11/13/2024

Lab 08 Report

Ryan Beck
Jared Bronson

Objectives

For this lab, build a project on the DE10-LITE that takes an ADC measurement at a 1 Hz rate. Display the reading from the ADC on the appropriate number of seven-segment displays using hex values. Interface a potentiometer to the system and show the displayed value changing as the pot is adjusted. This should be implemented entirely in VHDL.

Procedures

Using the system builder create a Quartus project file with the on-board clock, the Arduino header, and the 6 seven-segment displays. Take the generated Verilog file and create a VHDL file with matching port names. After creating the VHDL file, take the Verilog file out of the project. In the generic, create a value for to compare to the sample counter.

Using the IP catalog, create an ADC. The ADC will have an input clock of 10 MHz and a sample rate of 1 MHz. Change the reference voltage to internal, and activate channel zero. Using the IP catalog again create a PLL to pass into the ADC. Change the frequency of the input clock to 10 MHz to match the ADC.

In the architecture, create components to match the generated files of the ADC and PLL. Declare signals to use for the ADC and PLL signals. Next, create a table with values that correspond to each of the 16 hex values for display. Last, create signals to hold the display value, a counter for samples, and a value for when the display updates.

Instantiate the IP blocks for the ADC and PLL. Then, define a process for the on-board clock to add to the sample counter if it is less than the sample period value in the generic. If the sample counter is greater than the sample period, the counter will reset, and the display trigger will be driven high.

Define another process sensitive to the on-board clock. This process will look at the response valid output from the ADC. If the response valid goes high, the response data output will be scaled to a voltage and stored in the display.

The last process will again be sensitive to the on-board clock. This process will update the seven-segment display with the reading from the ADC. Next, compile the design and program to the development board for final testing.

Results

The ADC file, shown in figures, successfully reads voltage passed into channel zero of the ADC. The display is updated at a 1 Hz rate on the last three of the seven-segment displays. This is because the data read from the ADC is a 12-bit value. Using a potentiometer interfaced to the ADC, the voltage read decreases with more resistance, and increases with less resistance. This is done using only VHDL.

Conclusion

In conclusion, we were able to read an input voltage passed into the ADC and display it on 3 seven-segment displays. This was done using only VHDL in our Quartus project. The value of the voltage is shown using hex values, and is updated every second, or at a 1 Hz rate. A potentiometer successfully shows the value increasing and decreasing with varying voltage.

Figures

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity adc is
6
7      generic(
8         --     Add generics here     --
9         -- NAME : TYPE := DEFAULT_VALUE (separated by , ) --
10        SAMPLE_PERIOD : integer := 10000000
11
12     );
13
14     port (
15           -- Declare module ports here --
16        -- NAME : DIRECTION TYPE (separated by ; ) --
17
18        -- CLK input --
19        ADC_CLK_10  : in std_logic; -- 10 MHz
20        --MAX10_CLK1_50   : in std_logic; -- 50 MHz 1
21        --MAX10_CLK2_50   : in std_logic; -- 50 MHz 2
22
23        -- Button input --
24        KEY : in std_logic_vector (1 downto 0);
25
26        -- 7-Segment output --
27        HEX0 : out std_logic_vector(7 downto 0);
28        HEX1 : out std_logic_vector(7 downto 0);
29        HEX2 : out std_logic_vector(7 downto 0);
30        HEX3 : out std_logic_vector(7 downto 0);
31        HEX4 : out std_logic_vector(7 downto 0);
32        HEX5 : out std_logic_vector(7 downto 0);
33        HEX6 : out std_logic_vector(7 downto 0);
34
35        -- Arduino Header --
36        ARDUINO_IO        : inout std_logic_vector(15 downto 0);
37        ARDUINO_RESET_N   : inout std_logic
38
39     );
40
41  end entity adc;
42
```

Figure 1: ADC.vhd Pt 1

```
43  architecture behavioral of adc is
44
45      -- Components --
46      -- ADC --
47      component my_ADC is
48          port (
49              clock_clk              : in  std_logic                     := 'X';             -- clk
50              reset_sink_reset_n     : in  std_logic                     := 'X';             -- reset_n
51              adc_pll_clock_clk      : in  std_logic                     := 'X';             -- clk
52              adc_pll_locked_export  : in  std_logic                     := 'X';             -- export
53              command_valid          : in  std_logic                     := 'X';             -- valid
54              command_channel        : in  std_logic_vector(4 downto 0)  := (others => 'X'); -- channel
55              command_startofpacket  : in  std_logic                     := 'X';             -- startofpacket
56              command_endofpacket    : in  std_logic                     := 'X';             -- endofpacket
57              command_ready          : out std_logic;                                        -- ready
58              response_valid         : out std_logic;                                        -- valid
59              response_channel       : out std_logic_vector(4 downto 0);                     -- channel
60              response_data          : out unsigned(11 downto 0);                    -- data
61              response_startofpacket : out std_logic;                                        -- startofpacket
62              response_endofpacket   : out std_logic                                         -- endofpacket
63          );
64      end component my_ADC;
65
66      -- PLL --
67      component my_PLL IS
68          PORT
69          (
70              areset   : IN STD_LOGIC  := '0';
71              inclk0   : IN STD_LOGIC  := '0';
72              c0       : OUT STD_LOGIC ;
73              locked   : OUT STD_LOGIC
74          );
75      END component;
76
```

Figure 2:ADC.vhd Pt 2

