EE 214: Digital Circuits Lab
Dev Arora (23B1271)
Arjun Singh (23B1272)

ADC to DAC Using SPI

Digital-to-Analog Conversion and Verification via SPI Interface with DAC and Multimeter

Task 2B

November 4, 2024

Work Distribution:

Dev & Arjun:

Both of us worked on the code together and made the report together.

Explanation:

Retrieving the Digital Value

- The **reg_a** register on the FPGA holds a 10-bit digital value representing the analog input voltage that was sampled and converted by the ADC in Task 2A.
- This value acts as the input to the DAC, which will convert it back to an analog signal.

Configuring the SPI Master for DAC Communication:

- The SPI protocol is utilized to transfer the digital data from the FPGA to the DAC, ensuring precise and synchronized data transmission.
- The CS(Chip Select) signal is initially set low to begin the communication sequence. The SPI master then transmits the 10-bit digital value stored in reg_a over the MOSI(Master Out Slave In) line, while the SCLK(Serial Clock) signal controls the timing of each bit sent to DAC.
- The DAC operates in SPI Mode 0, as specified, ensuring compatibility with the SPI master setup and correct timing synchronization.

Digital-to-Analog Conversion in the DAC:

- The DAC receives the digital value from the FPGA and converts it to an analog voltage output.
- A reference voltage (Vref) of 3.3V is applied to the DAC, enabling accurate scaling of the digital-to-analog conversion so that the output voltage matches the original analog input.

Verification Using Multimeter:

- The DAC's analog output is measured with a multimeter to ensure it matches the original input voltage supplied to the ADC.
- This measurement shows that the whole SPI communication path—from the ADC to the FPGA to the DAC—kept the signal's quality and converted it back to its original analog form.

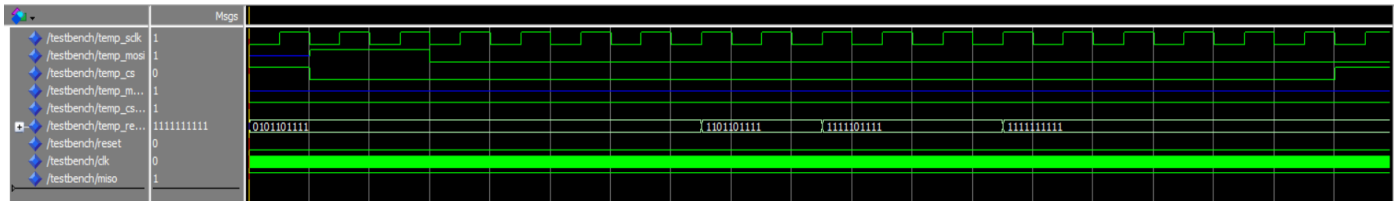
Implementation in VHDL:

- The SPI master code is updated to send data specifically for the DAC interface, including setting the right clock and CS signals needed by the DAC.
- Separate reset signals are configured for the DAC to ensure reliable operation without interference from other modules in the system.

Pin Planning:

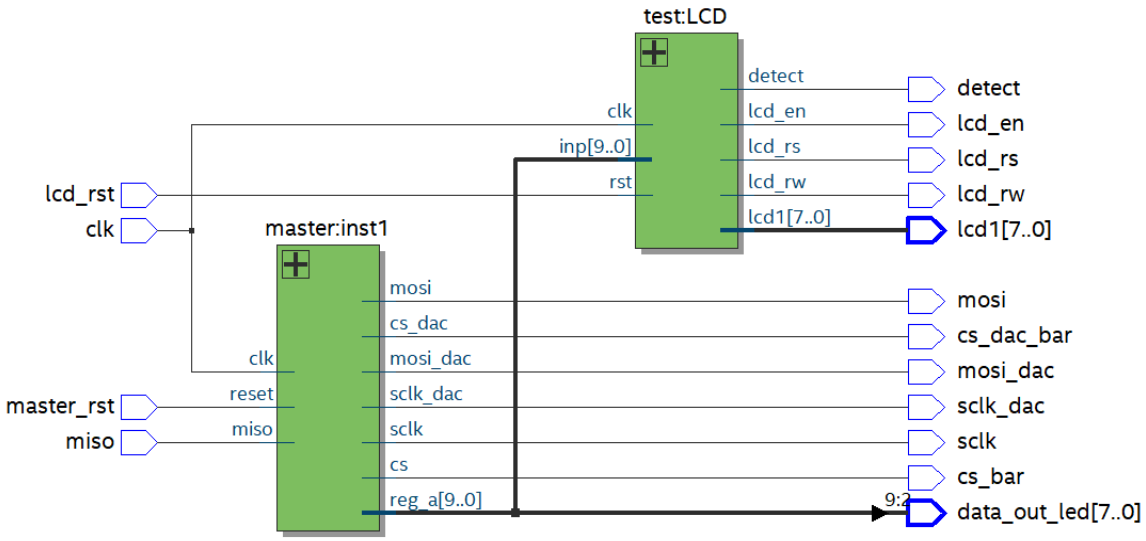
Results and Observation:

