

Lab 1 report

**Serial Interface Module to an ADC and DAC**

**Lab objective**:

In this lab task,we should develop a module for communication with a digital-to analog converter (Burr Brown DAC8801) that shall be connected via a 3-wire

serial data interface to the FPGA. Before the data can be sent to the DAC, the

data has to be serialized. In order to test the design, 12 bit data words are read

from switches connected to the FPGA and sent from there to the DAC via the

serial bus. In a second step, the DAC module is connected to an ADC module

(ADS7947) and the entire signal processing chain ADC-DAC can be tested.

**Task description**:

What we basically had to do was at first to design the DAC001 module

and then integrate it inside the ADC\_DAC\_TOP module and test the whole design virtually with ModelSim software and also practically with FPGA.The following are the steps we made to reach the final result.

1.**Design the DAC001 module**:What we realized was that the structures of ADC and DAC modules are partially similar to each other except that FSM and shift register blocks are different and also there is no save register block in DAC module.

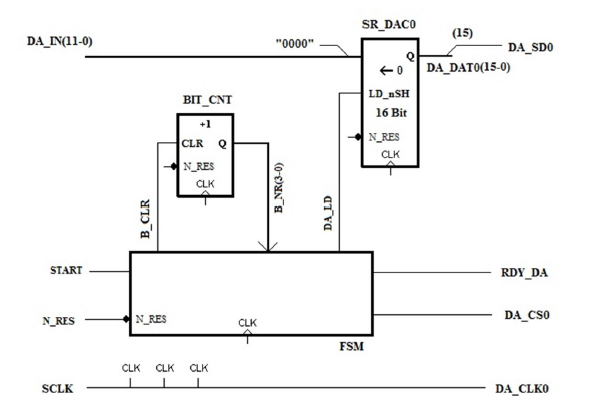


Figure 1: structure of the DAC module

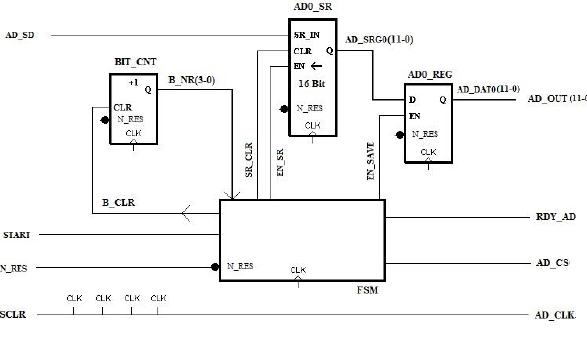


Figure 2: structure of the ADC module

.The FSM block has three Moore( DA\_CS0, DA\_LD,RDY\_DA) and one Mealy( B\_CLR) outputs.



Figure 3: FSM state diagram

Here is the VHDL code of the DAC001:

library IEEE;

use IEEE.std\_logic\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity DAC\_001 is

port(SCLK, N\_RES : in bit;

START : in bit; -- start signal

DA\_IN : in bit\_vector(11 downto 0) ;

RDY\_DA : out bit;

DA\_SD0 : out bit ;

DA\_CS0 : out bit ;

DA\_CLK0 : out bit );

end DAC\_001 ;

architecture BEHAVIOUR of DAC\_001 is

-- bit counter

signal B\_NR : unsigned(3 downto 0):= "0000" ;

signal B\_CLR : bit ;

--Control Signals DAC

signal DA\_LD : bit; --if load '1' means start shifting else load in DA\_IN

signal DA\_DATA0 : bit\_vector (15 downto 0) :=(others=> '0'); -- save all the shifted bits to DA\_DAT0 and then from DA\_DAT0 send always MSB to the DA\_SD0

--states of the DAC communication module

type State\_Type\_DA is (IDLE\_DA, LOAD\_DA);

signal DA\_STATE : State\_Type\_DA;

signal DA\_NEXTSTATE : State\_Type\_DA;

begin

-- clock to DAC

DA\_CLK0 <= not SCLK;

-- counter for serial bits -------------------------------

BIT\_CNT: process(SCLK)

begin

if SCLK = '1' and SCLK' event then

if B\_CLR = '1' then -- B\_CLR is clearing the counter

B\_NR <= "0000" after 5 ns ;

else

B\_NR <= B\_NR + 1 after 5 ns ; -- its output og bit counter

end if ;

end if ;

end process BIT\_CNT;

SR\_DAC0 :process (SCLK, N\_RES)

begin

if N\_RES ='0' then

DA\_DATA0 <=(others=>'0') after 5 ns;

elsif SCLK = '1' and SCLK' event then

if DA\_LD = '1' then

DA\_DATA0 <= "0000"&DA\_IN ;

elsif DA\_LD = '0' then

DA\_DATA0 <= DA\_DATA0(14 downto 0) & '0' ;

end if;

end if;

end process SR\_DAC0;

