Lab 8 - ADC

Nate Herbst
A02307138
Nathan Walker
A02364124

Introduction

This project required configuring a DE10-Lite development board to take an ADC measurement at a 1 Hz rate, display the ADC reading in hexadecimal on seven-segment displays, and interface a potentiometer. Adjusting the potentiometer alters the displayed value, providing a visual representation of changing ADC readings. The system includes a reset button.

Procedure

Project Setup:

- The DE10-Lite board was configured in Quartus to initialize the ADC module and display hex values on the seven-segment displays. The potentiometer was connected to provide an adjustable input for the ADC.

Configuring the ADC and Seven-Segment Display:

- We implemented a module to capture ADC readings and convert them to a hexadecimal format compatible with the seven-segment display outputs. Each display segment was mapped to reflect the hex values directly from the ADC.

Challenges and Solutions:

- Inverted Reset Pin: An unexpected issue arose from the auto-generated ADC module, where the reset pin was already inverted. Initially, our code manually inverted the reset signal, which resulted in unintended behavior. Upon recognizing this, we corrected our code to align with the pre-inverted signal, ensuring proper reset functionality.
- Syntax Errors: Frequent syntax errors slowed down our progress. These issues were resolved by thorough debugging and referring to VHDL syntax rules, reinforcing our understanding of syntax constraints and improving code accuracy.

Results

The final implementation displayed real-time ADC values on the seven-segment displays and responded dynamically to changes in the potentiometer's position. The reset functionality worked as expected after adjustments. No unexpected behaviors were observed in the final design, and all components functioned according to specifications.

Figures

1. **Wiring of the ADC Connection** (Figure 1): Shows connections between the ADC module and seven-segment displays.