Waveform:

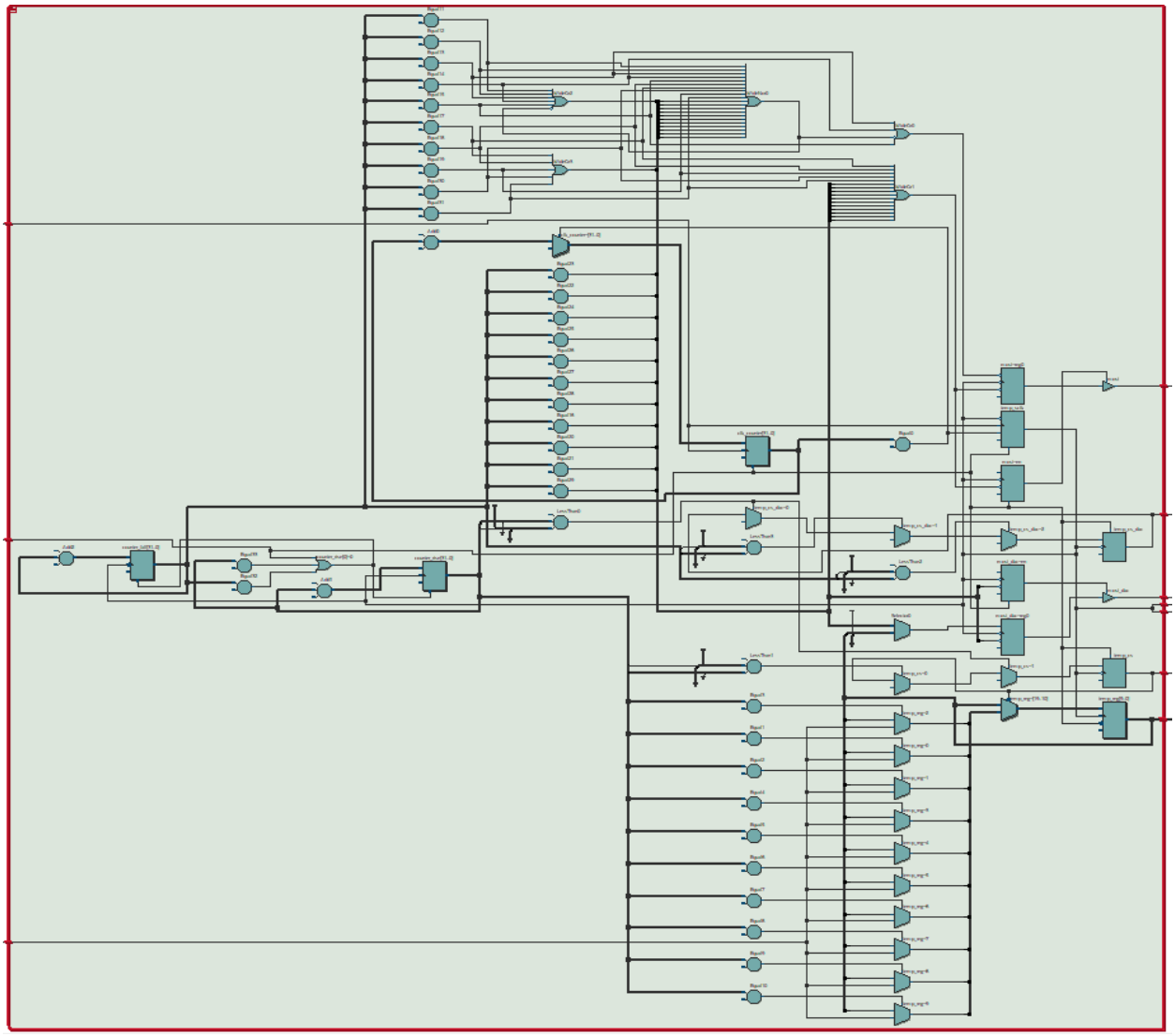


Netlist Viewer:

