffer_full : OUT std_logic;
                buffer_half_full : OUT std_logic
                );
        END COMPONENT;


COMPONENT in2slv--to convert logic vector into integer
        PORT(
                clk : in std_logic;
        --      data_cnt: in integer range 0 to 39;
                temp_final : IN std_logic_vector(15 downto 0);
                temp_rate : IN std_logic_vector(19 downto 0);
                holdtime : IN std_logic_vector(11 downto 0);
                result : in std_logic_vector(15 downto 0);---adc o/p
                feedback : out integer range 0 to 65535;--16383;--feedback
                final_temp : OUT integer range 0 to 16383;--std_logic_vector(0 to 13);
                hold_time : OUT integer range 0 to 3600;--std_logic_vector(0 to 11);
                rate : OUT integer range 0 to 1048575;--66000;--std_logic_vector(0 to 15)
                light_final : IN std_logic_vector(15 downto 0);
                light_rate : IN std_logic_vector(19 downto 0);
                lgt_holdtime : IN std_logic_vector(11 downto 0);
                light_final_i : OUT integer range 0 to 16383;--std_logic_vector(0 to 13);
                light_rate_i : OUT integer range 0 to 1048575;--66000;--light_rate_i : OUT
integer range 0 to 3600;--std_logic_vector(0 to 11);
                lgt_holdtime_i : OUT integer range 0 to 3600--std_logic_vector(0 to 15)
                );
END COMPONENT;

COMPONENT hex_2_asciihex
        PORT(
```

```
		clk : in std_logic;
		lsb0_ri : IN std_logic_vector(3 downto 0);
		lsb1_ri : IN std_logic_vector(3 downto 0);
		msb0_ri : IN std_logic_vector(3 downto 0);
		msb1_ri : IN std_logic_vector(3 downto 0);
		lsb0_fi : IN std_logic_vector(3 downto 0);
		lsb1_fi : IN std_logic_vector(3 downto 0);
		msb0_fi : IN std_logic_vector(3 downto 0);
		msb1_fi : IN std_logic_vector(3 downto 0);
		lsb0_ro : OUT std_logic_vector(7 downto 0);
		lsb1_ro : OUT std_logic_vector(7 downto 0);
		msb0_ro : OUT std_logic_vector(7 downto 0);
		msb1_ro : OUT std_logic_vector(7 downto 0);
		lsb0_fo : OUT std_logic_vector(7 downto 0);
		lsb1_fo : OUT std_logic_vector(7 downto 0);
		msb0_fo : OUT std_logic_vector(7 downto 0);
		msb1_fo : OUT std_logic_vector(7 downto 0)
		);
	END COMPONENT;

COMPONENT hex_to_asciihex
	PORT(
		clk : in std_logic;
		lsb0_ri : IN std_logic_vector(3 downto 0);
		lsb1_ri : IN std_logic_vector(3 downto 0);
		msb0_ri : IN std_logic_vector(3 downto 0);
		msb1_ri : IN std_logic_vector(3 downto 0);
		lsb0_fi : IN std_logic_vector(3 downto 0);
		lsb1_fi : IN std_logic_vector(3 downto 0);
		msb0_fi : IN std_logic_vector(3 downto 0);
		msb1_fi : IN std_logic_vector(3 downto 0);
		lsb0_ro : OUT std_logic_vector(7 downto 0);
		lsb1_ro : OUT std_logic_vector(7 downto 0);
		msb0_ro : OUT std_logic_vector(7 downto 0);
		msb1_ro : OUT std_logic_vector(7 downto 0);
		lsb0_fo : OUT std_logic_vector(7 downto 0);
		lsb1_fo : OUT std_logic_vector(7 downto 0);
		msb0_fo : OUT std_logic_vector(7 downto 0);
		msb1_fo : OUT std_logic_vector(7 downto 0)
		);
	END COMPONENT;

begin


f_temp <= final_temp_i;
rate <= rate_i;
holdtime <= hold_time_i;
ramp_t <= ramp_in;
--result <= adc_op;
```

```
feedbck <= feedback;

----------receive and transmit module of rs232----------

rx: uart_rx PORT MAP(
                serial_in => serial_in,
                data_out => open,--data_out,
                en_16_x_baud => en_16_x_baud,
                buffer_full => open,
                buffer_half_full => open ,
                clk => clk,
                temp_final => temp_final_v,
                temp_rate =>  temp_rate_v,
                holdtime => holdtime_v,
                light_final => light_final_v,
                light_rate => light_rate_v,
                lgt_holdtime => lgt_holdtime_v,
                new_comd => new_comd,
                new_comd_o => new_comd_o,
                over => over,
                over_o => over_o,
                fb => adc
        );


tx: uart_tx PORT MAP(
                data_in => dat_transfer,
                write_buffer => wr,
                reset_buffer => '0',
                en_16_x_baud => en_16_x_baud,
                serial_out => serial_out,
                buffer_full => buffer_full,
                buffer_half_full => open,
                clk => clk
        );

--------------------------------------------------------------------------------
--data conversion modules: integer into std logic vector and vica-versa; hex ascii into number
and vica-versa-----
ramp <= CONV_STD_LOGIC_VECTOR(ramp_t, 16);--converts integer into std logic vector
adc <= CONV_STD_LOGIC_VECTOR(filter_fb,16);--filtered adc o/p
ramp_lig <= CONV_STD_LOGIC_VECTOR(ramp_lig_in,16);

lgc_vctr2int: in2slv PORT MAP(
                clk => clk,
        --      data_cnt => data_cnt,
                final_temp => final_temp_i,
                hold_time => hold_time_i,
                rate => rate_i,
                result => result,
```

```
                feedback => feedback,--feedback
                temp_final => temp_final_v,
                temp_rate => temp_rate_v,
                holdtime => holdtime_v,
                light_final => light_final_v,
                light_rate => light_rate_v,
                lgt_holdtime => lgt_holdtime_v,
                light_final_i => light_final,
                light_rate_i => light_rate,
                lgt_holdtime_i => lgt_holdtime
        );


hex_to_asciihex_temp: hex_2_asciihex PORT MAP(
                clk => clk,
                lsb0_ri => ramp(15 downto 12),
                lsb1_ri => ramp(11 downto 8),
                msb0_ri => ramp(7 downto 4),
                msb1_ri => ramp(3 downto 0),
                lsb0_fi => data_msb(7 downto 4),
                lsb1_fi => data_msb(3 downto 0),
                msb0_fi => data_lsb(7 downto 4),
                msb1_fi => data_lsb(3 downto 0),
                lsb0_ro => ramp_ac0,
                lsb1_ro => ramp_ac1,
                msb0_ro => ramp_ac2,
                msb1_ro => ramp_ac3,
                lsb0_fo => feedback_ac0,
                lsb1_fo => feedback_ac1,
                msb0_fo => feedback_ac2,
                msb1_fo => feedback_ac3
        );


hex_to_asciihex_light: hex_to_asciihex PORT MAP(
                clk => clk,
                lsb0_ri => ramp_lig(15 downto 12),
                lsb1_ri => ramp_lig(11 downto 8),
                msb0_ri => ramp_lig(7 downto 4),
                msb1_ri => ramp_lig(3 downto 0),
                lsb0_fi => result2(15 downto 12),
                lsb1_fi => result2(11 downto 8),
                msb0_fi => result2(7 downto 4),
                msb1_fi => result2(3 downto 0),
                lsb0_ro => ramp_lig0,
                lsb1_ro => ramp_lig1,
                msb0_ro => ramp_lig2,
                msb1_ro => ramp_lig3,
                lsb0_fo => feedback_lig0,
                lsb1_fo => feedback_lig1,
```

```
		msb0_fo => feedback_lig2,
		msb1_fo => feedback_lig3
	);

baud_timer :process (clk)
begin
if (clk = '1' and clk'event) then
	if baud_count = 27 then
		baud_count <= 0;
		en_16_x_baud <= '1';
	else
		baud_count <= baud_count + 1;
		en_16_x_baud <= '0';
	end if;
end if;
end process baud_timer;

process (clk)--latching the data of ser2pall at 28, need to work out at what no. to latch for 2
ch of adc
begin
if (clk = '0' and clk'event) then-- +ve edge
		data_lsb <= adc(7 downto 0);--
		data_msb <= adc(15 downto 8);--result(15 downto 8);--
end if;
end process;

adc2rs:process (clk)--clk50
begin
if (clk = '0' and clk'event) then-- -ve edge

	if buffer_full = '0' then
		wr <= '1';
		if cnt < 27 then--13--27
			cnt <= cnt + 1;
		else
			cnt <= 0;
		end if;
-------------------------temp ramp transfer---------------------------------
		if cnt = 0 then
			dat_transfer <= "00100011";--# start bit
		elsif cnt = 1 then
			dat_transfer <= "01000111";--1 for ramp -- ascii G
		elsif cnt = 2 then
			dat_transfer <= ramp_ac0;--"00110011";--
			rp0 <= ramp_ac0;
			rp1 <= ramp_ac1;
			rp2 <= ramp_ac2;
			rp3 <= ramp_ac3;
--			cs_rp <= not(rp0 + rp1 + rp2 + rp3) + '1';
		elsif cnt = 3 then
```

```
                dat_transfer <= rp1;
            elsif cnt = 4 then
                dat_transfer <= rp2;
            elsif cnt = 5 then
                dat_transfer <= rp3;
                cs_rp <= not(rp0 + rp1 + rp2 + rp3) + '1';
            elsif cnt = 6 then
                dat_transfer <= cs_rp;
        --  end if;

----------------------------temp feedback transfer----------------------------
            elsif cnt = 7 then
                dat_transfer <= "00100011";--# start bit
            elsif cnt = 8 then
                dat_transfer <= "01001000";--2 for feedback 0 to 250--ascii H
            elsif cnt = 9 then
                dat_transfer <= feedback_ac0;
                fd0 <= feedback_ac0;
                fd1 <= feedback_ac1;
                fd2 <= feedback_ac2;
                fd3 <= feedback_ac3;
                cs_fd <= not(fd0 + fd1 + fd2 + fd3) + '1';
            elsif cnt = 10 then
                dat_transfer <= fd1;
            elsif cnt = 11 then
                dat_transfer <= fd2;
            elsif cnt = 12 then
                dat_transfer <= fd3;
            elsif cnt = 13 then
                dat_transfer <= cs_fd;
            --end if;
----------------------------lig ramp transfer----------------------------
            elsif cnt = 14 then
                dat_transfer <= "00100011";--# start bit
            elsif cnt = 15 then
                dat_transfer <= "01001001";----ascii I
            elsif cnt = 16 then
                dat_transfer <= ramp_lig0;--feedback_lig0;
                lig_r0 <= ramp_lig0;--feedback_lig0;
                lig_r1 <= ramp_lig1;--feedback_lig1;
                lig_r2 <= ramp_lig2;--feedback_lig2;
                lig_r3 <= ramp_lig3;--feedback_lig3;
            --  lig_r4 <= feedback_lig0;
                --cs_lig_r <= not(lig_r0 + lig_r1 + lig_r2 + lig_r3) + '1';
            elsif cnt = 17 then
                dat_transfer <= lig_r1;
            elsif cnt = 18 then
                dat_transfer <= lig_r2;
            elsif cnt = 19 then
                dat_transfer <= lig_r3;
```

```
                        cs_lig_r <= not(lig_r0 + lig_r1 + lig_r2 + lig_r3) + '1';
                elsif cnt = 20 then
                        --dat_transfer <= lig_r4;
                        cs_lig_r <= not(lig_r0 + lig_r1 + lig_r2 + lig_r3) + '1';-- + lig_r4
                        dat_transfer <= cs_lig_r;
--              end if;
----------------------------light feedback transfer----------------------------
                elsif cnt = 21 then
                        dat_transfer <= "00100011";--# start bit
                elsif cnt = 22 then
                        dat_transfer <= "01001010";--ascii J
                elsif cnt = 23 then
                        dat_transfer <= feedback_lig0;
                        lig_fd0 <= feedback_lig0;
                        lig_fd1 <= feedback_lig1;
                        lig_fd2 <= feedback_lig2;
                        lig_fd3 <= feedback_lig3;
                        lig_fd <= not(lig_fd0 + lig_fd1 + lig_fd2 + lig_fd3) + '1';
                elsif cnt = 24 then
                        dat_transfer <= lig_fd1;
                elsif cnt = 25 then
                        dat_transfer <= lig_fd2;
                elsif cnt = 26 then
                        dat_transfer <= lig_fd3;
                elsif cnt = 27 then
                        dat_transfer <= lig_fd;
                end if;