DA\_SD0 <= DA\_DATA0(15);

-- we have to create one(upper) FSM for whatever is related to clock and other FSM should contain all combinational logics

FSM\_DA\_REG : process(SCLK , N\_RES)

begin

if N\_RES ='0' then

DA\_STATE <= IDLE\_DA after 5 ns ;

elsif SCLK = '1' and SCLK ' event then

DA\_STATE <= DA\_NEXTSTATE after 5 ns ;

end if ;

end process ;

FSM\_DA\_SN : process(DA\_STATE,B\_NR,START) -- why don’t we combine both FSMs

begin

DA\_NEXTSTATE <= DA\_STATE after 5 ns ;-- idle state is our next state

DA\_CS0 <= '1' after 5 ns ; -- CS is HIGH ( see Timing Diagram ) in IDLE STATE

DA\_LD <= '0' after 5 ns ;

RDY\_DA <= '0' after 5 ns ;

B\_CLR <= '1' after 5 ns ;

case DA\_STATE is

when IDLE\_DA =>

RDY\_DA <= '1' after 5 ns ;

DA\_CS0 <= '1' after 5 ns ;

B\_CLR <= '1' after 5 ns ;

DA\_LD <='1' after 5 ns;

if START = '1' then

DA\_NEXTSTATE <= LOAD\_DA after 5 ns ;

end if ;

when LOAD\_DA =>

DA\_CS0 <= '0' after 5 ns;

B\_CLR <='0' after 5 ns;-- we don’t clear until the shifting is in process

DA\_LD <='0' after 5 ns;

if B\_NR ="1111" then

B\_CLR <='1' after 5 ns;

DA\_NEXTSTATE<= IDLE\_DA after 5 ns ;

end if ;

end case;

end process;

end BEHAVIOUR ;

2.**Simulate the DAC001 module**:

* ***Test bench:***

vlib work

vcom -93 -work work DAC\_001.vhd

vsim DAC\_001

# view signal wave forms

view wave

# number format is hex

radix hex

# show input signals

add wave -divider -height 32 Inputs

add wave -height 30 -radix default N\_RES

add wave -height 30 -radix default SCLK

add wave -height 30 -radix default START

add wave -height 30 -radix default DA\_IN

# show reg signals

add wave -divider -height 32 Reg

add wave -height 30 -radix default DA\_DAT0

add wave -height 30 -radix default DA\_LD

add wave -height 30 -radix default DA\_STATE

add wave -height 30 -radix default DA\_NEXTSTATE

# show DAC signals

add wave -divider -height 32 ADC\_signals

add wave -height 30 -radix default DA\_CS0

add wave -height 30 -radix default DA\_CLK0

# show output of register

add wave -divider -height 32 Reg\_output

add wave -height 30 -radix default DA\_SD0

# generate input stimuli

force SCLK 0 0ns, 1 160ns -r 320ns

force N\_RES 1 0ns, 0 33ns, 1 57ns

force START 0 0ns, 1 1000ns, 0 1320ns, 1 20000ns, 0 22500ns, 1 30000ns

force DA\_IN AAA 0ns

run 50000ns

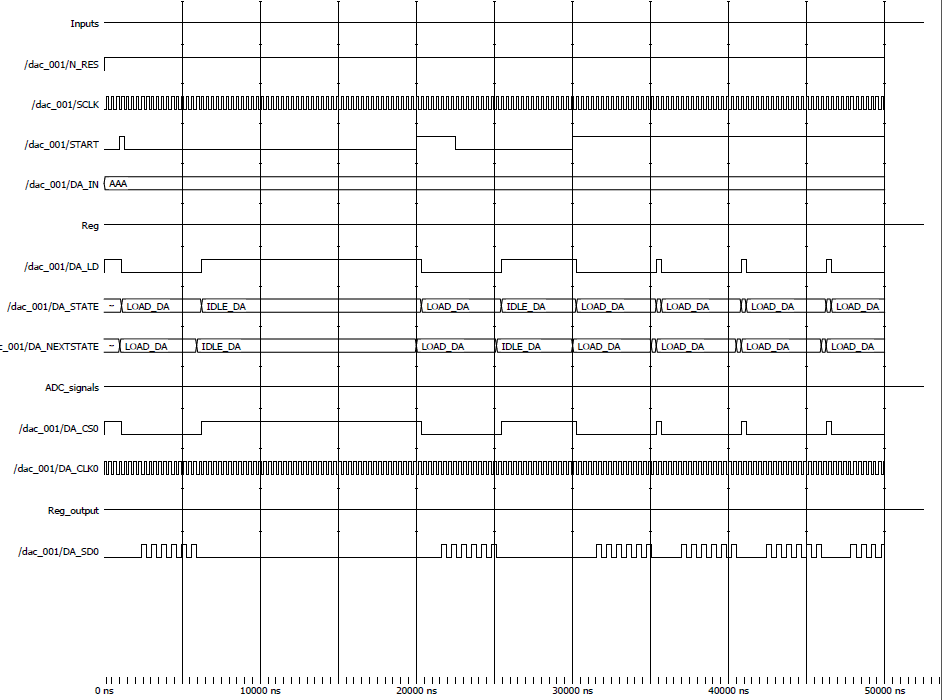


Figure 4: Simulation results

3.**Create an entity ADCDAC\_TOP that instantiates the module DAC\_001 S and adds the sample rate generator and the clock divider**.