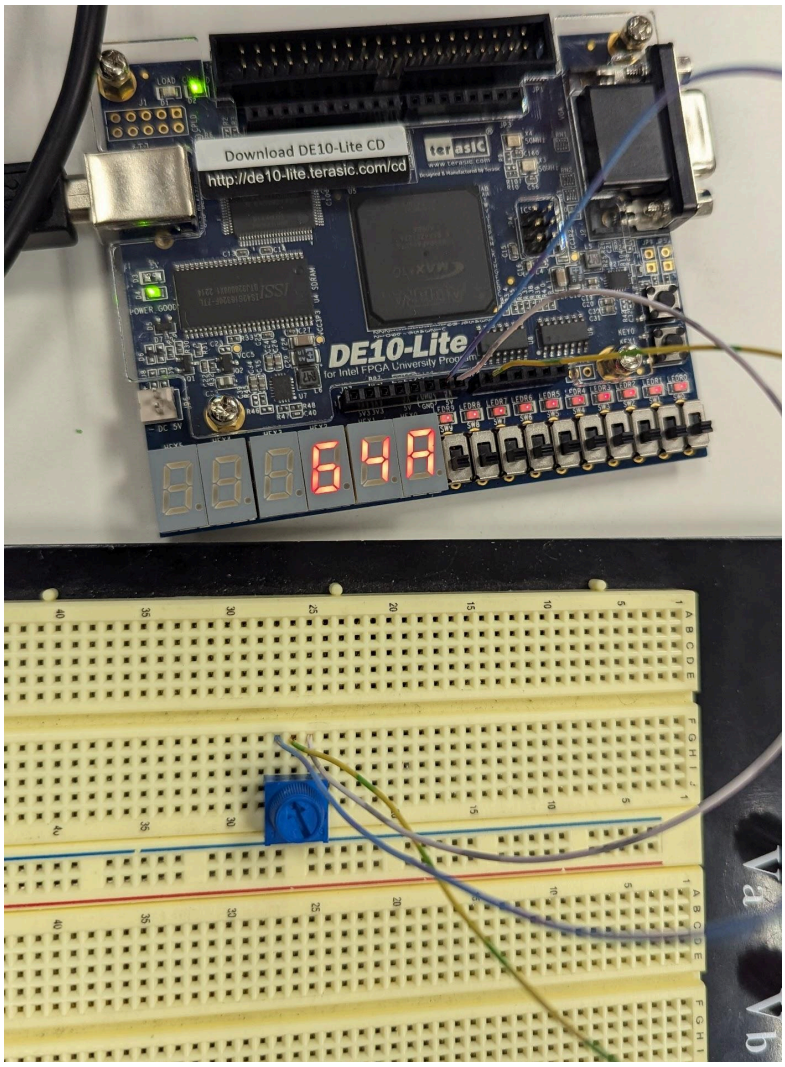

Figure 1: Wiring of Circuit

Conclusion

This lab provided valuable insights into FPGA configuration, ADC interfacing, and VHDL troubleshooting. Resolving issues such as the inverted reset pin and syntax errors helped refine our approach to debugging and code management, reinforcing best practices for FPGA development.

Appendix

Lab8_ADC.vhd top level file

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

entity Lab8_ADC is
   port (
       ADC_CLK_10 : in std_logic;
       MAX10_CLK1_50 : in std_logic;
       MAX10_CLK2_50 : in std_logic;

       KEY : in std_logic_vector(1 downto 0);

       HEX0 : out std_logic_vector(7 downto 0);
       HEX1 : out std_logic_vector(7 downto 0);
       HEX2 : out std_logic_vector(7 downto 0);
       HEX3 : out std_logic_vector(7 downto 0);
       HEX4 : out std_logic_vector(7 downto 0);
       HEX5 : out std_logic_vector(7 downto 0)
   );
end Lab8_ADC;

architecture component_list of Lab8_ADC is

   component debouncer
       port (
           clk : in std_logic;
           btn : in std_logic;
           output : out std_logic
```

```
    );
end component debouncer;

component HEX_seven_seg_disp_6
    port (
        IN0 : in std_logic_vector(3 downto 0);
        IN1 : in std_logic_vector(3 downto 0);
        IN2 : in std_logic_vector(3 downto 0);
        IN3 : in std_logic_vector(3 downto 0);
        IN4 : in std_logic_vector(3 downto 0);
        IN5 : in std_logic_vector(3 downto 0);
        clk : in  std_logic;
        HEX0 : out std_logic_vector(7 downto 0);
        HEX1 : out std_logic_vector(7 downto 0);
        HEX2 : out std_logic_vector(7 downto 0);
        HEX3 : out std_logic_vector(7 downto 0);
        HEX4 : out std_logic_vector(7 downto 0);
        HEX5 : out std_logic_vector(7 downto 0)
    );
end component HEX_seven_seg_disp_6;

component ADC
    -- generic (
    --  N : integer := 10000000
    -- );
    port (
        clk : in std_logic;
        btn : in std_logic;
        output : out std_logic_vector(11 downto 0)
    );
end component ADC;

signal rst_btn : std_logic;
signal en : std_logic;

signal key0_l : std_logic;
signal key1_l : std_logic;

signal sum      : std_logic_vector(11 downto 0);
signal turn_off : std_logic_vector(7 downto 0);
```

```
begin

   adc1 : ADC
       -- generic map (
       --  N => 10000000
       -- )
       port map (
           clk => ADC_CLK_10,
           btn => not rst_btn,
           output => sum
       );

   display : HEX_seven_seg_disp_6
       port map (
           clk => ADC_CLK_10,
           IN0 => sum(3 downto 0),
           IN1 => sum(7 downto 4),
           IN2 => sum(11 downto 8),
           IN3 => turn_off(3 downto 0),
           IN4 => turn_off(3 downto 0),
           IN5 => turn_off(3 downto 0),
           HEX0 => HEX0,
           HEX1 => HEX1,
           HEX2 => HEX2,
           HEX3 => turn_off,
           HEX4 => turn_off,
           HEX5 => turn_off
       );

   rst_bbtn : debouncer
       port map (
           clk => ADC_CLK_10,
           btn => key0_l,
           output => rst_btn
       );

   key0_l <= not KEY(0);
   -- key1_l <= not KEY(1);
   HEX3 <= (others => '1');
```