--      ------------------------------------------------------------------------
        else
                wr <= '0';
        end if;
end if;
end process;

process (clk)--clk50
begin
if (clk = '0' and clk'event) then---PUT HERE  INSTEAD ramp_in OF AND VERIFY
        if filter_fb > 491 then--491 1091  ramp_in
                emccd_f <= '1';
        elsif ramp_in < 370 then--feedback
                emccd_f <= '0';
                cnt_t <= 0;
        else
                emccd_f <= emccd_f;
        end if;


        if emccd_f = '1' and cnt_t < 2449 then
                emccd <= '0';--1
```

```
                cnt_t <= cnt_t + 1;
        else
                emccd <= '1';--0
        end if;

end if;
end process;

end Behavioral;
```

```
----------------------------------------------------------------------------|
-- UART Receiver with integral 16 byte FIFO buffer
--
-- 8 bit, no parity, 1 stop bit
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;
--
----------------------------------------------------------------------------
--
-- Main Entity for UART_RX
--
entity uart_rx is
   Port (          serial_in : in std_logic;
                  data_out : out std_logic_vector(7 downto 0);
                   en_16_x_baud : in std_logic;
                    buffer_full : out std_logic;
                    buffer_half_full : out std_logic;
                   clk : in std_logic;
                    temp_final: out std_logic_vector(15 downto 0);
                   temp_rate: out std_logic_vector(19 downto 0);--15 - 0
                   holdtime: out std_logic_vector(11 downto 0);
                   light_final: out std_logic_vector(15 downto 0);
                   light_rate: out std_logic_vector(19 downto 0);
                   lgt_holdtime: out std_logic_vector(11 downto 0);
                   new_comd: out std_logic:= '0';
                   new_comd_o: out std_logic:= '0';
                   over : in std_logic:='0';--1
                   over_o: in std_logic:= '0';
                   fb : in std_logic_vector(15 downto 0)
                                                          );
            end uart_rx;
--
----------------------------------------------------------------------------
--
-- Start of Main Architecture for UART_RX
--
```

```
architecture macro_level_definition of uart_rx is
--
-------------------------------------------------------------------------------
--
-- Components used in UART_RX and defined in subsequent entities.
--
-------------------------------------------------------------------------------
--
-- Constant (K) Compact UART Receiver
--
component kcuart_rx
   Port (      serial_in : in std_logic;
               data_out : out std_logic_vector(7 downto 0);
            data_strobe : out std_logic;
           en_16_x_baud : in std_logic;
                  clk : in std_logic);
   end component;
--
-- 'Bucket Brigade' FIFO
--
component rxfifo_16x8
   Port (       data_in : in std_logic_vector(7 downto 0);
              data_out : out std_logic_vector(7 downto 0);
                 reset : in std_logic;
                 writ : in std_logic;
                  reed : in std_logic;
                  full : out std_logic;
             half_full : out std_logic;
          data_present : out std_logic;
                                          clk : in std_logic);
   end component;

signal uart_data_out      : std_logic_vector(7 downto 0);
signal fifo_write, flag, n_flag, data_present, flag_r: std_logic;
signal baud_count  : integer range 0 to 27 := 0;
signal read_buffer, buffer_half_full_s, reset_buffer  : std_logic;
signal cnt  : integer range 0 to 25 := 0;
signal dat_out,  temp_rate_1,temp_rate_2,temp_rate_3,temp_rate_4, hold_time :
std_logic_vector(7 downto 0);
signal
temp_rate_5,temp_rate_6,temp_rate_7,temp_rate_8,temp_rate_9,temp_rate_10,temp_rate_11
,temp_rate_12,temp_rate_13,temp_rate_14 : std_logic_vector(7 downto 0);
signal
temp_rate_16,temp_rate_17,temp_rate_18,temp_rate_19,temp_rate_20,temp_rate_21,temp_r
ate_22,temp_rate_23: std_logic_vector(7 downto 0);
signal temp_rate_24,temp_rate_25: std_logic_vector(7 downto 0);
signal temp_rate_0,temp_rate_15, comp,temp_f1,
temp_f2,temp_f3,light_f1,light_f2,lgt_f2,light_f3, cs: std_logic_vector(7 downto 0);
signal tep0,tep1,tep2,tep3,tep4,tep5,tep6,tep7,tep8,tep9,tep10,tep11,tep12,tep13,tep14,tep15:
std_logic_vector(3 downto 0);
```

```
signal tep16,tep17,tep18,tep19,tep20,tep21,tep22,tep23,tep24,tep25: std_logic_vector(3
downto 0);
signal temp_f, temp_r, comp1 :std_logic_vector(15 downto 0);
signal temp_h, comp2,temp_f4,light_f4 :std_logic_vector(11 downto 0);
signal cnt_f : integer range 0 to 1 := 0;
signal nc_osl, new_comd_os: std_logic;
signal tf,tr: std_logic_vector(15 downto 0);
signal tr_i: integer range 0 to 65535;
signal ft: integer range 0 to 4095;
signal rate2:std_logic_vector(7 downto 0);

--- ----------------------------------------------------------------------------
--
-- Start of UART_RX circuit description
--
-------------------------------------------------------------------------------
--
begin
buffer_half_full <= buffer_half_full_s;

  kcuart: kcuart_rx
  port map (     serial_in => serial_in,
                 data_out => uart_data_out,
              data_strobe => fifo_write,
             en_16_x_baud => en_16_x_baud,
                    clk => clk );


  buf: rxfifo_16x8
  port map (       data_in => uart_data_out,
                  data_out => dat_out,--data_t,
                     reset => '0',--reset_buffer,
                     writ => fifo_write,
                      reed => read_buffer,
                      full => buffer_full,
                 half_full => buffer_half_full_s,
             data_present => data_present,--buffer_data_present,
                    clk => clk);

process(clk)
begin
if (clk = '0' and clk'event) then
          if data_present = '1' then--if data_present = '1' then
                    read_buffer <= '1';
                    if cnt = 0 then
                              temp_rate_0 <= dat_out;
                    elsif cnt = 1 then
                              temp_rate_1 <= dat_out;
                    elsif cnt = 2 then
                              temp_rate_2 <= dat_out;
```

```
        elsif cnt = 3 then
                temp_rate_3 <= dat_out;
        elsif cnt = 4 then
                temp_rate_4 <= dat_out;
        elsif cnt = 5 then
                temp_rate_5 <= dat_out;
        elsif cnt = 6 then
                temp_rate_6 <= dat_out;
        elsif cnt = 7 then
                temp_rate_7 <= dat_out;
        elsif cnt = 8 then
                temp_rate_8 <= dat_out;
        elsif cnt = 9 then
                temp_rate_9 <= dat_out;
        elsif cnt = 10 then
                temp_rate_10 <= dat_out;
        elsif cnt = 11 then
                temp_rate_11 <= dat_out;
        elsif cnt = 12 then
                temp_rate_12 <= dat_out;
        elsif cnt = 13 then
                temp_rate_13 <= dat_out;
        elsif cnt = 14 then
                temp_rate_14 <= dat_out;
        elsif cnt = 15 then
                temp_rate_15 <= dat_out;
        elsif cnt = 16 then
                temp_rate_16 <= dat_out;
        elsif cnt = 17 then
                temp_rate_17 <= dat_out;
        elsif cnt = 18 then
                temp_rate_18 <= dat_out;
        elsif cnt = 19 then
                temp_rate_19 <= dat_out;
        elsif cnt = 20 then
                temp_rate_20 <= dat_out;
        elsif cnt = 21 then
                temp_rate_21 <= dat_out;
        elsif cnt = 22 then
                temp_rate_22 <= dat_out;
        elsif cnt = 23 then
                temp_rate_23 <= dat_out;
        elsif cnt = 24 then
                temp_rate_24 <= dat_out;
        elsif cnt = 25 then
                temp_rate_25 <= dat_out;
        end if;