Here is the VHDL code for ADCDAC\_TOP module.

library IEEE;

use IEEE.std\_logic\_1164.ALL;

use IEEE.std\_logic\_unsigned.ALL;

entity ADCDAC\_TOP is

port ( CLK, N\_RES: in bit; -- 100MHz; asynch, active low reset

DA\_IN : in bit\_vector(11 downto 0) ;

-- M\_ODE : in bit;

-- Signals to DAC0

DA\_SD0 : out bit ;

DA\_CLK0 : out bit ;

DA\_CS0 : out bit );

end ADCDAC\_TOP;

architecture BEHAVIOUR of ADCDAC\_TOP is

component DAC\_001 is

port ( SCLK, N\_RES : in bit;

DA\_IN : in bit\_vector(11 downto 0);

START : in bit; --Startsignal

RDY\_DA : out bit ; --Ready-Flag

---DAC0\_SIGNALS

DA\_SD0 : out bit ;

DA\_CS0 : out bit ;

DA\_CLK0 : out bit) ;

end component ;

component SR\_GEN is

port ( CLK, N\_RES : in bit; -- 100MHz; asynch, active low reset

START\_TE : out bit);

end component SR\_GEN;

component CLK\_DIV is

port ( CLK, N\_RES: in bit; -- 100 MHz; asynch, active low reset

SCLK : out bit); -- 3.125 MHz clk out

end component CLK\_DIV;

signal START\_TE : bit ;

--DAC-Signals

signal DA\_INT\_SD0 , DA\_INT\_CS0, DA\_INT\_CLK0 : bit ;

signal DIN\_TE0 : bit\_vector(11 downto 0) ;

--Takt, Taktteiler, Bitzaehler

signal SCLK : bit; -- interner Takt

begin

DIN\_TE0 <= DA\_IN ;

DA\_SD0 <= DA\_INT\_SD0 ;

DA\_CS0 <= DA\_INT\_CS0 ;

DA\_CLK0 <= DA\_INT\_CLK0 ;

DAC\_0 : DAC\_001

port map ( SCLK, N\_RES , DIN\_TE0 , START\_TE, open, DA\_INT\_SD0, DA\_INT\_CS0, DA\_INT\_CLK0 ) ;

SR\_GEN0 : SR\_GEN

port map ( CLK, N\_RES, START\_TE);

CLK\_DIV0: CLK\_DIV

port map (CLK, N\_RES,SCLK);

end BEHAVIOUR ;

4.**Simulate the ADCDAC\_TOP module**.