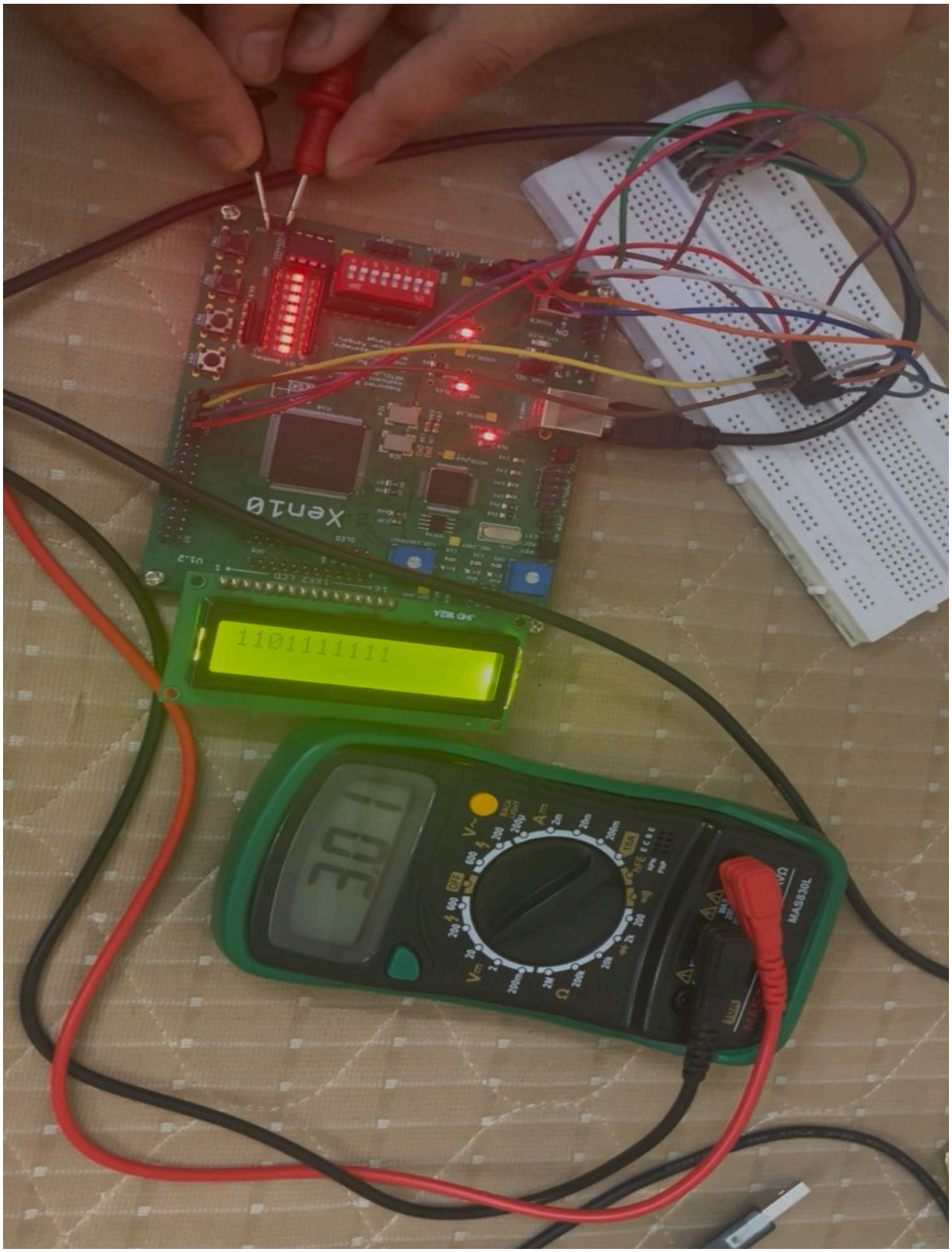
Overall Netlist Viewer:

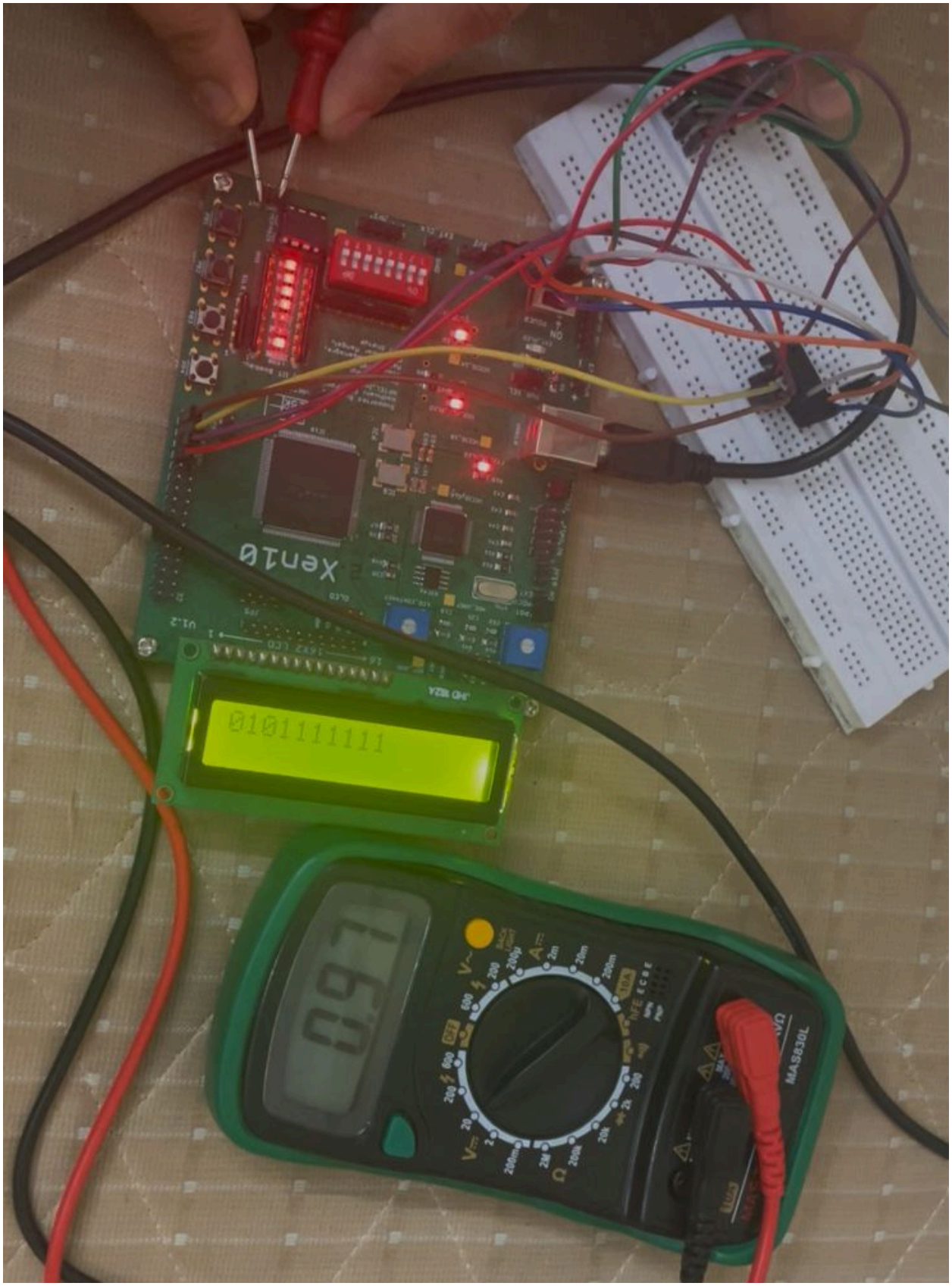


Master Netlist Viewer:



LCD output:




Xen10
0101111111
MAS830L

Regeneration of Sine Wave using ADC- DAC

Task 3(Bonus Task)

November 5, 2024

Explanation:

In this setup, a continuous loop was added to the code for "2B," allowing it to keep reading live data from the ADC in real-time. This loop lets the system continuously capture and process the changing analog input signals, which are provided by an Arbitrary Function Generator (AFG).