else
        read_buffer <= '0';
```

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

dat_receive:process (clk)
begin
if (clk = '0' and clk'event) then-- -ve edge
-----------------------------------------------

        if dat_out <= "00100011" or dat_out <= "00100010" then
                new_comd <= '1';--this will start the ramp process in file ramp1
        --      nc_osl <= '1';
        elsif over = '1' then--after initializing the values for ramp this flag is set.
                new_comd <= '0';--due to this the fpga can immediatly take a new command
and discard the existing one
        --      nc_osl <= '0';
        end if;

        if dat_out <= "00100010" then
        --      new_comd <= '1';--this will start the ramp process in file ramp1
                nc_osl <= '1';
        elsif new_comd_os = '1' then
                nc_osl <= '0';
        end if;


        if nc_osl = '1' and fb >= ft and cnt = 25 then--24
                new_comd_o <= '1';
                new_comd_os <= '1';
        elsif over_o = '1' then--after initializing the values for ramp this flag is set.
                new_comd_o <= '0';--due to this the fpga can immediatly take a new
command and discard the existing one
          new_comd_os <= '0';
        else
                new_comd_o <= new_comd_os;--new_comd_os;
                new_comd_os <= new_comd_os;
        end if;
if data_present = '1' then
        if dat_out <= "00100011" or dat_out <= "00100010" then
                cnt <= 0;
        elsif cnt < 25 then--15--actually there r 24 data
                cnt <= cnt + 1;

        end if;
end if;

end if; -- for clk

end process dat_receive;
```

```
ascii_to_num: process (clk)
begin
if clk = '1' and clk'event then

        if temp_rate_0(7 downto 4) = "0011" then--Fin temp
                tep0 <= temp_rate_0(3 downto 0);
        elsif temp_rate_0(7 downto 4) = "0100" then
                tep0 <= temp_rate_0(3 downto 0) + "1001";
        end if;

        if temp_rate_1(7 downto 4) = "0011" then--Fin temp
                tep1 <= temp_rate_1(3 downto 0);
                temp_f1 <= tep0 & tep1;
        elsif temp_rate_1(7 downto 4) = "0100" then
                tep1 <= temp_rate_1(3 downto 0) + "1001";
                temp_f1 <= tep0 & tep1;
        end if;

        if temp_rate_2(7 downto 4) = "0011" then--Fin temp
                tep2 <= temp_rate_2(3 downto 0);
        elsif temp_rate_2(7 downto 4) = "0100" then
                tep2 <= temp_rate_2(3 downto 0) + "1001";
        end if;

        if temp_rate_3(7 downto 4) = "0011" then--Fin temp
                tep3 <= temp_rate_3(3 downto 0);
        elsif temp_rate_3(7 downto 4) = "0100" then
                tep3 <= temp_rate_3(3 downto 0) + "1001";
        end if;

        if temp_rate_4(7 downto 4) = "0011" then--temp rate1
                tep4 <= temp_rate_4(3 downto 0);
        elsif temp_rate_4(7 downto 4) = "0100" then
                tep4 <= temp_rate_4(3 downto 0) + "1001";
        end if;

        if temp_rate_5(7 downto 4) = "0011" then--temp rate2
                tep5 <= temp_rate_5(3 downto 0);
                temp_f3 <= tep4 & tep5;
        elsif temp_rate_5(7 downto 4) = "0100" then
                tep5 <= temp_rate_5(3 downto 0) + "1001";
                temp_f3 <= tep4 & tep5;
        end if;

        if temp_rate_6(7 downto 4) = "0011" then--temp rate3
                tep6 <= temp_rate_6(3 downto 0);
        elsif temp_rate_6(7 downto 4) = "0100" then
                tep6 <= temp_rate_6(3 downto 0) + "1001";
        end if;
```

```
if temp_rate_7(7 downto 4) = "0011" then--temp rate4
        tep7 <= temp_rate_7(3 downto 0);
        rate2 <= tep6 & tep7;
elsif temp_rate_7(7 downto 4) = "0100" then
        tep7 <= temp_rate_7(3 downto 0) + "1001";
  rate2 <= tep6 & tep7;
end if;

if temp_rate_8(7 downto 4) = "0011" then--temp rate5
        tep8 <= temp_rate_8(3 downto 0);
elsif temp_rate_8(7 downto 4) = "0100" then
        tep8 <= temp_rate_8(3 downto 0) + "1001";
end if;

if temp_rate_9(7 downto 4) = "0011" then--HT1
        tep9 <= temp_rate_9(3 downto 0);
        --temp_f2 <= tep8 & tep9;
elsif temp_rate_9(7 downto 4) = "0100" then
        tep9 <= temp_rate_9(3 downto 0) + "1001";
        --temp_f2 <= tep8 & tep9;
end if;

if temp_rate_10(7 downto 4) = "0011" then--HT2
        tep10 <= temp_rate_10(3 downto 0);
        temp_f2 <= tep9 & tep10;
elsif temp_rate_10(7 downto 4) = "0100" then
        tep10 <= temp_rate_10(3 downto 0) + "1001";
        temp_f2 <= tep9 & tep10;
end if;

if temp_rate_11(7 downto 4) = "0011" then--HT3
        tep11 <= temp_rate_11(3 downto 0);
        temp_f4 <= temp_f2 & tep11;
elsif temp_rate_11(7 downto 4) = "0100" then
        tep11 <= temp_rate_11(3 downto 0) + "1001";
        temp_f4 <= temp_f2 & tep11;
end if;

if temp_rate_12(7 downto 4) = "0011" then--fin lig
        tep12 <= temp_rate_12(3 downto 0);
        --light_f1 <= tep11 & tep12;
elsif temp_rate_12(7 downto 4) = "0100" then
        tep12 <= temp_rate_12(3 downto 0) + "1001";
        --light_f1 <= tep11 & tep12;
end if;

if temp_rate_13(7 downto 4) = "0011" then--fin lig
        tep13 <= temp_rate_13(3 downto 0);
        light_f1 <= tep12 & tep13;
elsif temp_rate_13(7 downto 4) = "0100" then
```

```
        tep13 <= temp_rate_13(3 downto 0) + "1001";
        light_f1 <= tep12 & tep13;
end if;

if temp_rate_14(7 downto 4) = "0011" then--fin lig
        tep14 <= temp_rate_14(3 downto 0);
elsif temp_rate_14(7 downto 4) = "0100" then
        tep14 <= temp_rate_14(3 downto 0) + "1001";
end if;

if temp_rate_15(7 downto 4) = "0011" then--fin lig
        tep15 <= temp_rate_15(3 downto 0);
elsif temp_rate_15(7 downto 4) = "0100" then
        tep15 <= temp_rate_15(3 downto 0) + "1001";
end if;

if temp_rate_16(7 downto 4) = "0011" then--lig rate1
        tep16 <= temp_rate_16(3 downto 0);
--      light_f2 <= tep15 & tep16;
elsif temp_rate_16(7 downto 4) = "0100" then
        tep16 <= temp_rate_16(3 downto 0) + "1001";
        --light_f2 <= tep15 & tep16;
end if;

if temp_rate_17(7 downto 4) = "0011" then--lig rate2
        tep17 <= temp_rate_17(3 downto 0);
        light_f2 <= tep16 & tep17;
elsif temp_rate_17(7 downto 4) = "0100" then
        tep17 <= temp_rate_17(3 downto 0) + "1001";
        light_f2 <= tep16 & tep17;
end if;

if temp_rate_18(7 downto 4) = "0011" then--lig rate3
        tep18 <= temp_rate_18(3 downto 0);
        --lgt_f2 <= tep17 & tep18;
elsif temp_rate_18(7 downto 4) = "0100" then
        tep18 <= temp_rate_18(3 downto 0) + "1001";
        --lgt_f2 <= tep17 & tep18;
end if;

if temp_rate_19(7 downto 4) = "0011" then--lig rate4
        tep19 <= temp_rate_19(3 downto 0);
        lgt_f2 <= tep18 & tep19;
elsif temp_rate_19(7 downto 4) = "0100" then
        tep19 <= temp_rate_19(3 downto 0) + "1001";
        lgt_f2 <= tep18 & tep19;
end if;

if temp_rate_20(7 downto 4) = "0011" then--lig rate5
        tep20 <= temp_rate_20(3 downto 0);
```

```
            --light_f3 <= tep19 & tep20;
        elsif temp_rate_20(7 downto 4) = "0100" then
            tep20 <= temp_rate_20(3 downto 0) + "1001";
            --light_f3 <= tep19 & tep20;
        end if;

        if temp_rate_21(7 downto 4) = "0011" then--HT lig1
            tep21 <= temp_rate_21(3 downto 0);
        elsif temp_rate_21(7 downto 4) = "0100" then
            tep21 <= temp_rate_21(3 downto 0) + "1001";
            --light_f4 <= light_f3 & tep21;
        end if;

        if temp_rate_22(7 downto 4) = "0011" then--HT lig2
            tep22 <= temp_rate_22(3 downto 0);
            light_f3 <= tep21 & tep22;
        elsif temp_rate_22(7 downto 4) = "0100" then
            tep22 <= temp_rate_22(3 downto 0) + "1001";
            light_f3 <= tep21 & tep22;
        end if;

        if temp_rate_23(7 downto 4) = "0011" then--HT lig3
            tep23 <= temp_rate_23(3 downto 0);
            light_f4 <= light_f3 & tep23;
        elsif temp_rate_23(7 downto 4) = "0100" then
            tep23 <= temp_rate_23(3 downto 0) + "1001";
            light_f4 <= light_f3 & tep23;
        end if;

        if temp_rate_24(7 downto 4) = "0011" then
            tep24 <= temp_rate_24(3 downto 0);
        elsif temp_rate_24(7 downto 4) = "0100" then
            tep24 <= temp_rate_24(3 downto 0) + "1001";
        end if;

        if temp_rate_25(7 downto 4) = "0011" then
            tep25 <= temp_rate_25(3 downto 0);
        elsif temp_rate_25(7 downto 4) = "0100" then
            tep25 <= temp_rate_25(3 downto 0) + "1001";
        end if;


--comp <= (tep0 & tep1 & tep2 & tep3)+(tep4 & tep5 & tep6 & tep7)+(tep8 & tep9 &
tep10)+(tep11 & tep12);
--      if comp ="00000000" then
--          temp_final <= temp_f1 & tep2 & tep3;
--          temp_rate <= temp_f3 & tep6 & tep7;
--          holdtime <= temp_f4;
--      end if;
```

```
--comp <= (tep0 & tep1 & tep2 & tep3)+(tep4 & tep5 & tep6 & tep7)+(tep8 & tep9 &
tep10)+(tep11 & tep12 & tep13 & tep14)+(tep15 & tep16 & tep17 & tep18)+(tep19 & tep20
& tep21)+(tep22 + tep23);
        --if comp ="00000000" then
                temp_final <= temp_f1 & tep2 & tep3;
                tf <= temp_f1 & tep2 & tep3;
                temp_rate <= temp_f3 & rate2 & tep8;
        --      tr <= temp_f3 & tep6 & tep7;
                holdtime <= temp_f4;
        --end if;
                light_final <= light_f1 & tep14 & tep15;
                light_rate <= light_f2 & lgt_f2 & tep20;
                --light_rate <= light_f2 & tep17 & tep18;
                lgt_holdtime <= light_f4;
        --end if;
--      tep21 & tep22 is for checksum
end if;--for clk




end process;

tr_i <= CONV_INTEGER(tf);
ft <= tr_i/4;

end macro_level_definition;

------------------------------------------------------------------------------
--
-- END OF FILE UART_RX.VHD
--
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----
---- END OF FILE UART_RX.VHD
----
------------------------------------------------------------------------------
--
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;
--
------------------------------------------------------------------------------
--
-- Main Entity for KCUART_RX
```

```
--
entity kcuart_rx is
   Port (      serial_in : in std_logic;
               data_out : out std_logic_vector(7 downto 0);
            data_strobe : out std_logic;
           en_16_x_baud : in std_logic;
                  clk : in std_logic);
   end kcuart_rx;
--
----------------------------------------------------------------------------------------
--
-- Start of Main Architecture for KCUART_RX
--
architecture low_level_definition of kcuart_rx is
--
----------------------------------------------------------------------------------------
--
-- Signals used in KCUART_RX
--
----------------------------------------------------------------------------------------
--
signal sync_serial        : std_logic;
signal stop_bit           : std_logic;
signal data_int           : std_logic_vector(7 downto 0);
signal data_delay         : std_logic_vector(7 downto 0);
signal start_delay        : std_logic;
signal start_bit          : std_logic;
signal edge_delay         : std_logic;
signal start_edge         : std_logic;
signal decode_valid_char  : std_logic;
signal valid_char         : std_logic;
signal decode_purge       : std_logic;
signal purge              : std_logic;
signal valid_srl_delay    : std_logic_vector(8 downto 0);
signal valid_reg_delay    : std_logic_vector(8 downto 0);
signal decode_data_strobe : std_logic;
--
--
----------------------------------------------------------------------------------------
--
-- Start of KCUART_RX circuit description
--
----------------------------------------------------------------------------------------
--
begin

  -- Synchronise input serial data to system clock

  sync_reg: FD
  port map ( D => serial_in,
```

```
        Q => sync_serial,
        C => clk);

stop_reg: FD
port map ( D => sync_serial,
        Q => stop_bit,
        C => clk);


-- Data delays to capture data at 16 time baud rate
-- Each SRL16E is followed by a flip-flop for best timing

data_loop: for i in 0 to 7 generate
begin

  lsbs: if i<7 generate
  begin

    delay15_srl: SRL16E
    generic map (INIT => X"0000")
    port map(   D => data_int(i+1),
              CE => en_16_x_baud,
             CLK => clk,
              A0 => '0',
              A1 => '1',
              A2 => '1',
              A3 => '1',
               Q => data_delay(i) );

  end generate lsbs;

  msb: if i=7 generate
  begin

    delay15_srl: SRL16E
    generic map (INIT => X"0000")
    port map(   D => stop_bit,
              CE => en_16_x_baud,
             CLK => clk,
              A0 => '0',
              A1 => '1',
              A2 => '1',
              A3 => '1',
               Q => data_delay(i) );

  end generate msb;

  data_reg: FDE
  port map ( D => data_delay(i),
           Q => data_int(i),
```

```
            CE => en_16_x_baud,
             C => clk);

end generate data_loop;

-- Assign internal signals to outputs

data_out <= data_int;

-- Data delays to capture start bit at 16 time baud rate

start_srl: SRL16E
generic map (INIT => X"0000")
port map(   D => data_int(0),
          CE => en_16_x_baud,
         CLK => clk,
          A0 => '0',
          A1 => '1',