* ***Test bench:***

vlib work

vcom -93 -work work ADCDAC\_TOP.vhd

vsim ADCDAC\_TOP

view wave

radix hex

add wave -height 30 -radix default sim:/ADCDAC\_TOP/\*

force CLK 0 0ns, 1 160ns -r 320ns

force N\_RES 1 0ns, 0 33ns, 1 57ns

#force START\_TE 0 0ns, 1 1000ns, 0 1320ns, 1 20000ns, 0 22500ns, 1 30000ns

force DA\_IN AAA 0ns 10ns, DDD 25000ns

run 60000ns

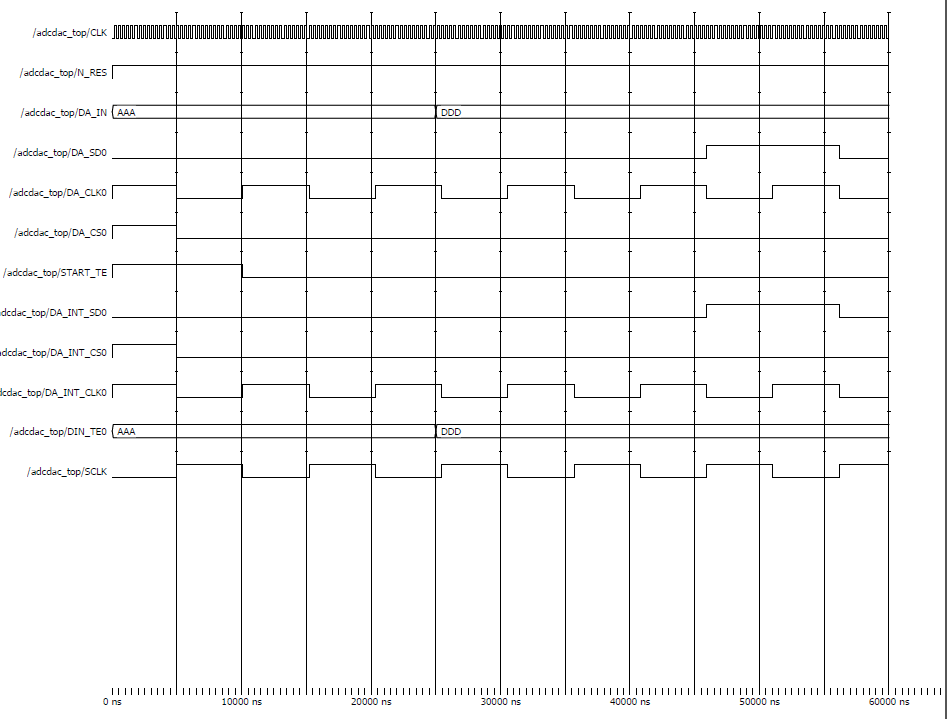


Figure 5: Simulation results

5.**Implement the top level module ADCDAC\_TOP including DAC module on the ML507 Board**:

After simulating DAC and ADCDCA\_TOP modules with ModelSim software and getting the desired results we started implementing the ADCDCA\_TOP module on the FPGA and tested the DAC with with 12 switches of the SLB board and applied different digital values to our DAC module and compared the value to the analog output voltage of the DAC.We set M\_ODE to 0 by turning off the 7th switch of the SLB board to get the data from it.We obsereved that the output voltage range was not between -1 and +1, and was lower instead, and that was because of using an amplifier extra added to the circuit.So with the help of the lecturer we changed the code(shifted the data being saved in the shift register by two the left) so that the output was in the range of -1 and +1. Below are the results of different test cases using oscilloscope.

