Experiment No. 12
Successive Approximation A/D Converter
ECE 446

Peter Chinetti

December 4, 2014

Instructor: Professor Shanechi

1 Introduction

When working with real world signals, it is useful to be able to sample voltage levels. In this lab, the design of Analog to Digital Converters is explored and implemented.

The translation is done with a DAC and a comparator. The sample voltage is held to one end of the comparator, and the DAC is tied into the other side. The DAC's voltage is stepped up from 0 until the comparator transitions, at which point the DAC voltage level is recorded as the output voltage level.

The algorithm used to determine the voltage has two problems: it is slow, in the worst case taking as many cycles to complete as there are DAC voltage levels, and it takes different amounts of time to encode different voltage levels. These problems can be solved by changing how the voltage on the DAC is stepped.

2 Pre-Lab Questions

2.1

The input range is also 0 to 3.3 volts, as higher voltages can not be generated to feed into the comparator.

2.2

The DAC output could be DC biased down 1.65 volts to move it's 0 volt output into the negative voltage range.

2.3

-1.65 to 1.65 volts.

2.4

If the voltage is increased to a value higher than 3.3 volts, but lower than the Op-Amp maximum voltage, the ADC will read as if the input was 3.3 volts. If the voltage is increased past the Op-amps maximum voltage, it will release the magic smoke and go into an undefined state.

3 Procedure

a. Build resistor net and comparator circuit

b. Write VHDL to implement ADC

c. Assign pins to ports

d. Simulate

e. Program and Test

4 Equipment

- PC
- Spartan-3E development board
- Op-Amp
- Assorted resistors and diodes
- Breadboard

5 Code

5.1 Top-Level Module

```
1 library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
3
5 entity adc is
    Port ( clk : in  STD_LOGIC;
7          da_out, result_out : out  STD_LOGIC_VECTOR (3 downto 0);
           done : out  STD_LOGIC;
9          start_in : in  STD_LOGIC;
           comp : in  STD_LOGIC);
```

```
11 end adc;
13 architecture Behavioral of adc is
     component Counter is
15   Port ( clk_in, en : in  STD_LOGIC;
              o : out STD_LOGIC_VECTOR(3 downto 0) :="0000";
17          tc : out STD_LOGIC := '0';
              reset : in  STD_LOGIC;
19          clk_out : out STD_LOGIC);
     end component;
21
     component debounce is
23   Port ( input : in std_logic;
           clk_in : in std_logic;
25         output : out std_logic);
     end component;
27
     signal r, cnt_clk, TC, c_en, start : STD_LOGIC;
29   signal result : STD_LOGIC_VECTOR (3 downto 0);
     type state is (IDLE, COUNT);
31   signal f, p : state;
   begin
33
     da_out <= result;
35   result_out <= result;

37   cnt_0 : Counter
     port map(
39     clk_in => clk,
       en => c_en,
41     o => result,
       tc => TC,
43     reset => r,
       clk_out => cnt_clk);
45
     deb_0 : debounce
47   port map(
       input => start_in,
49     clk_in => clk,
       output => start);
51

53 process (p, start, comp, cnt_clk, TC)
   begin
55   if rising_edge(cnt_clk) then
       p <= f;
57   end if;

59   case p is
       when IDLE =>
61       if start = '1' then
           r <= '1';
63         f <= COUNT;
           c_en <= '1';
65         done <= '0';
         else
67         r <= '0';
```

```
          f <= IDLE;
          c_en <= '0';
          done <= '1';
        end if;
      when COUNT =>
        if comp = '1' or TC = '1' then
          done <= '1';
          f <= IDLE;
          c_en <= '0';
        else
          f <= count;
        end if;
    end case;

end process;

end Behavioral;
```

adc.vhd

5.2 Counter Module

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

entity Counter is
    Port ( clk_in, en_in : in  STD_LOGIC;
           o, result : out STD_LOGIC_VECTOR(3 downto 0) :="0000";
        tc : out STD_LOGIC := '0';
           reset : in  STD_LOGIC := '0';
        done : out STD_LOGIC);
end Counter;

architecture Behavioral of Counter is
  signal counter: unsigned(3 downto 0) :="0000";
  signal cnt_clk, clk_i, en: STD_LOGIC;

  component selectable_clock is
  Port ( clk : in std_logic;
         s0 : in std_logic;
         s1 : in std_logic;
         out_clk : out std_logic);
  end component;

  component debounce is
  Port ( input : in std_logic;
    clk : in std_logic;
    output : out std_logic);
  end component;
begin

  done <= not en_in;

  clk_div : selectable_clock
```

```
  port map(
    clk => clk_in,
    s0 => '1',
    s1 => '1',
    out_clk => clk_i);

  deb : debounce
  port map(
    input => reset,
    clk => clk_in,
    output => en);