          A2 => '1',
          A3 => '1',
           Q => start_delay );

start_reg: FDE
port map ( D => start_delay,
          Q => start_bit,
         CE => en_16_x_baud,
          C => clk);


-- Data delays to capture start bit leading edge at 16 time baud rate
-- Delay ensures data is captured at mid-bit position

edge_srl: SRL16E
generic map (INIT => X"0000")
port map(   D => start_bit,
          CE => en_16_x_baud,
         CLK => clk,
          A0 => '1',
          A1 => '0',
          A2 => '1',
          A3 => '0',
           Q => edge_delay );

edge_reg: FDE
port map ( D => edge_delay,
          Q => start_edge,
         CE => en_16_x_baud,
          C => clk);

-- Detect a valid character
```

```
valid_lut: LUT4
generic map (INIT => X"0040")
port map( I0 => purge,
          I1 => stop_bit,
          I2 => start_edge,
          I3 => edge_delay,
           O => decode_valid_char );

valid_reg: FDE
port map ( D => decode_valid_char,
           Q => valid_char,
          CE => en_16_x_baud,
           C => clk);

-- Purge of data status

purge_lut: LUT3
generic map (INIT => X"54")
port map( I0 => valid_reg_delay(8),
          I1 => valid_char,
          I2 => purge,
           O => decode_purge );

purge_reg: FDE
port map ( D => decode_purge,
           Q => purge,
          CE => en_16_x_baud,
           C => clk);

-- Delay of valid_char pulse of length equivalent to the time taken
-- to purge data shift register of all data which has been used.
-- Requires 9x16 + 8 delays which is achieved by packing of SRL16E with
-- up to 16 delays and utilising the dedicated flip flop in each stage.

valid_loop: for i in 0 to 8 generate
begin

  lsb: if i=0 generate
  begin

    delay15_srl: SRL16E
    generic map (INIT => X"0000")
    port map(   D => valid_char,
              CE => en_16_x_baud,
             CLK => clk,
              A0 => '0',
              A1 => '1',
              A2 => '1',
              A3 => '1',
```

```
                Q => valid_srl_delay(i) );

    end generate lsb;

    msbs: if i>0 generate
    begin

      delay16_srl: SRL16E
      generic map (INIT => X"0000")
      port map(   D => valid_reg_delay(i-1),
                 CE => en_16_x_baud,
                CLK => clk,
                 A0 => '1',
                 A1 => '1',
                 A2 => '1',
                 A3 => '1',
                  Q => valid_srl_delay(i) );

    end generate msbs;

    data_reg: FDE
    port map ( D => valid_srl_delay(i),
               Q => valid_reg_delay(i),
              CE => en_16_x_baud,
               C => clk);

  end generate valid_loop;

  -- Form data strobe

  strobe_lut: LUT2
  generic map (INIT => X"8")
  port map( I0 => valid_char,
            I1 => en_16_x_baud,
             O => decode_data_strobe );

  strobe_reg: FD
  port map ( D => decode_data_strobe,
             Q => data_strobe,
             C => clk);

end low_level_definition;

------------------------------------------------------------------------------------
--
-- END OF FILE kcuart.vhd
--
------------------------------------------------------------------------------------
```

```
-- 16 deep
-- 8-bit data
-- Library declarations
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;
entity rxfifo_16x8 is
    Port (       data_in : in std_logic_vector(7 downto 0);
                data_out : out std_logic_vector(7 downto 0);
                   reset : in std_logic;
                   writ : in std_logic;
                    reed : in std_logic;
                    full : out std_logic;
               half_full : out std_logic;
            data_present : out std_logic;
                     clk : in std_logic);
    end rxfifo_16x8;
--
----------------------------------------------------------------------------------------------
--
-- Start of Main Architecture for BBFIFO_16x8
--
architecture low_level_definition of rxfifo_16x8 is
--
----------------------------------------------------------------------------------------------
--
-- Signals used in BBFIFO_16x8
--
----------------------------------------------------------------------------------------------
--
signal pointer             : std_logic_vector(3 downto 0);
signal next_count          : std_logic_vector(3 downto 0);
signal half_count          : std_logic_vector(3 downto 0);
signal count_carry         : std_logic_vector(2 downto 0);

signal pointer_zero        : std_logic;
signal pointer_full        : std_logic;
signal decode_data_present : std_logic;
signal data_present_int    : std_logic;
signal valid_write         : std_logic;
--
----------------------------------------------------------------------------------------------
--
-- Start of BBFIFO_16x8 circuit description
--
----------------------------------------------------------------------------------------------
--
```

```
begin

  -- SRL16E data storage

  data_width_loop: for i in 0 to 7 generate
  begin

    data_srl: SRL16E
    generic map (INIT => X"0000")
    port map(   D => data_in(i),
              CE => valid_write,
             CLK => clk,
              A0 => pointer(0),
              A1 => pointer(1),
              A2 => pointer(2),
              A3 => pointer(3),
               Q => data_out(i) );

  end generate data_width_loop;

  -- 4-bit counter to act as data pointer
  -- Counter is clock enabled by 'data_present'
  -- Counter will be reset when 'reset' is active
  -- Counter will increment when 'valid_write' is active

  count_width_loop: for i in 0 to 3 generate
  begin

    register_bit: FDRE
    port map ( D => next_count(i),
               Q => pointer(i),
              CE => data_present_int,
               R => reset,
               C => clk);

    count_lut: LUT4
    generic map (INIT => X"6606")
    port map( I0 => pointer(i),
              I1 => reed,
              I2 => pointer_zero,
              I3 => writ,
               O => half_count(i));

    lsb_count: if i=0 generate
    begin

      count_muxcy: MUXCY
      port map( DI => pointer(i),
                CI => valid_write,
                 S => half_count(i),
```

```
            O => count_carry(i));

    count_xor: XORCY
    port map( LI => half_count(i),
              CI => valid_write,
              O => next_count(i));

  end generate lsb_count;

  mid_count: if i>0 and i<3 generate
  begin

    count_muxcy: MUXCY
    port map( DI => pointer(i),
              CI => count_carry(i-1),
              S => half_count(i),
              O => count_carry(i));

    count_xor: XORCY
    port map( LI => half_count(i),
              CI => count_carry(i-1),
              O => next_count(i));

  end generate mid_count;

  upper_count: if i=3 generate
  begin

    count_xor: XORCY
    port map( LI => half_count(i),
              CI => count_carry(i-1),
              O => next_count(i));

  end generate upper_count;

end generate count_width_loop;


-- Detect when pointer is zero and maximum

zero_lut: LUT4
generic map (INIT => X"0001")
port map( I0 => pointer(0),
          I1 => pointer(1),
          I2 => pointer(2),
          I3 => pointer(3),
           O => pointer_zero );


full_lut: LUT4
```

```
  generic map (INIT => X"8000")
  port map( I0 => pointer(0),
            I1 => pointer(1),
            I2 => pointer(2),
            I3 => pointer(3),
             O => pointer_full );


  -- Data Present status

  dp_lut: LUT4
  generic map (INIT => X"BFA0")
  port map( I0 => writ,
            I1 => reed,
            I2 => pointer_zero,
            I3 => data_present_int,
             O => decode_data_present );

  dp_flop: FDR
  port map ( D => decode_data_present,
             Q => data_present_int,
             R => reset,
             C => clk);

  -- Valid write signal

  valid_lut: LUT3
  generic map (INIT => X"C4")
  port map( I0 => pointer_full,
            I1 => writ,
            I2 => reed,
             O => valid_write );


  -- assign internal signals to outputs

  full <= pointer_full;
  half_full <= pointer(3);
  data_present <= data_present_int;

end low_level_definition;

------------------------------------------------------------------------------------
--
-- END OF FILE bbfifo_16x8.vhd
--
------------------------------------------------------------------------------------
```

```
-- UART Transmitter with integral 16 byte FIFO buffer
--
```

```
-- 8 bit, no parity, 1 stop bit
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;
--
----------------------------------------------------------------------------------
--
-- Main Entity for UART_TX
--
entity uart_tx is
    Port (           data_in : in std_logic_vector(7 downto 0);
                write_buffer : in std_logic;
                reset_buffer : in std_logic;
                en_16_x_baud : in std_logic;
                  serial_out : out std_logic;
                 buffer_full : out std_logic;
           buffer_half_full : out std_logic;
                         clk : in std_logic);
    end uart_tx;
--
----------------------------------------------------------------------------------
--
-- Start of Main Architecture for UART_TX
--
architecture macro_level_definition of uart_tx is
--
----------------------------------------------------------------------------------
--
-- Components used in UART_TX and defined in subsequent entities.
--
----------------------------------------------------------------------------------
--
-- Constant (K) Compact UART Transmitter
--
component kcuart_tx
    Port (        data_in : in std_logic_vector(7 downto 0);
          send_character : in std_logic;
            en_16_x_baud : in std_logic;
              serial_out : out std_logic;
             Tx_complete : out std_logic;
                     clk : in std_logic);
    end component;
component txfifo_16x8
    Port (       data_in : in std_logic_vector(7 downto 0);
                data_out : out std_logic_vector(7 downto 0);
                   reset : in std_logic;
                   write : in std_logic;
```

```
                read : in std_logic;
                full : out std_logic;
           half_full : out std_logic;
        data_present : out std_logic;
                clk : in std_logic);
   end component;
--
-------------------------------------------------------------------------------
--
-- Signals used in UART_TX
--
-------------------------------------------------------------------------------
--
signal fifo_data_out      : std_logic_vector(7 downto 0);
signal fifo_data_present  : std_logic;
signal fifo_read          : std_logic;
--
-------------------------------------------------------------------------------
--
-- Start of UART_TX circuit description
--
-------------------------------------------------------------------------------
--
begin

  -- 8 to 1 multiplexer to convert parallel data to serial

  kcuart: kcuart_tx
  port map (       data_in => fifo_data_out,
           send_character => fifo_data_present,
             en_16_x_baud => en_16_x_baud,
               serial_out => serial_out,
              Tx_complete => fifo_read,
                      clk => clk);


  buf: txfifo_16x8
  port map (      data_in => data_in,
                 data_out => fifo_data_out,
                    reset => reset_buffer,
                    write => write_buffer,
                     read => fifo_read,
                     full => buffer_full,
                half_full => buffer_half_full,
             data_present => fifo_data_present,
                      clk => clk);

end macro_level_definition;

-------------------------------------------------------------------------------
```

```
--
-- END OF FILE UART_TX.VHD
--
--------------------------------------------------------------------------------
```

```
--UART Transmitter
--
-- 8 data bits, no parity, 1 stop bit
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;
--
----------------------------------------------------------------------------------------------
--
-- Main Entity for KCUART_TX
--
entity kcuart_tx is
    Port (       data_in : in std_logic_vector(7 downto 0);
          send_character : in std_logic;
            en_16_x_baud : in std_logic;
              serial_out : out std_logic;
             Tx_complete : out std_logic;
                     clk : in std_logic);
    end kcuart_tx;
--
----------------------------------------------------------------------------------------------
--
-- Start of Main Architecture for KCUART_TX
--
architecture low_level_definition of kcuart_tx is
--
----------------------------------------------------------------------------------------------
--
-- Signals used in KCUART_TX
--
----------------------------------------------------------------------------------------------
--
signal data_01           : std_logic;
signal data_23           : std_logic;
signal data_45           : std_logic;
signal data_67           : std_logic;
signal data_0123          : std_logic;
signal data_4567          : std_logic;
signal data_01234567      : std_logic;
signal bit_select         : std_logic_vector(2 downto 0);
signal next_count         : std_logic_vector(2 downto 0);
signal mask_count         : std_logic_vector(2 downto 0);
```

```
signal mask_count_carry   : std_logic_vector(2 downto 0);
signal count_carry        : std_logic_vector(2 downto 0);