```
   HEX4 <= (others => '1');
   HEX5 <= (others => '1');

end component_list;
```

ADC.vhd

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

entity ADC is
   -- generic (
   --  N : integer := 10000000
   -- );
   port (
       clk : in std_logic;
       btn : in std_logic;
       output : out std_logic_vector(11 downto 0)
   );
end;

architecture counter of ADC is
   component base_ADC is
       port (
           clock_clk               : in  std_logic                     :=
'X';             -- clk
           reset_sink_reset_n      : in  std_logic                     :=
'X';             -- reset_n
           adc_pll_clock_clk       : in  std_logic                     :=
'X';             -- clk
           adc_pll_locked_export   : in  std_logic                     :=
'X';             -- export
           command_valid           : in  std_logic                     :=
'X';             -- valid
           command_channel         : in  std_logic_vector(4 downto 0)  :=
(others => 'X'); -- channel
           command_startofpacket   : in  std_logic                     :=
'X';             -- startofpacket
```

```
            command_endofpacket     : in  std_logic                     :=
'X';             -- endofpacket
            command_ready           : out std_logic;
-- ready
            response_valid          : out std_logic;
-- valid
            response_channel        : out std_logic_vector(4 downto 0);
-- channel
            response_data           : out std_logic_vector(11 downto 0);
-- data
            response_startofpacket  : out std_logic;
-- startofpacket
            response_endofpacket    : out std_logic
-- endofpacket
        );
    end component base_ADC;

    component PLL_10M is
        port (
            inclk0                  : IN STD_LOGIC  := '0';
            c0                      : OUT STD_LOGIC ;
            locked                  : OUT STD_LOGIC
        );
    end component PLL_10M;

    signal s_clock_clk              : std_logic;
    -- signal s_reset_sink_reset_n     : std_logic;
    signal s_adc_pll_clock_clk      : std_logic;
    signal s_adc_pll_locked_export  : std_logic;
    signal s_command_valid          : std_logic := '1';
    signal s_command_channel        : std_logic_vector(4 downto 0) :=
"00001";
    signal s_command_startofpacket  : std_logic := '1';
    signal s_command_endofpacket    : std_logic := '1';
    signal s_command_ready          : std_logic;
-- ready
    signal s_response_valid         : std_logic;
-- valid
    signal s_response_channel       : std_logic_vector(4 downto 0);
    signal s_response_data          : std_logic_vector(11 downto 0);
```

```
   signal s_response_startofpacket : std_logic;
-- startofpacket
   signal s_response_endofpacket   : std_logic;
-- endofpacket

   signal temp_Data                : std_logic_vector(11 downto 0);


   signal count : integer := 0;
   -- signal v_counter : integer := 0;
begin

   u0 : component base_ADC
       port map (
           clock_clk               => clk,               --
clock.clk
           reset_sink_reset_n      => btn,      --      reset_sink.reset_n
           adc_pll_clock_clk       => s_adc_pll_clock_clk,       --
adc_pll_clock.clk
           adc_pll_locked_export   => s_adc_pll_locked_export,   --
adc_pll_locked.export
           command_valid           => s_command_valid,           --
command.valid
           command_channel         => s_command_channel,         --
.channel
           command_startofpacket   => s_command_startofpacket,   --
.startofpacket
           command_endofpacket     => s_command_endofpacket,     --
.endofpacket
           command_ready           => s_command_ready,           --
.ready
           response_valid          => s_response_valid,          --
response.valid
           response_channel        => s_response_channel,        --
.channel
           response_data           => s_response_data,           --
.data
           response_startofpacket => s_response_startofpacket, --
.startofpacket
```

```
            response_endofpacket    => s_response_endofpacket    --
.endofpacket
        );

    PLL_10M_inst : PLL_10M PORT MAP (
        inclk0   => clk,
        c0   => s_adc_pll_clock_clk,
        locked   => s_adc_pll_locked_export
    );

    proc1: process(clk)
    begin
        if rising_edge(clk) then
            if btn = '0' then
                output <= (others => '0');
                count <= 0;
            elsif s_response_valid = '1' then
                temp_Data <= s_response_data;
            end if;
            if count = 10000000 then
                output <= temp_Data;
                count <= 0;
            else
                count <= count + 1;
            end if;
        end if;
    end process proc1;



end counter;
```