The process works as follows:

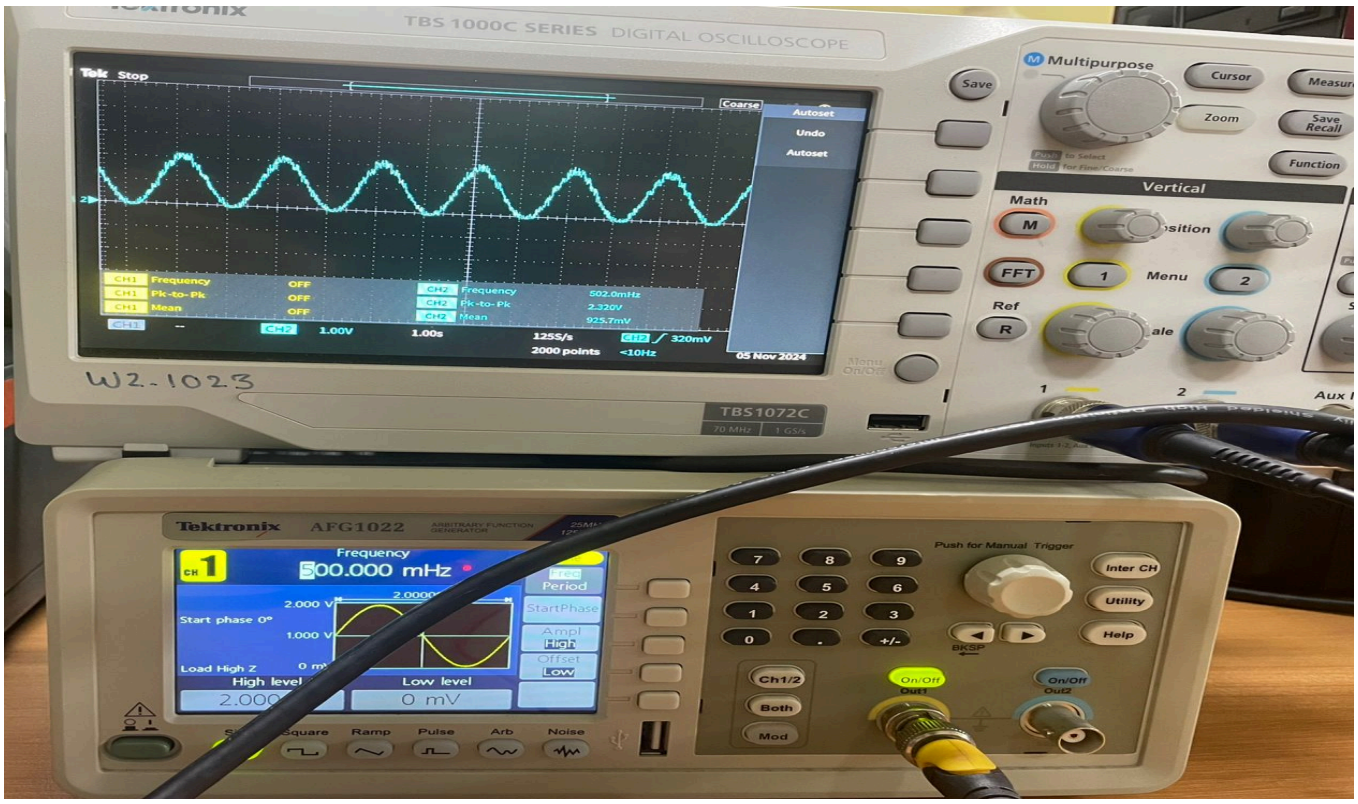
1. **Analog Signal Input:** The AFG supplies a continuous analog signal to the ADC, acting as the source of the changing data.

2. **Continuous ADC Reading:** The code is designed with a loop that repeatedly samples and converts this analog input into digital form using the ADC. This setup ensures that each new reading happens without delay, making real-time data collection possible.

3. **Data Transfer to DAC:** After the analog signal is digitized by the ADC, this digital data is sent directly to the DAC through the SPI protocol. The DAC then converts this data back into an analog signal.

4. **Output Observation:** The analog output from the DAC can then be observed on an oscilloscope or another display tool to view the signal changes over time.