signal ready_to_start     : std_logic;
signal decode_Tx_start    : std_logic;
signal Tx_start           : std_logic;
signal decode_Tx_run      : std_logic;
signal Tx_run             : std_logic;
signal decode_hot_state   : std_logic;
signal hot_state          : std_logic;
signal hot_delay          : std_logic;
signal Tx_bit             : std_logic;
signal decode_Tx_stop     : std_logic;
signal Tx_stop            : std_logic;
signal decode_Tx_complete : std_logic;
--
-------------------------------------------------------------------------------------------
--
-- Start of KCUART_TX circuit description
--
-------------------------------------------------------------------------------------------
--
begin

  -- 8 to 1 multiplexer to convert parallel data to serial

  mux1_lut: LUT4
  generic map (INIT => X"E4FF")
  port map( I0 => bit_select(0),
            I1 => data_in(0),
            I2 => data_in(1),
            I3 => Tx_run,
             O => data_01 );

  mux2_lut: LUT4
  generic map (INIT => X"E4FF")
  port map( I0 => bit_select(0),
            I1 => data_in(2),
            I2 => data_in(3),
            I3 => Tx_run,
             O => data_23 );

  mux3_lut: LUT4
  generic map (INIT => X"E4FF")
  port map( I0 => bit_select(0),
            I1 => data_in(4),
            I2 => data_in(5),
            I3 => Tx_run,
             O => data_45 );

  mux4_lut: LUT4
```

```
generic map (INIT => X"E4FF")
port map( I0 => bit_select(0),
          I1 => data_in(6),
          I2 => data_in(7),
          I3 => Tx_run,
           O => data_67 );

mux5_muxf5: MUXF5
port map(  I1 => data_23,
           I0 => data_01,
            S => bit_select(1),
            O => data_0123 );

mux6_muxf5: MUXF5
port map(  I1 => data_67,
           I0 => data_45,
            S => bit_select(1),
            O => data_4567 );

mux7_muxf6: MUXF6
port map(  I1 => data_4567,
           I0 => data_0123,
            S => bit_select(2),
            O => data_01234567 );

-- Register serial output and force start and stop bits

pipeline_serial: FDRS
port map ( D => data_01234567,
           Q => serial_out,
           R => Tx_start,
           S => Tx_stop,
           C => clk);

-- 3-bit counter
-- Counter is clock enabled by en_16_x_baud
-- Counter will be reset when 'Tx_start' is active
-- Counter will increment when Tx_bit is active
-- Tx_run must be active to count
-- count_carry(2) indicates when terminal count (7) is reached and Tx_bit=1 (i.e. overflow)

count_width_loop: for i in 0 to 2 generate
begin

  register_bit: FDRE
  port map ( D => next_count(i),
             Q => bit_select(i),
            CE => en_16_x_baud,
             R => Tx_start,
             C => clk);
```

```
  count_lut: LUT2
  generic map (INIT => X"8")
  port map( I0 => bit_select(i),
            I1 => Tx_run,
            O => mask_count(i));

  mask_and: MULT_AND
  port map( I0 => bit_select(i),
            I1 => Tx_run,
            LO => mask_count_carry(i));

  lsb_count: if i=0 generate
  begin

    count_muxcy: MUXCY
    port map( DI => mask_count_carry(i),
              CI => Tx_bit,
              S => mask_count(i),
              O => count_carry(i));

    count_xor: XORCY
    port map( LI => mask_count(i),
              CI => Tx_bit,
              O => next_count(i));

  end generate lsb_count;

  upper_count: if i>0 generate
  begin

    count_muxcy: MUXCY
    port map( DI => mask_count_carry(i),
              CI => count_carry(i-1),
              S => mask_count(i),
              O => count_carry(i));

    count_xor: XORCY
    port map( LI => mask_count(i),
              CI => count_carry(i-1),
              O => next_count(i));

  end generate upper_count;

end generate count_width_loop;

-- Ready to start decode

ready_lut: LUT3
generic map (INIT => X"10")
```

```
port map( I0 => Tx_run,
          I1 => Tx_start,
          I2 => send_character,
           O => ready_to_start );

-- Start bit enable

start_lut: LUT4
generic map (INIT => X"0190")
port map( I0 => Tx_bit,
          I1 => Tx_stop,
          I2 => ready_to_start,
          I3 => Tx_start,
           O => decode_Tx_start );

Tx_start_reg: FDE
port map ( D => decode_Tx_start,
           Q => Tx_start,
          CE => en_16_x_baud,
           C => clk);


-- Run bit enable

run_lut: LUT4
generic map (INIT => X"1540")
port map( I0 => count_carry(2),
          I1 => Tx_bit,
          I2 => Tx_start,
          I3 => Tx_run,
           O => decode_Tx_run );

Tx_run_reg: FDE
port map ( D => decode_Tx_run,
           Q => Tx_run,
          CE => en_16_x_baud,
           C => clk);

-- Bit rate enable

hot_state_lut: LUT3
generic map (INIT => X"94")
port map( I0 => Tx_stop,
          I1 => ready_to_start,
          I2 => Tx_bit,
           O => decode_hot_state );

hot_state_reg: FDE
port map ( D => decode_hot_state,
           Q => hot_state,
```

```
        CE => en_16_x_baud,
         C => clk);

delay14_srl: SRL16E
generic map (INIT => X"0000")
port map(   D => hot_state,
         CE => en_16_x_baud,
        CLK => clk,
         A0 => '1',
         A1 => '0',
         A2 => '1',
         A3 => '1',
          Q => hot_delay );

Tx_bit_reg: FDE
port map ( D => hot_delay,
         Q => Tx_bit,
        CE => en_16_x_baud,
         C => clk);

-- Stop bit enable

stop_lut: LUT4
generic map (INIT => X"0180")
port map( I0 => Tx_bit,
        I1 => Tx_run,
        I2 => count_carry(2),
        I3 => Tx_stop,
         O => decode_Tx_stop );

Tx_stop_reg: FDE
port map ( D => decode_Tx_stop,
         Q => Tx_stop,
        CE => en_16_x_baud,
         C => clk);

-- Tx_complete strobe

complete_lut: LUT2
generic map (INIT => X"8")
port map( I0 => count_carry(2),
        I1 => en_16_x_baud,
         O => decode_Tx_complete );

Tx_complete_reg: FD
port map ( D => decode_Tx_complete,
         Q => Tx_complete,
         C => clk);
```

```
end low_level_definition;

------------------------------------------------------------------------------------------
--
-- END OF FILE kcuart_tx.vhd
------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;
--
------------------------------------------------------------------------------------------
--
-- Main Entity for BBFIFO_16x8
--
entity txfifo_16x8 is
    Port (       data_in : in std_logic_vector(7 downto 0);
                data_out : out std_logic_vector(7 downto 0);
                   reset : in std_logic;
                   write : in std_logic;
                    read : in std_logic;
                    full : out std_logic;
               half_full : out std_logic;
            data_present : out std_logic;
                     clk : in std_logic);
    end txfifo_16x8;
--
------------------------------------------------------------------------------------------
--
-- Start of Main Architecture for BBFIFO_16x8
--
architecture low_level_definition of txfifo_16x8 is
--
------------------------------------------------------------------------------------------
--
-- Signals used in BBFIFO_16x8
--
------------------------------------------------------------------------------------------
--
signal pointer             : std_logic_vector(3 downto 0);
signal next_count          : std_logic_vector(3 downto 0);
signal half_count          : std_logic_vector(3 downto 0);
signal count_carry         : std_logic_vector(2 downto 0);

signal pointer_zero        : std_logic;
signal pointer_full        : std_logic;
signal decode_data_present : std_logic;
signal data_present_int    : std_logic;
signal valid_write         : std_logic;
```

```
--
------------------------------------------------------------------------------
--
-- Start of BBFIFO_16x8 circuit description
--
------------------------------------------------------------------------------
--
begin

  -- SRL16E data storage

  data_width_loop: for i in 0 to 7 generate
  begin

    data_srl: SRL16E
    generic map (INIT => X"0000")
    port map(   D => data_in(i),
              CE => valid_write,
             CLK => clk,
              A0 => pointer(0),
              A1 => pointer(1),
              A2 => pointer(2),
              A3 => pointer(3),
               Q => data_out(i) );

  end generate data_width_loop;

  -- 4-bit counter to act as data pointer
  -- Counter is clock enabled by 'data_present'
  -- Counter will be reset when 'reset' is active
  -- Counter will increment when 'valid_write' is active

  count_width_loop: for i in 0 to 3 generate
  begin

    register_bit: FDRE
    port map ( D => next_count(i),
               Q => pointer(i),
              CE => data_present_int,
               R => reset,
               C => clk);

    count_lut: LUT4
    generic map (INIT => X"6606")
    port map( I0 => pointer(i),
              I1 => read,
              I2 => pointer_zero,
              I3 => write,
               O => half_count(i));
```

```
  lsb_count: if i=0 generate
  begin

    count_muxcy: MUXCY
    port map( DI => pointer(i),
              CI => valid_write,
               S => half_count(i),
               O => count_carry(i));

    count_xor: XORCY
    port map( LI => half_count(i),
              CI => valid_write,
               O => next_count(i));

  end generate lsb_count;

  mid_count: if i>0 and i<3 generate
  begin

    count_muxcy: MUXCY
    port map( DI => pointer(i),
              CI => count_carry(i-1),
               S => half_count(i),
               O => count_carry(i));

    count_xor: XORCY
    port map( LI => half_count(i),
              CI => count_carry(i-1),
               O => next_count(i));

  end generate mid_count;

  upper_count: if i=3 generate
  begin

    count_xor: XORCY
    port map( LI => half_count(i),
              CI => count_carry(i-1),
               O => next_count(i));

  end generate upper_count;

end generate count_width_loop;


-- Detect when pointer is zero and maximum

zero_lut: LUT4
generic map (INIT => X"0001")
port map( I0 => pointer(0),
```

```
          I1 => pointer(1),
          I2 => pointer(2),
          I3 => pointer(3),
           O => pointer_zero );


  full_lut: LUT4
  generic map (INIT => X"8000")
  port map( I0 => pointer(0),
          I1 => pointer(1),
          I2 => pointer(2),
          I3 => pointer(3),
           O => pointer_full );


  -- Data Present status

  dp_lut: LUT4
  generic map (INIT => X"BFA0")
  port map( I0 => write,
          I1 => read,
          I2 => pointer_zero,
          I3 => data_present_int,
           O => decode_data_present );

  dp_flop: FDR
  port map ( D => decode_data_present,
           Q => data_present_int,
           R => reset,
           C => clk);

  -- Valid write signal

  valid_lut: LUT3
  generic map (INIT => X"C4")
  port map( I0 => pointer_full,
          I1 => write,
          I2 => read,
           O => valid_write );


  -- assign internal signals to outputs

  full <= pointer_full;
  half_full <= pointer(3);
  data_present <= data_present_int;

end low_level_definition;

------------------------------------------------------------------------------------
```

```
--
-- END OF FILE bbfifo_16x8.vhd
--
--------------------------------------------------------------------------------
--FIFO
---- 16 deep
---- 8-bit data

--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.std_logic_signed.all;
use IEEE.STD_LOGIC_1164.ALL;

entity in2slv is
    Port (          clk: in std_logic;
                            --          data_cnt: in integer range 0 to 39;
                                        final_temp: out integer range 0 to 16383;
                                        hold_time: out integer range 0 to 3600;
                                        rate : out integer range 0 to 1048575;--66000;
                                        result : in std_logic_vector(15 downto 0);---adc o/p
                                        feedback : out integer range 0 to 65535;--16383;--feedback
                                        temp_final: in std_logic_vector(15 downto 0);
                                        temp_rate: in std_logic_vector(19 downto 0);
                                        holdtime: in std_logic_vector(11 downto 0);
                                        light_final : IN std_logic_vector(15 downto 0);
                                        light_rate : IN std_logic_vector(19 downto 0);
                                        lgt_holdtime : IN std_logic_vector(11 downto 0);
                            light_final_i : OUT integer range 0 to 16383;--std_logic_vector(0 to
13);
                                        light_rate_i : OUT integer range 0 to 1048575;--66000;
                            lgt_holdtime_i : OUT integer range 0 to 3600--std_logic_vector(0 to
15)

                            );
end in2slv;

architecture Behavioral of in2slv is

signal fb0,fb1, fb2,fb3,fb4,fb5,fb6,fb7,fb8,fts: integer range 0 to 16383;
signal ht: integer range 0 to 3600;
--signal rt: integer range 0 to 1048575;--66000;
signal tfs: std_logic_vector(15 downto 0);
signal htv: std_logic_vector(11 downto 0);
signal flag, flag2: std_logic;
signal cnt: integer range 0 to 3;

begin

final_temp <= fts;
tfs <= temp_final;
```

```
--tr <= temp_rate;
--rate <= rt;
htv <= holdtime;
hold_time <= ht;

        fts <= CONV_INTEGER(tfs);--
        rate <= CONV_INTEGER(temp_rate);