```
77      -- Declare internal signals here -- (terminated by ; )
78      -- signal NAME : TYPE ;
79
80      -- ADC Command Signals --
81      signal command_valid          : std_logic                     := '1';             -- valid
82      signal command_channel        : std_logic_vector(4 downto 0)  := "00001";         -- channel
83      signal command_startofpacket  : std_logic                     := '1';             -- startofpacket
84      signal command_endofpacket    : std_logic                     := '1';             -- endofpacket
85      signal command_ready          : std_logic;                                        -- ready
86
87      -- ADC Response Signals --
88      signal response_valid         : std_logic;                                        -- valid
89      signal response_channel       : std_logic_vector(4 downto 0);                     -- channel
90      signal response_data          : unsigned(11 downto 0);                -- data
91      signal response_startofpacket : std_logic;                                        -- startofpacket
92      signal response_endofpacket   : std_logic;                                        -- endofpacket
93
94      -- PLL Signals --
95      signal c0_sig                   :  std_logic                    := 'X';         -- clk
96      signal locked_sig               :  std_logic                    := 'X';         -- export
97
98
99      --7-segment display
100     type SEVEN_SEG is array (0 to 15) of std_logic_vector(7 downto 0); -- Define new type for lookup table
101     constant table : SEVEN_SEG := (
102                 X"C0", X"F9", X"A4", X"B0",  -- 0, 1, 2, 3
103                 X"99", X"92", X"82", X"F8",  -- 4, 5, 6, 7
104                 X"80", X"90", X"88", X"83",  -- 8, 9, A, B
105                 X"C6", X"A1", X"86", X"8E"); -- C, D, E, F
106
107     -- MISC --
108     signal sample_counter : integer := 0;
109     signal sample_trigger : std_logic;
110     signal display : unsigned(12 downto 0) := (others => '0');
111     signal next_display : unsigned(12 downto 0) := (others => '0');
112     signal temp_display : integer;
113
114
```

Figure 3: ADC.vhd Pt 3

```
115  begin
116
117     -- Instantiate IP Blocks --
118
119     -- ADC --
120     u0 : component my_ADC
121        port map (
122           -- Input
123           clock_clk              => ADC_CLK_10,                    --          clock.clk
124           reset_sink_reset_n     => KEY(0),                        --     reset_sink.reset_n
125           adc_pll_clock_clk      => c0_sig,                        --  adc_pll_clock.clk
126           adc_pll_locked_export  => locked_sig,                    -- adc_pll_locked.export
127           command_valid          => command_valid,                 --        command.valid
128           command_channel        => command_channel,               --                .channel
129           command_startofpacket  => command_startofpacket,         --                .startofpacket
130           command_endofpacket    => command_endofpacket,           --                .endofpacket
131           -- Output
132           command_ready          => command_ready,                 --                .ready
133           response_valid         => response_valid,                --       response.valid
134           response_channel       => response_channel,              --                .channel
135           response_data          => response_data,                 --                .data
136           response_startofpacket => response_startofpacket,        --                .startofpacket
137           response_endofpacket   => response_endofpacket           --                .endofpacket
138        );
139
140     -- PLL --
141     my_PLL_inst : my_PLL PORT MAP (
142        areset    => NOT KEY(0),
143        inclk0    => ADC_CLK_10,
144        c0        => c0_sig,
145        locked    => locked_sig
146     );
147
```

Figure 4: ADC.vhd Pt 4

```
148     -- Timing Controller --
149     process (ADC_CLK_10)
150     begin
151        if rising_edge(ADC_CLK_10) then
152           if sample_counter < SAMPLE_PERIOD - 1 then
153              sample_counter <= sample_counter + 1;
154              sample_trigger <= '0';
155           else
156              sample_counter <= 0;
157              sample_trigger <= '1';
158           end if;
159        end if;
160     end process;
161
162     -- Sampling controller --
163     process (ADC_CLK_10)
164     begin
165        if rising_edge(ADC_CLK_10) then
166           if (response_valid = '1') then
167              temp_display <= to_integer(response_data) * 2 * 2500 / 4094;
168              display <= to_unsigned(temp_display, display'length);
169           end if;
170        end if;
171     end process;
172
```

Figure 5: ADC.vhd Pt 5

```
173     --process to drive 7 segment
174     process (ADC_CLK_10)
175     begin
176        if rising_edge(ADC_CLK_10) then
177           if (sample_trigger = '1') then
178              HEX0 <= table(to_integer(display(3 downto 0)));
179              HEX1 <= table(to_integer(display(7 downto 4)));
180              HEX2 <= table(to_integer(display(11 downto 8)));
181              HEX3 <= X"FF";
182              HEX4 <= X"FF";
183              HEX5 <= X"FF";
184           elsif KEY(0) = '0' then
185              HEX0 <= table(0);
186              HEX1 <= table(0);
187              HEX2 <= table(0);
188              HEX3 <= X"FF";
189              HEX4 <= X"FF";
190              HEX5 <= X"FF";
191           end if;
192        end if;
193     end process;
194
195  end architecture behavioral;
196
```

Figure 6: VGA.vhd Pt 6