Output Observation on DSO:


TBS 1000C SERIES DIGITAL OSCILLOSCOPE
Tek Stop
Coarse
Autoset
Undo
Autoset
CH1 Frequency OFF
CH1 Pk-to-Pk OFF
CH1 Mean OFF
CH2 Frequency 502.0mHz
CH2 Pk-to-Pk 2.320V
CH2 Mean 925.7mV
CH1 -- CH2 1.00V 1.00s 125S/s 2000 points CH2 320mV <10Hz 05 Nov 2024
W2-1023
TBS1072C
70 MHz 1 GS/s
Save
Multipurpose
Cursor
Zoom
Save Recall
Function
Vertical
Math
M
FFT
1
Menu
2
Ref
R
Tektronix AFG1022
Frequency
500.000 mHz
Period
StartPhase
Ampl
High
Offset
Low
Start phase 0°
Load High Z
High level
Low level
0 mV
Sine
Square
Ramp
Pulse
Arb
Noise
Push for Manual Trigger
Inter CH
Utility
Help
BKSP
Ch1/2
Both
Mod
On/Off
Out1
On/Off
Out2

VHDL Code for Master Component

Digital Systems Lab

November 5, 2024

Master VHDL Code

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

entity master is
    port(
        clk : in std_logic;
        reset : in std_logic;
        miso : in std_logic;
        mosi : out std_logic;
        reg_a : out std_logic_vector(9 downto 0)
            :="1101000111";
        sclk : out std_logic;
        cs : out std_logic;

        mosi_dac : out std_logic;
        cs_dac : out std_logic;
        sclk_dac : out std_logic
    );
end entity;

architecture archi_master of master is

signal temp_sclk, temp_cs, temp_cs_dac : std_logic;
constant divider_value : integer := 25;
signal clk_counter : integer := 0;
signal adc_initial : std_logic_vector(4 downto 0) :="11000";
signal counter_rise, counter_fall, counter_dac : integer :=
    0;
signal temp_reg : std_logic_vector(9 downto 0)
    :="0101101111";

begin

clk_process : process(clk, reset)
begin
```

```
33      if reset='1' then
34          clk_counter <= 0;
35          temp_sclk <='0';
36      elsif rising_edge(clk) then
37          if clk_counter = divider_value - 1 then
38              temp_sclk <= NOT temp_sclk;
39              clk_counter <= 0;
40          else
41              clk_counter <= clk_counter + 1;
42          end if;
43      end if;
44  end process;
45
46  communication : process(temp_sclk, reset)
47  begin
48      if reset ='1' then
49          counter_rise <= 0;
50          counter_fall <= 0;
51          temp_cs <= '1';
52          temp_cs_dac <= '1';
53          mosi <= 'Z';
54          mosi_dac <= 'Z';
55      else
56          if rising_edge(temp_sclk) then
57              counter_rise <= counter_rise + 1;
58
59              if temp_cs = '0' then
60                  case counter_rise is
61                      when 7 =>  temp_reg(9) <= miso;
62                      when 8 =>  temp_reg(8) <= miso;
63                      when 9 =>  temp_reg(7) <= miso;
64                      when 10 => temp_reg(6) <= miso;
65                      when 11 => temp_reg(5) <= miso;
66                      when 12 => temp_reg(4) <= miso;
67                      when 13 => temp_reg(3) <= miso;
68                      when 14 => temp_reg(2) <= miso;
69                      when 15 => temp_reg(1) <= miso;
70                      when 16 => temp_reg(0) <= miso;
71                      when others => null;
72                  end case;
73              end if;
74          elsif falling_edge(temp_sclk) then
75              counter_fall <= counter_fall + 1;
76              if counter_rise < 17 then temp_cs <= '0';
                    temp_cs_dac <= '1';
77              elsif counter_rise > 17 then temp_cs <= '1';
                    end if;
78
79              if counter_fall > 17 then temp_cs_dac <= '0';
```