        ht <= CONV_INTEGER(htv);
        feedback <= CONV_INTEGER(result);--fb0

        light_final_i <= CONV_INTEGER(light_final);
        light_rate_i <= CONV_INTEGER(light_rate);
        lgt_holdtime_i <= CONV_INTEGER(lgt_holdtime);

end;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity hex_2_asciihex is
   Port ( clk: in std_logic;
                        lsb0_ri :in std_logic_vector(3 downto 0);-- r=ramp, f= fedback,i=
input, o=output
                        lsb1_ri :in std_logic_vector(3 downto 0);
                        msb0_ri :in std_logic_vector(3 downto 0);
                        msb1_ri :in std_logic_vector(3 downto 0);
                        lsb0_fi :in std_logic_vector(3 downto 0);
                        lsb1_fi :in std_logic_vector(3 downto 0);
                        msb0_fi :in std_logic_vector(3 downto 0);
                        msb1_fi :in std_logic_vector(3 downto 0);
                        lsb0_ro :out std_logic_vector(7 downto 0);
                        lsb1_ro :out std_logic_vector(7 downto 0);
                        msb0_ro :out std_logic_vector(7 downto 0);
                        msb1_ro :out std_logic_vector(7 downto 0);
                        lsb0_fo :out std_logic_vector(7 downto 0);
                        lsb1_fo :out std_logic_vector(7 downto 0);
                        msb0_fo :out std_logic_vector(7 downto 0);
                        msb1_fo :out std_logic_vector(7 downto 0)
                       );
end hex_2_asciihex;

architecture Behavioral of hex_2_asciihex is

begin

process(clk)
begin
```

```
if(clk'event and clk = '1')then

if lsb0_ri < "1010" then
        lsb0_ro <= "0011" & lsb0_ri; --when (lsb0_ri < "1010") else "0100" & (lsb0_ri +
"0111");
else
        lsb0_ro <= "0100" & (lsb0_ri + "0111");
end if;

if lsb1_ri < "1010" then
        lsb1_ro <= "0011" & lsb1_ri; --when (lsb0_ri < "1010") else "0100" & (lsb0_ri +
"0111");
else
        lsb1_ro <= "0100" & (lsb1_ri + "0111");
end if;

if msb0_ri < "1010" then
        msb0_ro <= "0011" & msb0_ri; --when (lsb0_ri < "1010") else "0100" & (lsb0_ri +
"0111");
else
        msb0_ro <= "0100" & (msb0_ri + "0111");
end if;

if msb1_ri <"1010" then
        msb1_ro <="0011" & msb1_ri;
else
        msb1_ro <= "0100" & (msb1_ri + "0111");
        end if;

if lsb0_fi < "1010" then
lsb0_fo <= "0011" & lsb0_fi ;
else
lsb0_fo <="0100" & (lsb0_fi + "0111");
end if;

if (lsb1_fi < "1010") then
lsb1_fo <= "0011" & lsb1_fi;
else
lsb1_fo <="0100" & (lsb1_fi + "0111");
end if;

if (msb0_fi < "1010") then
msb0_fo <= "0011" & msb0_fi;
else
msb0_fo <= "0100" & (msb0_fi + "0111");
end if;

if (msb1_fi < "1010") then
msb1_fo <= "0011" & msb1_fi;
else
```

```
msb1_fo <= "0100" & (msb1_fi + "0111");
end if;

end if;
end process;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity hex_to_asciihex is
   Port ( clk: in std_logic;
                  lsb0_ri :in std_logic_vector(3 downto 0);-- r=ramp, f= fedback,i= input,
o=output
                           lsb1_ri :in std_logic_vector(3 downto 0);
                           msb0_ri :in std_logic_vector(3 downto 0);
                           msb1_ri :in std_logic_vector(3 downto 0);
                           lsb0_fi :in std_logic_vector(3 downto 0);
                           lsb1_fi :in std_logic_vector(3 downto 0);
                           msb0_fi :in std_logic_vector(3 downto 0);
                           msb1_fi :in std_logic_vector(3 downto 0);
                           lsb0_ro :out std_logic_vector(7 downto 0);
                           lsb1_ro :out std_logic_vector(7 downto 0);
                           msb0_ro :out std_logic_vector(7 downto 0);
                           msb1_ro :out std_logic_vector(7 downto 0);
                           lsb0_fo :out std_logic_vector(7 downto 0);
                           lsb1_fo :out std_logic_vector(7 downto 0);
                           msb0_fo :out std_logic_vector(7 downto 0);
                           msb1_fo :out std_logic_vector(7 downto 0)
                         );
end hex_to_asciihex;

architecture Behavioral of hex_to_asciihex is

begin

process(clk)
begin
if(clk'event and clk = '1')then

if lsb0_ri < "1010" then
        lsb0_ro <= "0011" & lsb0_ri; --when (lsb0_ri < "1010") else "0100" & (lsb0_ri +
"0111");
else
        lsb0_ro <= "0100" & (lsb0_ri + "0111");
end if;
```

```
if lsb1_ri < "1010" then
        lsb1_ro <= "0011" & lsb1_ri; --when (lsb0_ri < "1010") else "0100" & (lsb0_ri +
"0111");
else
        lsb1_ro <= "0100" & (lsb1_ri + "0111");
end if;

if msb0_ri < "1010" then
        msb0_ro <= "0011" & msb0_ri; --when (lsb0_ri < "1010") else "0100" & (lsb0_ri +
"0111");
else
        msb0_ro <= "0100" & (msb0_ri + "0111");
end if;

if msb1_ri <"1010" then
        msb1_ro <="0011" & msb1_ri;
else
        msb1_ro <= "0100" & (msb1_ri + "0111");
        end if;

if lsb0_fi < "1010" then
lsb0_fo <= "0011" & lsb0_fi ;
else
lsb0_fo <="0100" & (lsb0_fi + "0111");
end if;

if (lsb1_fi < "1010") then
lsb1_fo <= "0011" & lsb1_fi;
else
lsb1_fo <="0100" & (lsb1_fi + "0111");
end if;

if (msb0_fi < "1010") then
msb0_fo <= "0011" & msb0_fi;
else
msb0_fo <= "0100" & (msb0_fi + "0111");
end if;

if (msb1_fi < "1010") then
msb1_fo <= "0011" & msb1_fi;
else
msb1_fo <= "0100" & (msb1_fi + "0111");
end if;

end if;
end process;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ADC is
  Port    (
  --General usage
    CLK      : in std_logic; --50MHz
    debug      : out std_logic;

  --Pmod interface signals
    SDATA1   : in std_logic;--data coming from ADC
    SDATA2   : in std_logic;--data coming from ADC
    SCLK     : out std_logic;--12.5 MHz
    nCS      : out std_logic;--chip select

    --User interface signals
    temp1    : out std_logic_vector(15 downto 0);
    light2    : out std_logic_vector(15 downto 0);
           light_opti : out std_logic_vector(15 downto 0)
          );

end ADC ;

architecture AD1 of ADC is

COMPONENT AD1RefComp
          PORT(
                    CLK : IN std_logic;
                    RST : IN std_logic;
                    SDATA1 : IN std_logic;
                    SDATA2 : IN std_logic;
                    START : IN std_logic;
                    SCLK : OUT std_logic;
                    nCS : OUT std_logic;
                    DATA1 : OUT std_logic_vector(11 downto 0);
                    DATA2 : OUT std_logic_vector(11 downto 0);
                    DONE : OUT std_logic
                    );
          END COMPONENT;


signal dat1, dat2 :std_logic_vector(11 downto 0);
signal over, conv : std_logic;

begin

AD1: AD1RefComp PORT MAP(
                    CLK => CLK,
                    RST => '0',
                    SDATA1 => SDATA1,
```

```
                SDATA2 => SDATA2,
                SCLK => SCLK,
                nCS => nCS,
                DATA1 => dat1,
                DATA2 => dat2,
                START => conv,
                DONE => over);


process (CLK)
begin
if CLK'event and CLK = '1' then
        if over = '1' then
                temp1 <= "0000" & dat1;-- & "0";
                light2 <= "00" & dat2 & "00";
                light_opti <= "0000" & dat2;
                conv <= '1';
        else
                conv <= '0';
        end if;
end if;
end process;

debug <= over;

end AD1;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity AD1RefComp is
  Port    (
  --General usage
    CLK      : in std_logic; --50MHz
    RST      : in std_logic;--asynchronous reset

  --Pmod interface signals
    SDATA1   : in std_logic;--data coming from ADC
    SDATA2   : in std_logic;--data coming from ADC
    SCLK     : out std_logic;--12.5 MHz
    nCS      : out std_logic;--chip select

    --User interface signals
    DATA1    : out std_logic_vector(11 downto 0);
    DATA2    : out std_logic_vector(11 downto 0);
    START    : in std_logic; --start of conversion
    DONE     : out std_logic--latch data when active; ck this pin in cro for data speed
          );
```

```
end AD1RefComp ;

architecture AD1 of AD1RefComp is

type states is (Idle,
                ShiftIn,
                SyncData);
          signal current_state : states;
          signal next_state    : states;

          signal temp1         : std_logic_vector(15 downto 0);
          signal temp2         : std_logic_vector(15 downto 0);
          signal clk_div       : std_logic;
          signal clk_counter   : std_logic_vector(1 downto 0);
          signal shiftCounter  : std_logic_vector(3 downto 0) := x"0";
          signal enShiftCounter: std_logic;
          signal enParalelLoad : std_logic;


begin

        clock_divide : process(rst,clk)
        begin
           if rst = '1' then
              clk_counter <= "00";
           elsif (clk = '1' and clk'event) then
              clk_counter <= clk_counter + '1';
           end if;
        end process;

        clk_div <= clk_counter(1);
        SCLK <=  not clk_counter(1);

counter : process(clk_div, enParalelLoad, enShiftCounter)
        begin
           if (clk_div = '1' and clk_div'event) then

              if (enShiftCounter = '1') then
                 temp1 <= temp1(14 downto 0) & SDATA1;
                  temp2 <= temp2(14 downto 0) & SDATA2;
                  shiftCounter <= shiftCounter + '1';
              elsif (enParalelLoad = '1') then
                 shiftCounter <= "0000";
                 DATA1 <= temp1(11 downto 0);
                  DATA2 <= temp2(11 downto 0);
              end if;
           end if;
        end process;
```

```
SYNC_PROC: process (clk_div, rst)
  begin
    if (clk_div'event and clk_div = '1') then
      if (rst = '1') then
        current_state <= Idle;
      else
        current_state <= next_state;
      end if;
    end if;
  end process;

OUTPUT_DECODE: process (current_state)
  begin
    if current_state = Idle then
        enShiftCounter <='0';
        DONE <='1';
        nCS <='1';
        enParalelLoad <= '0';
     elsif current_state = ShiftIn then
        enShiftCounter <='1';
        DONE <='0';
        nCS <='0';
        enParalelLoad <= '0';
     else --if current_state = SyncData then
        enShiftCounter <='0';
        DONE <='0';
        nCS <='1';
        enParalelLoad <= '1';
     end if;
  end process;

   NEXT_STATE_DECODE: process (current_state, START, shiftCounter)
  begin

    next_state <= current_state;  -- default is to stay in current state

    case (current_state) is
      when Idle =>
        if START = '1' then
          next_state <= ShiftIn;
        end if;
      when ShiftIn =>
        if shiftCounter = x"F" then
          next_state <= SyncData;
        end if;
      when SyncData =>
        if START = '0' then
        next_state <= Idle;
        end if;
      when others =>
```

```
            next_state <= Idle;
        end case;
    end process;


end AD1;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DA2RefComp is
    Port (

     --General usage
       CLK      : in std_logic;    -- System Clock (50MHz)
       RST      : in std_logic;

     --Pmod interface signals
       D1       : out std_logic;
       D2       : out std_logic;
       CLK_OUT  : out std_logic;
       nSYNC    : out std_logic;

     --User interface signals
       DATA1    : in std_logic_vector(11 downto 0);
       DATA2    : in std_logic_vector(11 downto 0);
       START    : in std_logic;
       DONE     : out std_logic

        );
end DA2RefComp ;

architecture DA2 of DA2RefComp is


          constant control     : std_logic_vector(3 downto 0) := "0000";

          type states is (Idle,
                          ShiftOut,
                          SyncData);
          signal current_state : states;
          signal next_state    : states;

          signal temp1         : std_logic_vector(15 downto 0);
          signal temp2         : std_logic_vector(15 downto 0);
          signal clk_div       : std_logic;
          signal clk_counter   : std_logic_vector(27 downto 0);
          signal shiftCounter  : std_logic_vector(3 downto 0);
```

```
        signal enShiftCounter: std_logic;
        signal enParalelLoad : std_logic;



begin

      clock_divide : process(rst,clk)
      begin
         if rst = '1' then
            clk_counter <= "00000000000000000000000000000000";
         elsif (clk = '1' and clk'event) then
            clk_counter <= clk_counter + '1';
         end if;
      end process;

      clk_div <= clk_counter(0);
      clk_out <= clk_counter(0);

counter : process(clk_div, enParalelLoad, enShiftCounter)
      begin
         if (clk_div = '1' and clk_div'event) then
           if enParalelLoad = '1' then
              shiftCounter <= "0000";
              temp1 <= control & DATA1;
              temp2 <= control & DATA2;
            elsif (enShiftCounter = '1') then
              temp1 <= temp1(14 downto 0)&temp1(15);
              temp2 <= temp2(14 downto 0)&temp2(15);
              shiftCounter <= shiftCounter + '1';
            end if;
         end if;
      end process;

                D1 <= temp1(15);
                D2 <= temp2(15);

SYNC_PROC: process (clk_div, rst)
  begin
    if (clk_div'event and clk_div = '1') then
      if (rst = '1') then
        current_state <= Idle;
      else
        current_state <= next_state;
      end if;