--  cnt_clk <= clk_i and en;

  o <= std_logic_vector(counter);
  result <= std_logic_vector(counter);

  process(clk_i)
  begin
    if rising_edge(clk_i) then
      if reset = '1' then
        counter <= "0000";
      elsif en_in = '1' and counter <= "1111" then
        counter <= counter + "0001";
      end if;
    end if;
  end process;

  process(counter)
  begin
    if counter = "1111" then
      tc <= '1';
    else
      tc <= '0';
    end if;
  end process;
end Behavioral;
```

Counter.vhd

5.3 Debouncer Module

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


entity debounce is
  Port ( input : in std_logic;
    clk : in std_logic;
    output : out std_logic);
end debounce;

architecture Behavioral of debounce is
begin
  process (clk, input) -- Start a process.
```

```
  variable count : integer := 0; -- Variable declaration.
  begin
    if clk = '1' and clk'event then -- Rising edge detection.
      if input = '1' then -- Input is high at clock.
        count := count + 1; -- Increment count.
      else -- Input is low at clock.
        count := 0; -- Reset count.
      end if;

      if count > 1000000 then -- Input high long enough to
                    -- output.
        output <= '1'; -- Output high.
      else -- Input not high long enough.
        output <= '0'; -- Output low.
      end if;
    end if;
  end process;
end Behavioral;
```

debounce.vhd

5.4 Clock Divider Module

```
-- Selectable output frequency clock divider code.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity selectable_clock is
  Port ( clk : in std_logic;
         s0 : in std_logic;
         s1 : in std_logic;
         out_clk : out std_logic);
end selectable_clock;
--If s1 and s0 are both low, the output clock rate is 1/10 Hz.
--If s1 is low and s0 is high, the output clock rate is 1 Hz.
--If s1 is high and s0 is low, the output clock rate is 10 Hz.
--If s1 and s0 are both high, the output clock rate is 1 KHz.
architecture Behavioral of selectable_clock is
begin
  process (clk, s0, s1)
    variable count : integer := 0;
  begin
    if clk = '1' and clk'event then
      count := count + 1;
      -- Start a process.
      -- Variable declaration.
      -- Rising edge detection.
      -- Code to create the 1/10 Hz clock.
      if s0 = '0' and s1 = '0' then
        if count >= 500000000 then
          -- Taken off a 50MHz clock.
          count := 0;
        -- Reset count for next cycle.
        end if;
```

```
        if count >= 0 and count <= 250000000 then
          out_clk <= '1';
        -- High portion of 1/10 HZ clock.
        else
          out_clk <= '0';
        -- Low portion of 1/10 HZ clock.
        end if;
      end if;
      -- Code to create the 1 Hz clock.
      if s0 = '1' and s1 = '0' then
        if count >= 50000000 then
          -- Taken off a 50MHz clock.
          count := 0;
        -- Reset count for next cycle.
        end if;
        if count >= 0 and count <= 25000000 then
          out_clk <= '1';
        -- High portion of 1 HZ clock.
        else
          out_clk <= '0';
        -- Low portion of 1 HZ clock.
        end if;
      end if;
      -- Code to create the 10 Hz clock.
      if s0 = '0' and s1 = '1' then
        if count >= 5000000 then
          -- Taken off a 50MHz clock.
          count := 0;
        -- Reset count for next cycle.
        end if;
        if count >= 0 and count <= 2500000 then
          out_clk <= '1';
        -- High portion of 10 HZ clock.
        else
          out_clk <= '0';
        -- Low portion of 10 HZ clock.
        end if;
      end if;
      -- Code to create the 1 KHz clock.
      if s0 = '1' and s1 = '1' then
        if count >= 50000 then
          -- Taken off a 50MHz clock.
          count := 0;
        -- Reset count for next cycle.
        end if;
        if count >= 0 and count <= 25000 then
          out_clk <= '1';
        -- High portion of 1 KHz clock.
        else
          out_clk <= '0';
        -- Low portion of 1 KHz clock.
        end if;
      end if;
    end if;
  end process;
end Behavioral;
```

clk_div.vhd

6 Conclusions

The purpose of this lab was achieved. An ADC was built and tested. Operation was verified through simulation and physical implementation.