```
80              elsif counter_fall > 33 then temp_cs_dac <=
                    '1'; end if;
81
82              case counter_fall is
83                  when 0 => mosi <= adc_initial(4);
84                  when 1 => mosi <= adc_initial(3);
85                  when 2 => mosi <= adc_initial(2);
86                  when 3 => mosi <= adc_initial(1);
87                  when 4 => mosi <= adc_initial(0);
88                  when 18 => mosi_dac <= '0';
89                  when 19 => mosi_dac <= '1';
90                  when 20 => mosi_dac <= '1';
91                  when 21 => mosi_dac <= '1';
92                  when 22 => mosi_dac <= temp_reg(9);
93                  when 23 => mosi_dac <= temp_reg(8);
94                  when 24 => mosi_dac <= temp_reg(7);
95                  when 25 => mosi_dac <= temp_reg(6);
96                  when 26 => mosi_dac <= temp_reg(5);
97                  when 27 => mosi_dac <= temp_reg(4);
98                  when 28 => mosi_dac <= temp_reg(3);
99                  when 29 => mosi_dac <= temp_reg(2);
100                 when 30 => mosi_dac <= temp_reg(1);
101                 when 31 => mosi_dac <= temp_reg(0);
102                 when 32 => mosi_dac <= '1';
103                 when 33 => mosi_dac <= '1';
104                 when others => mosi <= '0';
105             end case;
106         end if;
107
108         if counter_fall = 34 or counter_rise = 34 then
109             counter_rise <= 0;
110             counter_fall <= 0;
111         end if;
112     end if;
113 end process;
114
115 cs <= temp_cs;
116 sclk <= temp_sclk;
117 cs_dac <= temp_cs_dac;
118 sclk_dac <= temp_sclk;
119 reg_a <= temp_reg;
120
121 end architecture;
```

VHDL Code for Top Level Design

Digital Systems Lab

November 5, 2024

Top Level VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity top_level is
    port (
        clk                     : in std_logic;
        miso                    : in std_logic;
        lcd_rst                 : in std_logic;
        master_rst              : in std_logic;
        cs_bar                  : out std_logic := '1';
        sclk                    : out std_logic := '0';
        mosi                    : out std_logic := 'Z';

        mosi_dac                : out std_logic := 'Z';
        cs_dac_bar              : out std_logic := '1';
        sclk_dac                : out std_logic := '0';

        lcd_rw                  : out std_logic;
        lcd_en                  : out std_logic;
        lcd_rs                  : out std_logic;
        lcd1                    : out std_logic_vector(7
            downto 0);
        detect                  : out std_logic;
        data_out_led            : out std_logic_vector(7
            downto 0)
    );
end entity;

architecture archi_top_level of top_level is

    component master is
        port(
            clk          : in std_logic;
```

```
34              reset         : in std_logic;
35              miso          : in std_logic;
36              mosi          : out std_logic;
37              reg_a         : out std_logic_vector(9 downto 0)
                    := "ZZZZZZZZZZ";
38              sclk          : out std_logic;
39              cs            : out std_logic;
40              mosi_dac      : out std_logic;
41              cs_dac        : out std_logic;
42              sclk_dac      : out std_logic
43          );
44      end component;
45
46      component test is
47          port(
48              inp           : in std_logic_vector(9 downto 0);
49              clk           : in std_logic;
50              rst           : in std_logic;
51              lcd_rw        : out std_logic;        --read &
                    write control
52              lcd_en        : out std_logic;        --enable
                    control
53              lcd_rs        : out std_logic;        --data or
                    command control
54              lcd1          : out std_logic_vector(7 downto
                    0);  --see pin planning in krypton manual
55              detect        : out std_logic
56          );
57      end component;
58
59      signal data_out_master : std_logic_vector(9 downto 0);
60      signal clk_counter     : integer := 0;
61
62  begin
63      data_out_led <= data_out_master(9 downto 2);
64      inst1: master port map (
65          clk       => clk,
66          miso      => miso,
67          reset     => master_rst,
68          cs        => cs_bar,
69          sclk      => sclk,
70          mosi      => mosi,
71          reg_a     => data_out_master,
72          mosi_dac  => mosi_dac,
73          sclk_dac  => sclk_dac,
74          cs_dac    => cs_dac_bar
75      );
76
77      LCD: test port map (
78          inp       => data_out_master,
```

```
79          clk       => clk,
80          rst       => lcd_rst,
81          lcd_rw    => lcd_rw,
82          lcd_en    => lcd_en,
83          lcd_rs    => lcd_rs,
84          lcd1      => lcd1,
85          detect    => detect
86      );
87  end architecture;
```