    end if;
  end process;

OUTPUT_DECODE: process (current_state)
  begin
```

```
    if current_state = Idle then
        enShiftCounter <='0';
        DONE <='1';
        nSYNC <='1';
        enParalelLoad <= '1';
      elsif current_state = ShiftOut then
        enShiftCounter <='1';
        DONE <='0';
        nSYNC <='0';
        enParalelLoad <= '0';
      else --if current_state = SyncData then
        enShiftCounter <='0';
        DONE <='0';
        nSYNC <='1';
        enParalelLoad <= '0';
      end if;
  end process;

  NEXT_STATE_DECODE: process (current_state, START, shiftCounter)
  begin

    next_state <= current_state;  --default is to stay in current state

    case (current_state) is
      when Idle =>
        if START = '1' then
          next_state <= ShiftOut;
        end if;
      when ShiftOut =>
        if shiftCounter = x"F" then
          next_state <= SyncData;
        end if;
      when SyncData =>
        if START = '0' then
        next_state <= Idle;
        end if;
      when others =>
        next_state <= Idle;
    end case;
  end process;


end DA2;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
--use IEEE.std_logic_signed.all;--
library UNISIM;
use UNISIM.VComponents.all;

entity thermal is
   Port ( clk : in  STD_LOGIC;--50mhz i/p
                        ck_adc: in std_logic;
                       --  clk100 : in std_logic;--100mhz i/p
          final_temp : in  integer range 0 to 16383;
                        start : in  STD_LOGIC;--0=start
                        rate : in  integer range 0 to 1048575;--66000;
          hold_time : in  integer range 0 to 3600;
                        ramp_rs : out  integer range 0 to 16383;
                        ramp_t : out  integer range 0 to 16383;
                        err : out  integer range 0 to 16383;------
                        feedback : in integer range 0 to 16383;
                        filter_fb: out integer range 0 to 16383;
                        adc_no: out std_logic;
          pwm1 : out  STD_LOGIC;
          pwm2 : out  STD_LOGIC;
                        en : out std_logic;
                        new_comd: in std_logic:= '0';
                       --  data_cnt: in integer range 0 to 39;
                        st : out std_logic;
                        over : out std_logic;
                        debug1 : out std_logic
                        );
end thermal;

architecture Behavioral of thermal is

signal ramp,fb_tc : integer range 0 to 16383 := 0;--o/p of ramp generator,i/p of error amp
signal on_time: integer range 0 to 1799:= 0;--1800
signal clk_r: std_logic;

COMPONENT pid
        PORT(
                clk : IN std_logic;
                ck_adc: in std_logic;
                r_ck: in std_logic;
                rate : IN integer range 0 to 1048575;--66000;
                ramp : IN integer range 0 to 16383;
                feedback : IN integer range 0 to 16383;
                filter_fb: out integer range 0 to 16383;
                on_time : OUT integer range 0 to 1798;
                ramp_rs : OUT integer range 0 to 16383;
                err : OUT integer range 0 to 16383;
                ramp_t : OUT integer range 0 to 16383
                );
        END COMPONENT;
```

```
COMPONENT ramp1-- can be a max error of 33us in holdtime
	PORT(
		r_clk : IN std_logic;
		ck_r : out std_logic;
		--config_h : IN std_logic;
		rate : IN integer range 0 to 1048575;--66000;
		final_temp : IN integer range 0 to 16383;
		hold_time : IN integer range 0 to 3600;
		ramp_out : OUT integer range 0 to 16383;
		new_comd: in std_logic;
		over : out std_logic
		);
	END COMPONENT;

COMPONENT pm1
	PORT(
		clock : IN std_logic;
		start : IN std_logic;
		on_time : IN integer range 0 to 1799;
		adc_no_p : out std_logic;
		en : out std_logic;
		pwm1 : OUT std_logic;
		pwm2 : OUT std_logic;
		--e : out  std_logic;
		--ramp_t : out  integer range 0 to 16383;
		debug1 : out std_logic
		);
	END COMPONENT;

begin

pid_contr: pid PORT MAP(
		clk => clk,
		ck_adc => ck_adc,
		r_ck => clk_r,
		rate => rate,
		ramp => ramp,
		feedback => feedback,
		filter_fb => filter_fb,
		on_time => on_time,
		ramp_rs => ramp_rs,
		err => err,
		ramp_t => ramp_t
	);

ramp_generator: ramp1 PORT MAP(
		r_clk => clk,--r_ck,
		ck_r => clk_r,
```

```
                rate => rate,
                final_temp => final_temp,
                hold_time => hold_time,
                ramp_out => ramp,
                new_comd => new_comd,
                over => over
        );


pwm: pm1 PORT MAP(
                clock => clk,--b_clk,--clk100
                start => start,
                on_time => on_time,--
                adc_no_p => adc_no,
                en => en,
                pwm1 => pwm1,
                pwm2 => pwm2,
                debug1 => debug1
        );

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use IEEE.std_logic_signed.all;--
library UNISIM;
use UNISIM.VComponents.all;

entity pid is
   Port ( clk : in  STD_LOGIC;--50mhz i/p
                                ck_adc: in std_logic;
                                r_ck : in std_logic;
                        rate : in  integer range 0 to 1048575;--66000;
                        ramp : in  integer range 0 to 16383;
                        feedback : in integer range 0 to 16383;
                        filter_fb: out integer range 0 to 16383;
                    on_time : out integer range 0 to 1798;
                        ramp_rs, err,ramp_t : out integer range 0 to 16383
                        );
end pid;

architecture Behavioral of pid is

signal error, ramp_12, er, er1, in1, d1,d2, er_dp,er_dn,t1,t3,t4,t5,t12,t345 : integer range 0 to
16383 := 0;
signal b1,b2,er_i: integer range 0 to 16383;
signal cntp, cntn: integer range 0 to 14000;
```

```
signal sg, g6,d3 : integer range 0 to 1800:= 0;

begin

filter_fb <= b2; --feedback;--

b2 <=  b1 when b1 < 520 else --63
                        b1 - 10 when (b1 >= 520 and b1 < 1491) else
                        b1;

process(r_ck)
begin
if (r_ck = '0' and r_ck'event) then-- -ve edge

        if ramp < 2 then
                g6 <= 0;
        elsif error > 0 and g6 < 1299 then--699--349--647 743 793 1690 1199 1299
                g6 <= g6 + 1;
        elsif error < 1 and g6 > 0 then
                g6 <= g6 - 1;
        else
                g6 <= g6;
        end if;
end if;
end process;

process(ck_adc)
begin
if (ck_adc = '1' and ck_adc'event) then-- -ve edge
        if error > 0 and d1 < 499 then--254 159 109 599 499
                d1 <= d1 + 1;
        elsif error < 1 and d1 > 0 then
                d1 <= d1 - 1;
        else
                d1 <= d1;
        end if;
end if;
end process;

process(clk)
begin
if (clk = '0' and clk'event) then-- -ve edge
        if rate = 100 then
                sg <= g6;
        else
                sg <= g6 + d1;
        end if;
        if cntp < 1798 then
                cntp <= cntp + 1;
        else
```

```
                cntp <= 0;
        end if;

        if cntn < 1798 then--1798
                cntn <= cntn + 1;
        else
                cntn <= 0;
                if er < error and er < 4095 then--error--this could b deleated
                        er <= er + 1;
                elsif er > error and er > 0 then
                        er <= er - 1;
                else
                        er <= er;
                end if;-----
                if b1 < feedback then--error
                        b1 <= b1 + 1;
                elsif b1 > feedback then
                        b1 <= b1 - 1;
                else
                        b1 <= b1;
                end if;
        end if;
end if;
end process;

process (clk)
begin
if (clk = '0' and clk'event) then-- -ve edge
        ramp_12 <= ramp/4;

                        if (rate < 1000 and sg > 1798) then
                                on_time <= 1798;
                        else
                                on_time <= sg;--g6;
                        end if;
end if;
end process;

process (clk)
        begin

        if (clk'event and clk = '1') then

                if ramp_12 > b2 then--b1
                        error <= ramp_12 - b2;--b1
                else
                        error <= 0;
                end if;
end if;
end process;
```

```
ramp_rs <= g6;--t1;--error;--g6;--error;--g3;--g1;--ramp; tl white
err <= error;--g2;--b2;--n;--g6;--g3;--on_time;--on_time;--error; g2 tl red
ramp_t <= ramp_12;--g6;--on_time;--ramp; g1 osl red

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity ramp1 is
    Port ( r_clk : in  STD_LOGIC; -- 8 mhz ,now 50 mhz
                                    ck_r : out std_logic;
                                --  ck_ready : in  STD_LOGIC;
                                --  config_h : in  STD_LOGIC;--:= '1';--0=process begins-----deleat
this
                                 rate: in integer range 0 to 1048575;--66000;-- x 'c/s = 4000/x count.
this should be the value of rate,.1 upwards
                                 final_temp: in integer range 0 to 16383 := 0;-- X'c = 65.536X count.
this should be the value of final_temp.16384/250(temp range) = 65.536
                                 hold_time: in integer range 0 to 3600;--give this value directly in
seconds
                                 ramp_out :out integer range 0 to 16383;--16383;
                                 new_comd: in std_logic:= '0';--intial value needed??? 0
                                 over : out std_logic:= '0'

                                );
end ramp1;

architecture Behavioral of ramp1 is

signal stp: integer range 0 to 63579 := 0;--counter for clk generator of ramp counter--63579
signal clk_r: std_logic := '0';--clk for ramp counter
signal clk_32: std_logic := '0';--clk for rate counter
signal hold: std_logic:='1' ;-- 0 = hold-time counter on
signal reset: std_logic := '1';-- 0 = ramp value becomes 0, hold = 1. it happens at the end of
holdtime
signal start: std_logic:= '0';
signal start_1: std_logic:= '0';--1=ramp counter is on, it becomes 0 when hold-time bidgins
signal stop: std_logic := '1';--0=clk generation for ramp stops. it happens when fianl temp is
reached
signal count_32: integer range 0 to 790:= 0;--counter for 32khz generation
signal count: integer range 0 to 16383:= 0;--counter for ramp generation
signal hld_value: integer range 0 to 3600:= 0;-- counter for hold_time
signal hld: integer range 0 to 833333 := 0;--counter for the one second counter
signal hold_s :std_logic := '1';
```

```
signal hold_f, ck_st : std_logic;
signal step : std_logic := '0';
signal count_c: integer range 0 to 1048575 := 0;

begin
--ck_r <= ck_st when rate = 100 else clk_r;
process (r_clk)--produces 833.333 khz

begin

  if (r_clk'event and r_clk='1') then       -- CLK rising edge

        if rate = 100 then
              ck_r <= ck_st;
        else
              ck_r <= clk_r;
        end if;

        if stp < 15000 then
              stp <= stp;
        else
              stp <= 0;
              ck_st <= not ck_st;
        end if;

        if reset = '0' then
              step <= '0';
        elsif rate = 100 then--
              step <= '1';
        else
              step <= '0';
        end if;

        if step = '1' then
              ramp_out <= final_temp;
        else
              ramp_out <= count;
        end if;


              if new_comd = '1' then
                    start <= '1';--this will make the count(=ramp)0
              elsif count = 0 then--this is done so that a new command can be accepted
                    start <= '0';
              end if;

              if (count_32 < 29) then--833.333khz clk generation from 50mhz
                    count_32 <= count_32 + 1;
              else
                    count_32 <= 0;
```

```
                        clk_32 <= not clk_32;
                end if;
        end if;
end process;

process (clk_32)--produces clk for ramp counter; uses 833.33 khz clk

begin

if (clk_32'event and clk_32='1') then       -- CLK rising edge
        if (count_c < rate) then--rate = ramp rate
                count_c <= count_c + 1;
        else
                count_c <= 0;
                clk_r <= not clk_r;
        end if;
end if;

end process;

process (clk_r) -- ramp counter; this process generates the ramp

begin

if (clk_r'event and clk_r ='1') then       -- CLK rising edge; can make this -ve edge and test

if start = '1' then --or reset = '0'--start = 1; for new command ramp is initialized to 0
        count <=0;--0
        hold <= '1';--1=holdtime off, 0=holdtime on
elsif step = '1' then
        hold <= '0';
elsif (count < final_temp) then
        count <= count + 1;
else
        count <= count; --here the ramp has reached required temperture and is at that
temperture for the duration of holdtime
        hold <= '0';-- due to this the holdtime counter begins
end if;


if reset = '0' then--when holdtime is over reset becomes 0. it becomes 1 when a new
command comes
count <= 0;
hold <= '1';--holdtime counter is off
end if;
end if;

end process;

process (clk_32) -- hold time counter
```

```
begin

if (clk_32'event and clk_32 = '0') then  --falling edge

        if new_comd = '1' then--initialization at new command
                reset <= '1';