**Hint** : The signals in the screenshots are CS0,CLK0 and D0 of DAC8801 from up to down.

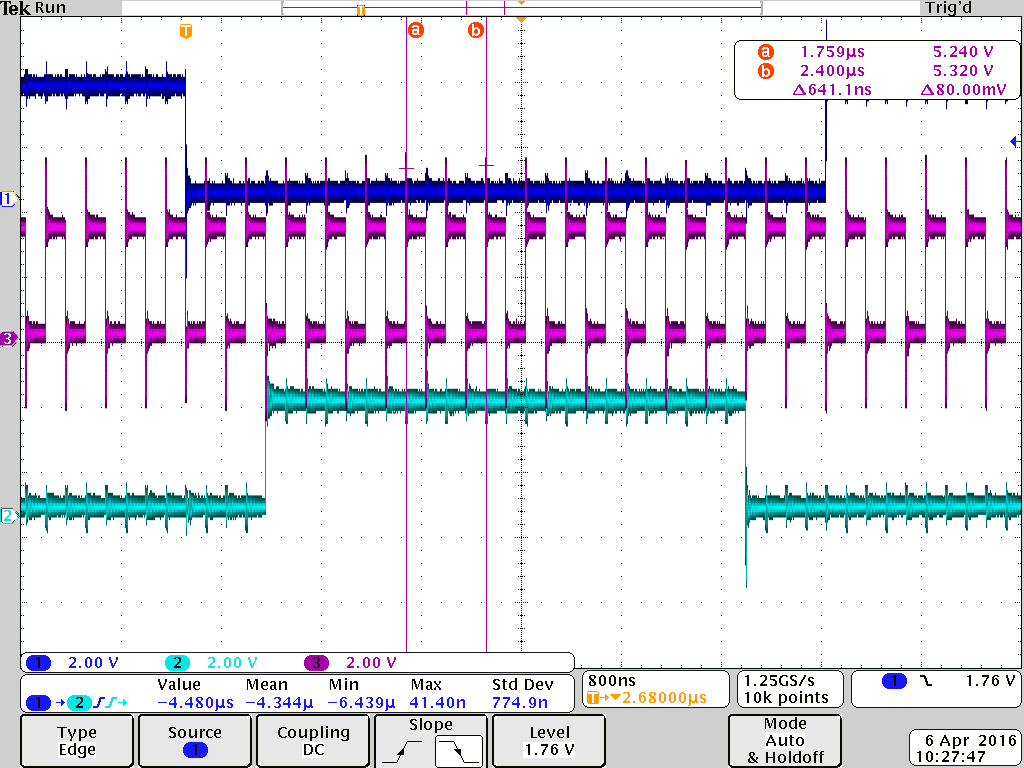


Figure 6: set all the switches on

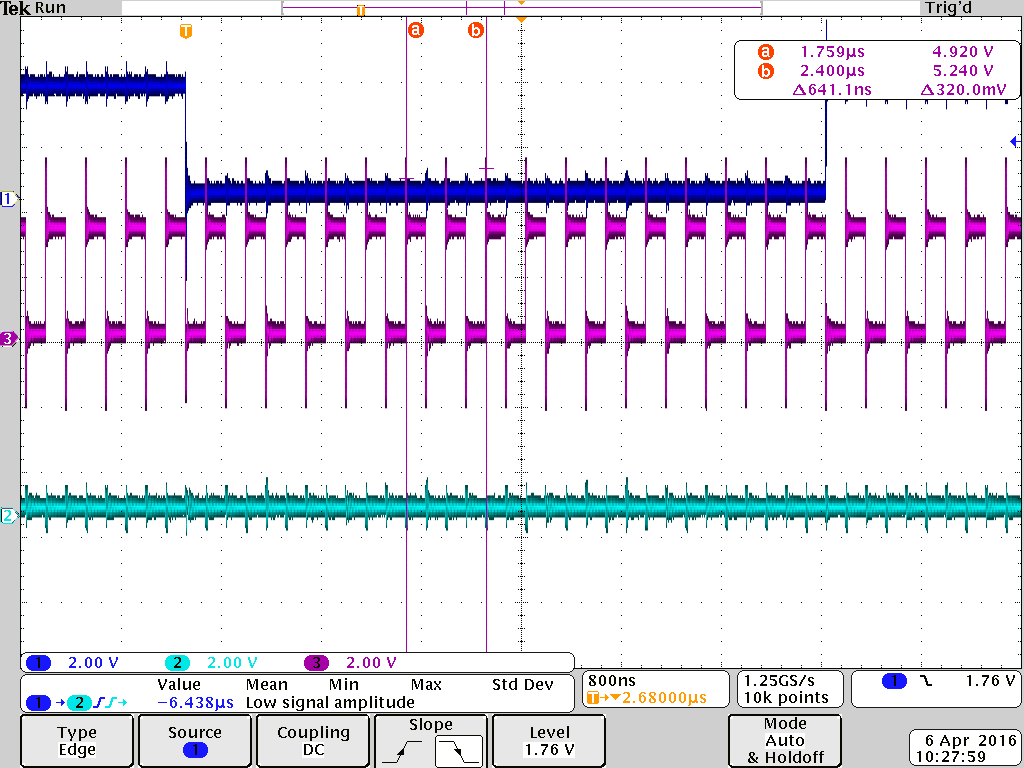


Figure 7: set all the switches off

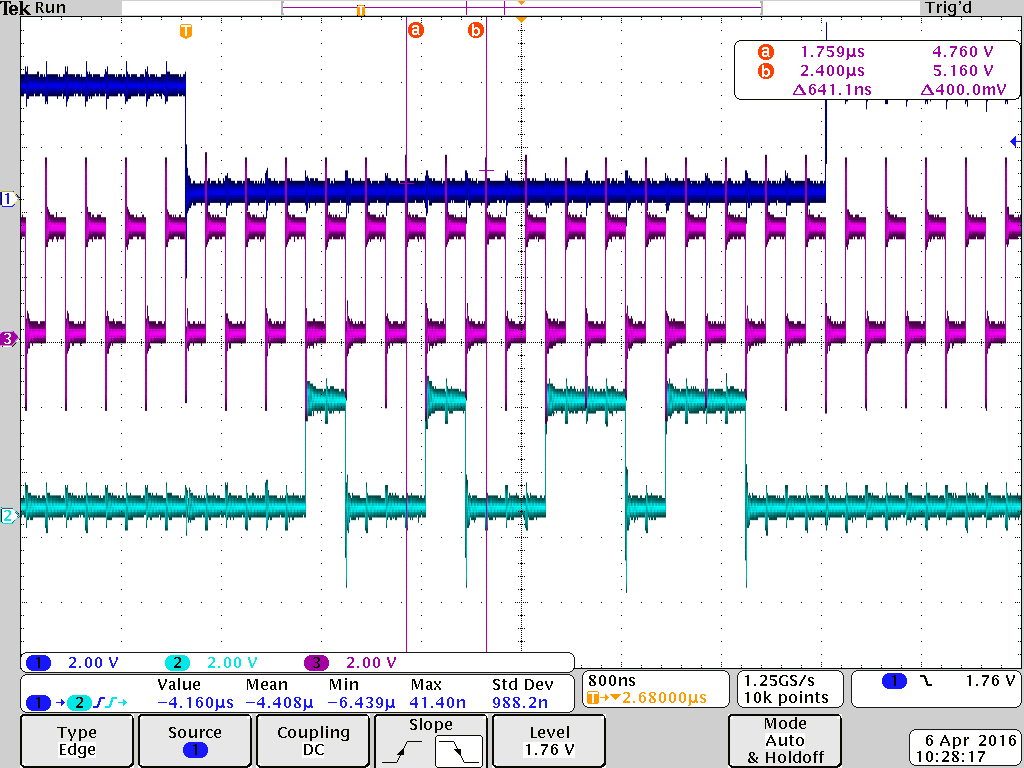


Figure 8: set 0100 1001 1011 as a random pattern