base_ADC.vhd

```
-- base_ADC.vhd

-- Generated using ACDS version 23.1 991

library IEEE;
```

```
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity base_ADC is
   port (
       adc_pll_clock_clk      : in  std_logic                     := '0';
--  adc_pll_clock.clk
       adc_pll_locked_export  : in  std_logic                     := '0';
-- adc_pll_locked.export
       clock_clk              : in  std_logic                     := '0';
--           clock.clk
       command_valid          : in  std_logic                     := '0';
--         command.valid
       command_channel        : in  std_logic_vector(4 downto 0)  :=
(others => '0'); --                .channel
       command_startofpacket  : in  std_logic                     := '0';
--                .startofpacket
       command_endofpacket    : in  std_logic                     := '0';
--                .endofpacket
       command_ready          : out std_logic;
--                .ready
       reset_sink_reset_n     : in  std_logic                     := '0';
--     reset_sink.reset_n
       response_valid         : out std_logic;
--       response.valid
       response_channel       : out std_logic_vector(4 downto 0);
--                .channel
       response_data          : out std_logic_vector(11 downto 0);
--                .data
       response_startofpacket : out std_logic;
--                .startofpacket
       response_endofpacket   : out std_logic
--                .endofpacket
   );
end entity base_ADC;

architecture rtl of base_ADC is
   component base_ADC_modular_adc_0 is
       generic (
           is_this_first_or_second_adc : integer := 1
```

```
        );
        port (
            clock_clk               : in  std_logic                     :=
'X';             -- clk
            reset_sink_reset_n      : in  std_logic                     :=
'X';             -- reset_n
            adc_pll_clock_clk       : in  std_logic                     :=
'X';             -- clk
            adc_pll_locked_export   : in  std_logic                     :=
'X';             -- export
            command_valid           : in  std_logic                     :=
'X';             -- valid
            command_channel         : in  std_logic_vector(4 downto 0)  :=
(others => 'X'); -- channel
            command_startofpacket   : in  std_logic                     :=
'X';             -- startofpacket
            command_endofpacket     : in  std_logic                     :=
'X';             -- endofpacket
            command_ready           : out std_logic;
-- ready
            response_valid          : out std_logic;
-- valid
            response_channel        : out std_logic_vector(4 downto 0);
-- channel
            response_data           : out std_logic_vector(11 downto 0);
-- data
            response_startofpacket  : out std_logic;
-- startofpacket
            response_endofpacket    : out std_logic
-- endofpacket
        );
    end component base_ADC_modular_adc_0;

begin

    modular_adc_0 : component base_ADC_modular_adc_0
        generic map (
            is_this_first_or_second_adc => 1
        )
        port map (
```

```
            clock_clk               => clock_clk,               --
clock.clk
            reset_sink_reset_n      => reset_sink_reset_n,      --
reset_sink.reset_n
            adc_pll_clock_clk       => adc_pll_clock_clk,       --
adc_pll_clock.clk
            adc_pll_locked_export   => adc_pll_locked_export,   --
adc_pll_locked.export
            command_valid           => command_valid,           --
command.valid
            command_channel         => command_channel,         --
.channel
            command_startofpacket   => command_startofpacket,   --
.startofpacket
            command_endofpacket     => command_endofpacket,     --
.endofpacket
            command_ready           => command_ready,           --
.ready
            response_valid          => response_valid,          --
response.valid
            response_channel        => response_channel,        --
.channel
            response_data           => response_data,           --
.data
            response_startofpacket => response_startofpacket, --
.startofpacket
            response_endofpacket    => response_endofpacket     --
.endofpacket
        );

end architecture rtl; -- of base_ADC
```