Experiment 8: ADC and DAC

Aaron John Sabu: 170070050

May 2, 2019

1 Overview

1. This experiment implements the conversion of an analog signal to a digital signal and vice versa by passing the signal through the ADC for the former and then a DAC for the latter.

2. Processing, such as low-pass filtering, is performed in the Krypton board.

2 Codes

2.0.1 ADC

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ADC is
  port (CLK: in std_logic; r: in std_logic;
    DATA: in std_logic_vector(7 downto 0);
    CS,RD,WR: out std_logic; interrupt: in std_logic;
    DAC: out std_logic_vector(7 downto 0));
end entity;
architecture Behave of ADC is
    type FsmState is (IDLE, INIT, WRITE, WRITE_DONE, READ, END);
    signal fsm_state: FsmState;
    signal R: std_logic_vector(7 downto 0);
    signal Count1: std_logic_vector(2 downto 0);
```

```
    signal Count2: std_logic_vector(3 downto 0);
    signal Count3: std_logic_vector(2 downto 0);
    signal Timer: std_logic_vector(15 downto 0);
begin
  process(CLK, R, fsm_state, Count1, Count2, Count3)
    variable Count1_var: std_logic_vector(2 downto 0);
    variable next_fsm_state_var: FsmState;
    variable Count2_var: std_logic_vector(3 downto 0);
    variable Count3_var: std_logic_vector(2 downto 0);
    variable CS_var: std_logic;
    variable R_var: std_logic_vector(7 downto 0);
    variable WR_var: std_logic;
    variable RD_var: std_logic;
    variable Timer_var: std_logic_vector(15 downto 0);
    begin
      -- default values.
      count1_var := Count1;
      count2_var := Count2;
      count3_var := Count3;
      next_fsm_state_var := fsm_state;
      case fsm_state is
        when IDLE =>
          CS_var := '1';
          WR_var := '1';
          RD_var := '1';
          next_fsm_state_var := INIT;
          Timer_var := "0000000000000001";
        when INIT =>
          CS_var := '0';
          WR_var := '1';
          RD_var := '1';
          count1_var := "000";
          next_fsm_state_var := WRITE;
          Timer_var := std_logic_vector(unsigned(Timer) + 1);
        when WRITE =>
          CS_var := '0';
          WR_var := '0';
          RD_var := '1';
```

```
  Timer_var := std_logic_vector(unsigned(Timer) + 1);
  if(Count1 = "100") then
  next_fsm_state_var := WRITE_DONE;
    Count2_var := "0000";
  else
    next_fsm_state_var := WRITE;
    Count1_var := std_logic_vector(unsigned(Count1) + 1);
  end if;
when WRITE_DONE =>
  CS_var := '0';
  WR_var := '1';
  RD_var := '1';
  Timer_var := std_logic_vector(unsigned(Timer) + 1);
  if(Count2 = "1000")then
    if(interrupt = '0') then
      next_fsm_state_var := READ;
      count3_var := "000";
    else
      next_fsm_state_var := WRITE_DONE;
    end if;
  else
    next_fsm_state_var := WRITE_DONE;
    Count2_var := std_logic_vector(unsigned(Count2) + 1);
  end if;
when READ =>
  CS_var := '0';
  WR_var := '1';
  RD_var := '0';
  Timer_var := std_logic_vector(unsigned(Timer) + 1);
  if(Count3 = "110") then
    next_fsm_state_var := END;
    R_var :=  DATA;
  else
    next_fsm_state_var := READ;
    Count3_var := std_logic_vector(unsigned(Count3) + 1);
  end if;
when END =>
  CS_var := '0';
```

```
          WR_var := '1';
          RD_var := '1';
          Timer_var := std_logic_vector(unsigned(Timer) + 1);
          if(Timer_var = "1100001101010000")then
            next_fsm_state_var:= IDLE;
            DAC <= R_var;
          end if;
    end case;
    if(CLK'event and CLK='1') then
      if(r = '1') then
        fsm_state <= IDLE;
      else
        fsm_state <= next_fsm_state_var;
        count1 <= count1_var;
        count2 <= count2_var;
        count3 <= count3_var;
        R <= R_var;
        WR <= WR_var;
        CS <= CS_var;
        RD <= RD_var;
        Timer <= Timer_var;
      end if;
    end if;
  end process;
end Behave;
```

2.0.2 Filter

```
library ieee;
use ieee.std_logic_1164.all;
entity Filter is
  port(CLK, RST: in std_logic; DAC: in std_logic_vector(7 downto 0),
    output: out std_logic_vector(7 downto 0));
end entity Filter;

architecture Struct of Filter is
  signal A, B, C, D, E, F, G, H: std_logic_vector(7 downto 0)
begin
```