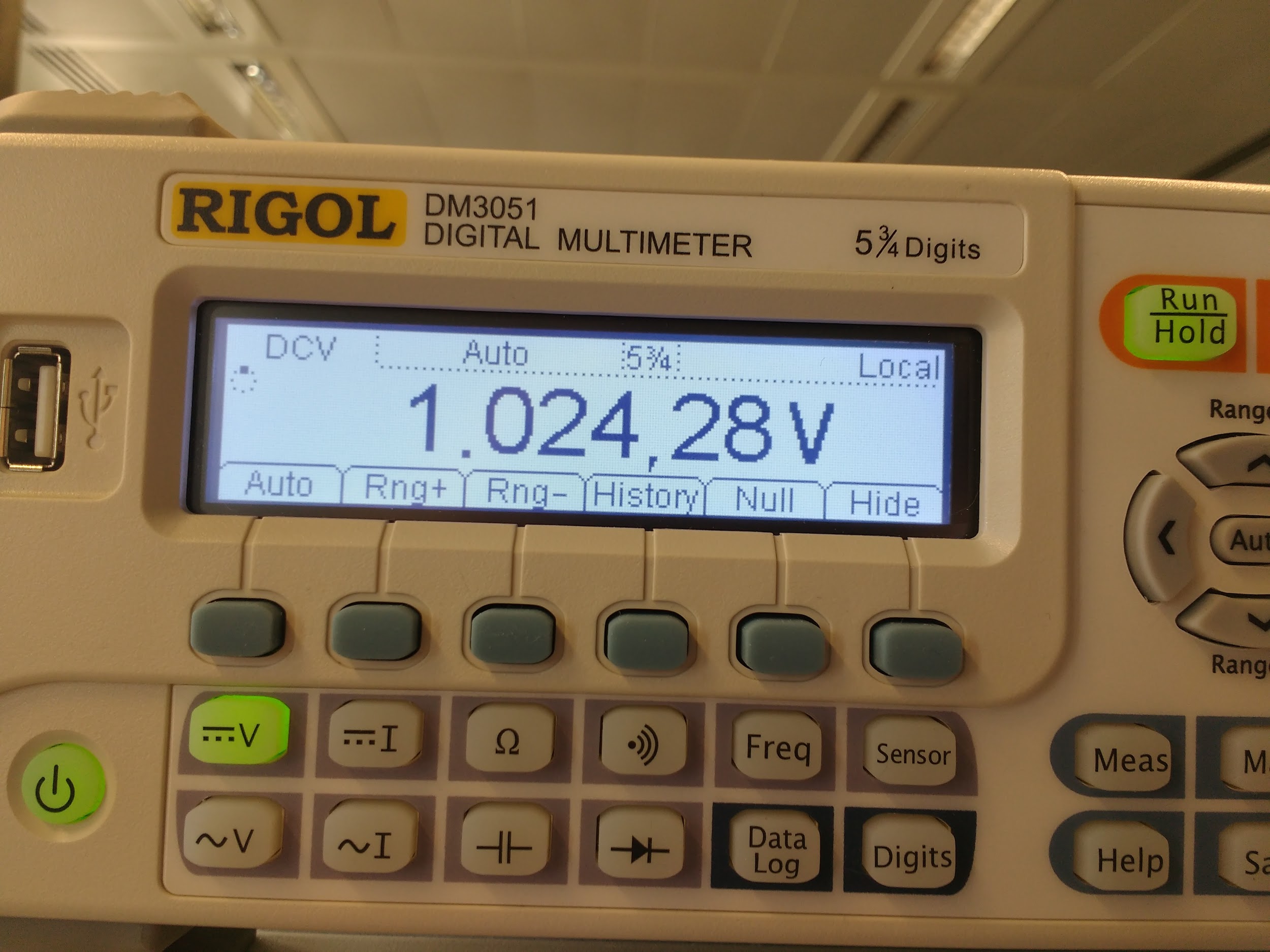


Figure 9: testing the max value of DAC output0

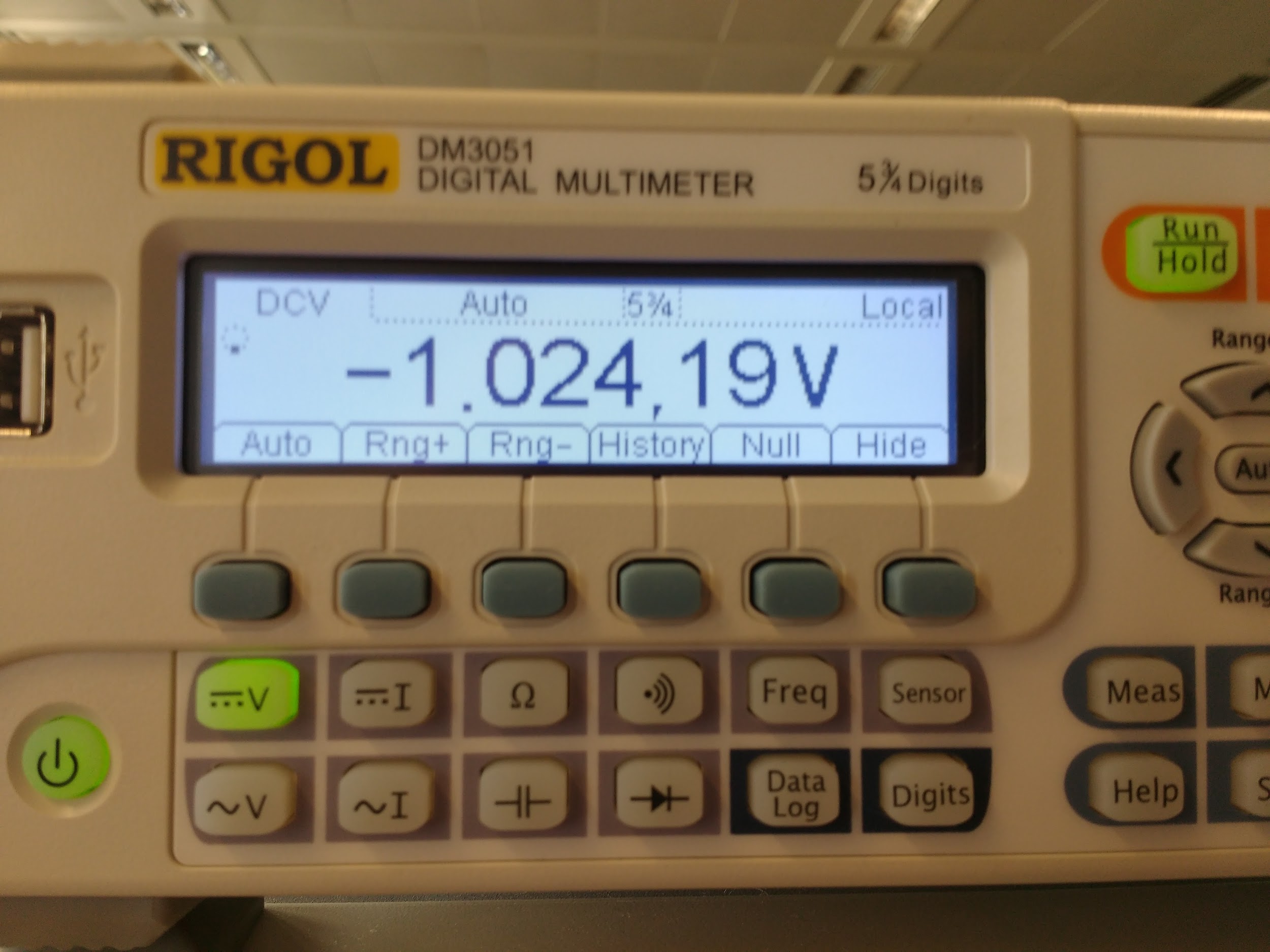


Figure 10: testing the min value of DAC output0

6. **Implement the top-level module ADCDAC\_TOP including DAC and ADC**:

Next we applied a voltage in the range -1V to +1V to the ADC and

measure the serial data signals from ADC and to DAC and compared the results.

Afterwards we applied a sinusoidal signal with amplitude of 1V to

the ADC and determined the amplitude and delay of the DAC output.Below are the screenshots of the results shown by the oscilloscope.

**Hint**: The purple signal in the following screenshots is the input and the blue one is the output of module ADCDAC\_TOP.