                over <= '1';--this is to reset the 'new_comd' flag
                hld <= 0; --for 1 second counter
                hld_value <= 0;-- for hold time counter; this increments at every 1 second
        elsif new_comd = '0' then
                        over <= '0';--reseting the flag which resets 'new_comd' after it has
reseted it.
        end if;

        if hold = '0' then
                if (hld_value < hold_time) then-- hold_time
                ------------------------------------------------------
                        if hld < 833333 then--one second counter
                                hld <= hld + 1;
                        else
                                hld <= 0;
                                hld_value <= hld_value + 1;
                        end if;
                ------------------------------------------------------
                else
                        if (hld_value > hold_time or hld_value = hold_time) then--holdtime is
over
                                hld <= 0;
                                hld_value <= 0;
                                reset <= '0';--at this the ramp becomes 0
                        end if;
                end if;
        end if;
end if;
end process;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity pm1 is--pwm produced at 27.77 Khz
   Port ( clock : in  Std_logic;
                        start : in  Std_logic;-- 0 = start
                        on_time : in integer range 0 to 1799;
                       adc_no_p : out  std_logic ;
                        en : out  std_logic ;
```

```
                pwm1 : out  std_logic := '0';
                pwm2 : out  std_logic := '0';
                debug1: out std_logic);
        end pm1;

architecture Behavioral of pm1 is

signal pwm_selector, adc_no, flag, en_s: std_logic := '0';
signal flg, flg2: std_logic := '0';
signal pwm_out: std_logic := '0';
signal count, count_c,t1, t2,t3: integer range 0 to 2802 := 0;

begin

adc_no_p <= adc_no;-- when flag = '1' else '0';
en <= '1';--en_s;

process (clock)-- to calculate the sampling point
begin

if (clock'event and clock = '1') then
        if on_time > 0 then
                if (count = t1) then--or count = t2--t1
                        adc_no <= '1';
                else
                        adc_no <= '0';
                end if;
        else
                if count_c < 1800 then
                        count_c <= count_c + 1;
                        if count_c = 0 or count_c = 450 or count_c = 900 or count_c = 1350
then
                                adc_no <= '1';
                        else
                                adc_no <= '0';
                        end if;
                else
                        count_c <= 0;
                end if;
        end if;
end if;
end process;

process (clock)--pwm o/p will change every 33 us(36 us) i.e. 30 khz
begin

if (clock'event and clock='1') then  -- CLK rising edge
        --if start = '0' then
                if on_time = 0 then
                        pwm_out <= '0';
```

```
                elsif (count < 1798) then--703 901
                        if (count = on_time) then --on_time
                                pwm_out <= '0';
                        end if;
                        count <= count + 1;
                else
                        pwm_selector <= not pwm_selector;
                        count <= 0;
                        if on_time > 0 then
                                pwm_out <= '1';
                        end if;
                end if;
end if;
end process;

                pwm1 <= pwm_out and pwm_selector;
                pwm2 <= pwm_out and (not pwm_selector);
debug1 <= pwm_out;

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.std_logic_signed.all;--
library UNISIM;
use UNISIM.VComponents.all;

entity optical is
   Port ( clk : in  STD_LOGIC;--50mhz i/p
                        final_intensity : in  integer range 0 to 16383;
                        adc_sp: in std_logic;
                        rate : in  integer range 0 to 1048575;--66000;
                        hold_time : in  integer range 0 to 3600;
                        ramp_rs : out  integer range 0 to 16383;
                        ramp_dac: out integer range 0 to 4095;
                        new_comd: in std_logic:= '0';
                        adc : in integer range 0 to 65535;---light feedback
                        over : out std_logic
                        );
end optical;

architecture Behavioral of optical is

signal ck_o,ck_r,hdtme: std_logic;
signal ramp,diffp, diffn : integer range 0 to 16383 := 0;--o/p of ramp generator,i/p of error
amp
signal int : integer range 0 to 8800;--1800
```

```
signal rmp_dac,ramp_4,adc_4,d2,d1,dr:integer range 0 to 4095;

COMPONENT optical_ramp-- can be a max error of 33us in holdtime
	PORT(
		r_clk : IN std_logic;
		rate : IN integer range 0 to 1048575;--66000;
		final_temp : IN integer range 0 to 16383;
		hold_time : IN integer range 0 to 3600;
		ramp_out : OUT integer range 0 to 16383;
		new_comd: in std_logic;
		over : out std_logic;
		r_ck: out std_logic;
		hldtim: out std_logic
		);
	END COMPONENT;

begin

 op_ramp_gen: optical_ramp PORT MAP(
		r_clk => clk,--r_ck,
		rate => rate,
		final_temp => final_intensity,
		hold_time => hold_time,
		ramp_out => ramp,
		new_comd => new_comd,
		over => over,
		r_ck => ck_r,
		hldtim => hdtme
	);


process(clk)
begin
if clk'event and clk = '1' then
	ramp_4 <= ramp/4;

	if hdtme = '0'  then
		d1 <= d2;
	else
		d1 <= rmp_dac + d2;
	end if;

	if adc > 150 then
		diffn <= adc - 150;--150
	else
		diffn <= 0;
	end if;

	if adc < 16333 then
		diffp <= adc + 150;--150
```

```
        else
                diffp <= 16383;
        end if;
end if;
end process;

process(clk)
begin
if clk'event and clk = '0' then
        ramp_dac <= d1;
end if;
end process;

process(ck_r)
begin
if ck_r'event and ck_r = '0' then
                if ramp < 2 then----------------------------
                        rmp_dac <= 0;
                elsif ramp > adc and rmp_dac < 4095 then--ramp_4 ramp
                        rmp_dac <= rmp_dac + 1;
                elsif adc > ramp and rmp_dac > 0 then--ramp
                        rmp_dac <= rmp_dac - 1;
                else
                        rmp_dac <= rmp_dac;
                end if;
        end if;
end process;

dr <= 3195 when (rate = 1000 or hdtme = '0') else 100;--4095   3895          3695

process(adc_sp)
begin
if adc_sp = '0' and adc_sp'event then
        if ramp > diffp and d2 < dr then--ramp
                d2 <= d2 + 1;
        elsif diffn > ramp and d2 > 0 then--ramp
                d2 <= d2 - 1;
        else
                d2 <= d2;
        end if;
end if;
end process;
ramp_rs <= ramp/4;

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
library UNISIM;
use UNISIM.VComponents.all;

entity optical_ramp is
    Port ( r_clk : in  STD_LOGIC; -- 8 mhz ,now 50 mhz
                              --  st : in  STD_LOGIC;
                                rate: in integer range 0 to 1048575;--66000;-- x 'c/s = 4000/x count.
this should be the value of rate,.1 upwards
                                final_temp: in integer range 0 to 16383 := 0;-- X'c = 65.536X count.
this should be the value of final_temp.16384/250(temp range) = 65.536
                                hold_time: in integer range 0 to 3600;--give this value directly in
seconds
                                ramp_out :out integer range 0 to 16383;--16383;
                                new_comd: in std_logic:= '0';--intial value needed??? 0
                                over : out std_logic:= '0';
                                r_ck: out std_logic;
                                hldtim: out std_logic
                               );
end optical_ramp;

architecture Behavioral of optical_ramp is

signal count_c: integer range 0 to 1048575 := 0;--counter for clk generator of ramp counter--
63579
signal clk_r: std_logic := '0';--clk for ramp counter
signal clk_32: std_logic := '0';--clk for rate counter
signal hold: std_logic:='1' ;-- 0 = hold-time counter on
signal reset: std_logic := '1';-- 0 = ramp value becomes 0, hold = 1. it happens at the end of
holdtime
signal start: std_logic:= '0';
signal start_1: std_logic:= '0';--1=ramp counter is on, it becomes 0 when hold-time bidgins
signal stop: std_logic := '1';--0=clk generation for ramp stops. it happens when fianl temp is
reached
signal count_32: integer range 0 to 790:= 0;--counter for 32khz generation
signal count: integer range 0 to 16383:= 0;--counter for ramp generation
signal hld_value: integer range 0 to 3600:= 0;-- counter for hold_time
signal hld: integer range 0 to 833333 := 0;--counter for the one second counter
signal hold_s :std_logic := '1';
signal hold_f : std_logic;
signal step : std_logic := '0';


begin

hldtim <= hold;
r_ck <= clk_r;

process (r_clk)--produces 833.333 khz
begin
```

```
  if (r_clk'event and r_clk='1') then       -- CLK rising edge
        if reset = '0' then
              step <= '0';
        elsif rate = 1000 then
              step <= '1';
        else
              step <= '0';
        end if;

        if step = '1' then
              ramp_out <= final_temp;
        else
              ramp_out <= count;
        end if;

              if new_comd = '1' then
                    start <= '1';--this will make the count(=ramp)0
              elsif count = 0 then--this is done so that a new command can be accepted
                    start <= '0';
              end if;
              if (count_32 < 29) then--833.333khz clk generation from 50mhz
                    count_32 <= count_32 + 1;
              else
                    count_32 <= 0;
                    clk_32 <= not clk_32;
              end if;
        end if;
end process;

process (clk_32)--produces clk for ramp counter; uses 833.33 khz clk

begin

if (clk_32'event and clk_32='1') then       -- CLK rising edge
        if (count_c < rate) then--rate = ramp rate
              count_c <= count_c + 1;
        else
              count_c <= 0;
              clk_r <= not clk_r;
        end if;
end if;

end process;

process (clk_r) -- ramp counter; this process generates the ramp
begin

if (clk_r'event and clk_r ='1') then       -- CLK rising edge; can make this -ve edge and test
```

```
if start = '1' then --or reset = '0'--start = 1; for new command ramp is initialized to 0
        count <=0;--0
        hold <= '1';--1=holdtime off, 0=holdtime on
elsif step = '1' then
        hold <= '0';
elsif (count < final_temp) then
        count <= count + 1;
else
        count <= count; --here the ramp has reached required temperture and is at that
temperture for the duration of holdtime
        hold <= '0';-- due to this the holdtime counter begins
end if;

if reset = '0' then--when holdtime is over reset becomes 0. it becomes 1 when a new
command comes
count <= 0;
hold <= '1';--holdtime counter is off
end if;

end if;
end process;

process (clk_32) -- hold time counter

begin

if (clk_32'event and clk_32 = '0') then  --falling edge

        if new_comd = '1' then--initialization at new command
                reset <= '1';
                over <= '1';--this is to reset the 'new_comd' flag
                hld <= 0; --for 1 second counter
                hld_value <= 0;-- for hold time counter; this increments at every 1 second
        elsif new_comd = '0' then
                        over <= '0';--reseting the flag which resets 'new_comd' after it has
reseted it.
        end if;

        if hold = '0' then
                if (hld_value < hold_time) then-- hold_time
                ------------------------------------------------------
                        if hld < 833333 then--one second counter
                                hld <= hld + 1;
                        else
                                hld <= 0;
                                hld_value <= hld_value + 1;
                        end if;
                ------------------------------------------------------
                else
```

```
                        if (hld_value > hold_time or hld_value = hold_time) then--holdtime is
over
                                hld <= 0;
                                hld_value <= 0;
                                reset <= '0';--at this the ramp becomes 0
                        end if;
                end if;
        end if;
end if;
end process;
end Behavioral;
```

```
NET "serial_out" LOC = "T7";
NET "serial_in" LOC = "R7" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW =
SLOW ;

NET "clk50" LOC = "C10";# 100 MHz clk

NET "nCS" LOC = "K12";     # adc cs--coneector J4 k12
NET "SDATA1" LOC = "K13"; # temp input k13
NET "SDATA2" LOC = "F17"; # light input f17
NET "SCLK" LOC = "F18";  # adc clk f18

NET "n_sync" LOC = "H12"; # dac syn for 2 ch  "H12"
NET "blue" LOC = "G13"; # dac 0/p for osl--dac is on J4  "G13"
NET "ir" LOC = "E16"; # for futrue expansion "e16"
NET "ck_dac" LOC = "E18"; # dac ck "e18"

#NET "dac_pw" LOC = "F14"; # power on-off for dac

NET "emccd" LOC = "F15"; # start trigger for EMCCD #--coneector J5
NET "en" LOC = "F16"; # H-bridge enable
NET "pwm_c" LOC = "C17";
NET "pwm_d" LOC = "C18";
```