```
  variable sum: std_logic_vector(10 downto 0);
  begin process(A, B, C, D, E, F, G, H)
    variable a, b, c, d, e, f, g, h: std_logic_vector(7 downto 0) := "00000000";
    variable a1, b1, c1, d1, e1, f1, g1, h1: std_logic_vector(10 downto 0);
    h := G;  h1 := "000" & h;
    g := F;  g1 := "000" & g;
    f := E;  f1 := "000" & f;
    e := D;  e1 := "000" & e;
    d := C;  d1 := "000" & d;
    c := B;  c1 := "000" & c;
    b := A;  b1 := "000" & b;
    a := DAC;  a1 := "000" & a;
    sum := (std_logic)((unsigned(a1)+unsigned(b1)+unsigned(c1)
      +unsigned(d1)+unsigned(e1)+unsigned(f1)+unsigned(g1)+unsigned(h1))/8);
    if(CLK'event and CLK='1') then
      if(RST = '1') then
        output <= "00000000";
      else
        output <= sum(7 downto 0);
      end if;
    end if;
  end process;
end Struct;
```

2.0.3 Counter50

```
library ieee;
use ieee.std_logic_1164.all;
entity Counter50 is
  port(CLK, RST: in std_logic; output: out std_logic_vector(7 downto 0));
end entity Counter50;
architecture Struct of Counter50 is
  signal Count1: std_logic_vector(15 downto 0);
begin
  process(CLK, R, fsm_state)
  begin
    variable Count1_var: std_logic_vector(15 downto 0);
    variable next_fsm_state_var: FsmState;
```

```
    begin
      -- default values.
      Count1_var := Count1;
      next_fsm_state_var := fsm_state;
      case fsm_state is
        when OFFSTATE =>
          output := '1';
          if(Count1 = "1100001101010000") then
            next_fsm_state_var := ONSTATE;
          else
            next_fsm_state_var := OFFSTATE;
            Count1_var := std_logic_vector(unsigned(Count1) + 1);
          end if;
        when ONSTATE =>
          output := '1';
          next_fsm_state_var := OFFSTATE;
    end case;
    if(CLK'event and CLK='1') then
      if(R = '1') then
        fsm_state <= OFFSTATE;
      else
        fsm_state <= next_fsm_state_var;
        Count1 <= Count1_var;
      end if;
    end if;
  end process;
end Struct;
```

2.0.4 Main Code

```
library ieee;
use ieee.std_logic_1164.all;
entity Main is
  port(CLK: in std_logic; r: in std_logic;
      interrupt: in std_logic; DATA: in std_logic_vector(7 downto 0);
      DAC: out std_logic_vector(7 downto 0); CS, RD, WR: out std_logic);
end entity Main;
architecture Struct of Main is
```

```
  component Filter is
    port(CLK, RST: in std_logic; DAC: in std_logic_vector(7 downto 0),
  output: out std_logic_vector(7 downto 0));
  end component Filter;
  component ADC is
    port (CLK: in std_logic; r: in std_logic;
      interrupt: in std_logic; DATA: in std_logic_vector(7 downto 0);
      DAC: out std_logic_vector(7 downto 0); CS, RD, WR: out std_logic);
  end component;
  component Counter50 is
    port(CLK, RST: in std_logic; output: out std_logic_vector(7 downto 0));
  end component Counter50;
  signal CLK2: std_logic; signal temp: std_logic_vector(7 downto 0);
begin
  cnt: Counter50 port map(CLK => CLK, RST => RST, output => CLK2);
  adc1: ADC port map(CLK => CLK, r => RST, interrupt => interrupt,
    DATA => DATA, DAC => temp, CS => CS, RD => RD, WR => WR);
  flt1: Filter port map(CLK => CLK2, RST => r, DAC => temp, output => DAC);
end Struct;
```

2.0.5 DUT

```
library ieee;
use ieee.std_logic_1164.all;
entity DUT is
  port(input_vector: in std_logic_vector(10 downto 0);
    output_vector: out std_logic_vector(10 downto 0));
end entity DUT;
architecture DutWrap of DUT is
  component Main is
    port (CLK: in std_logic; r: in std_logic;
      interrupt: in std_logic; DATA: in std_logic_vector(7 downto 0);
      DAC: out std_logic_vector(7 downto 0); CS, RD, WR: out std_logic);
  end component;
begin
  Main_1: Main port map(
    CLK => input_vector(9),
    interrupt => input_vector(8),
```

```
    DATA => input_vector(7 downto 0),
    r => input_vector(10),
    CS => output_vector(10),
    RD => output_vector(9),
    WR => output_vector(8),
    DAC => output_vector(7 downto 0)
);
end DutWrap;
```

3 Design

The state machine implementing the receipt and transmission of an analog signal via the ADC consisted of the following states:

- **IDLE** - The machine does not receive and the ADC is not enabled.
- **INIT** - The ADC is enabled in this state.
- **WRITE** - The *WRITE* pin of the ADC is set to low, hence making the converter receive the analog signal. In order to provide enough time for transmission, the counter keeps the machine in the same state for a few cycles.
- **WRITE_DONE** - This state ensures as an intermediary between the write state and the read state for the circuit to settle.
- **READ** - The *READ* pin of the ADC is set to low, hence making the converter send the digitized analog signal. In order to provide enough time for transmission, the counter keeps the machine in the same state for a few cycles.
- **END** - This state concludes the complete analog-to-digital-conversion process.

The Filter section (Part b of the experiment) consists of a filter, a counter of 50000 counts and a main code connecting the three machines. The counter acts as the 1kHz clock generator from the 50MHz clock in the board.
This clock signal is used in the filter code, at the edge of whose signal, eight numbers (from the last eight inputs) are averaged and the output is given as the output.
The averaging implies the low-pass filtering of the parallelly operated signals.