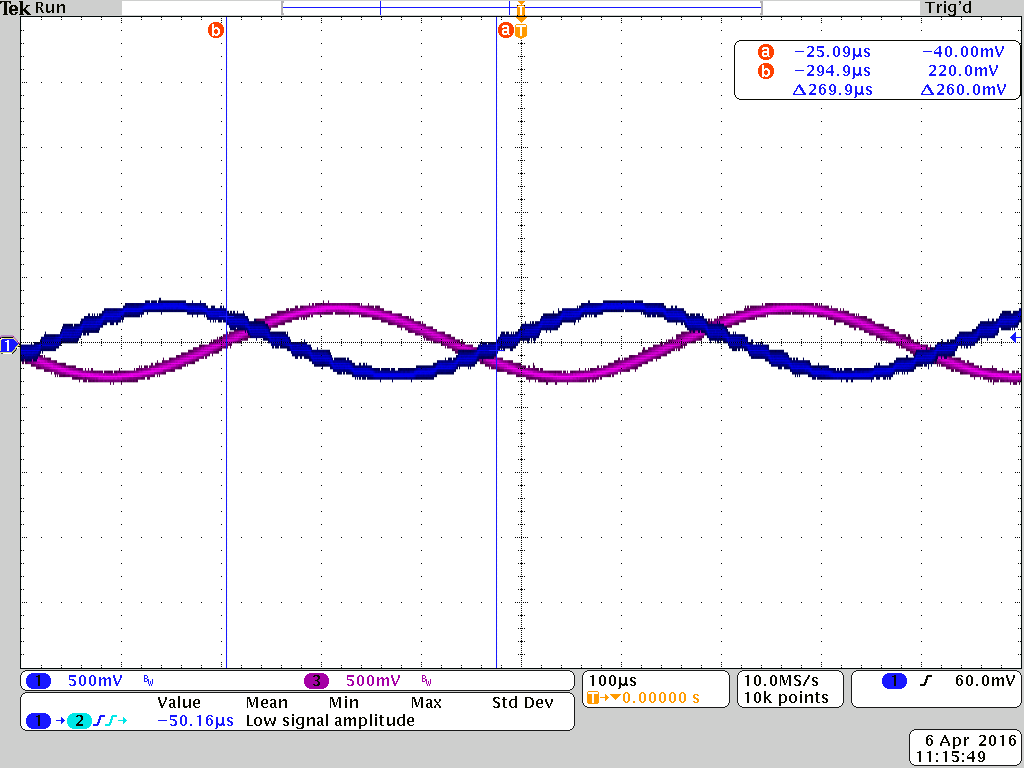


Figure 11: Check with 2.2 kHz frequency we had delay of 269.9µs.

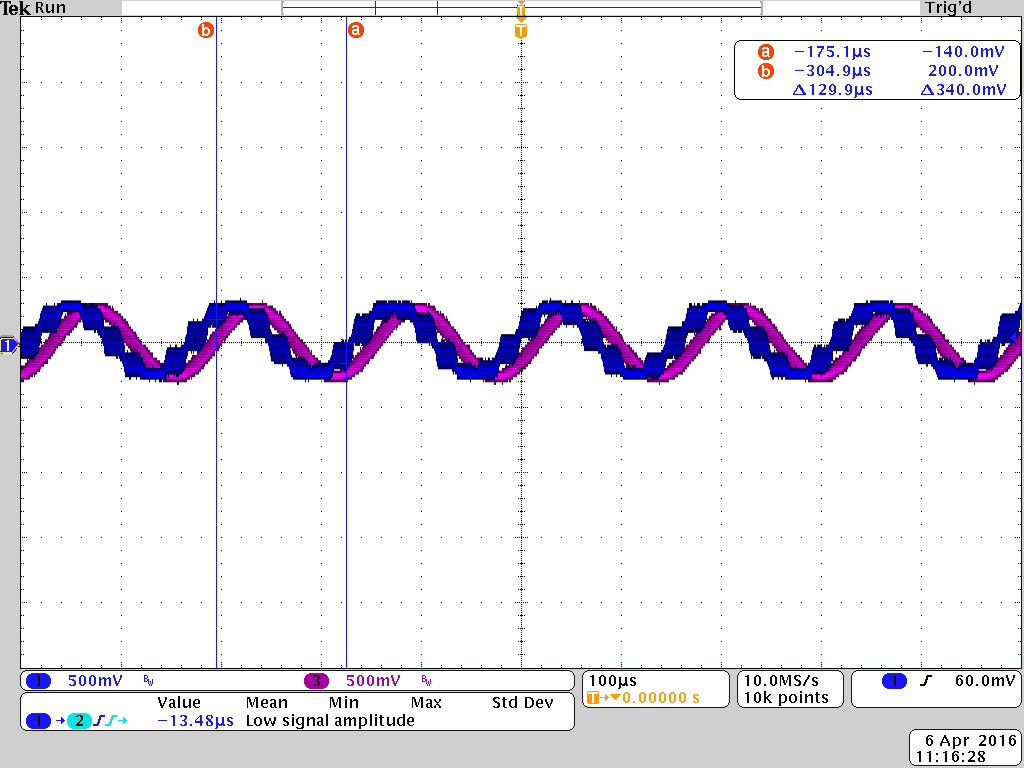


Figure 12: Check with 6.2 kHz frequency we had delay of 129.9µs.

**Conclusion**: We observed that the input and output signals were similar to each other but there was delay between them. When we increased the frequency of the input signal,the propagation delay decreased,however the output accuracy became lower.