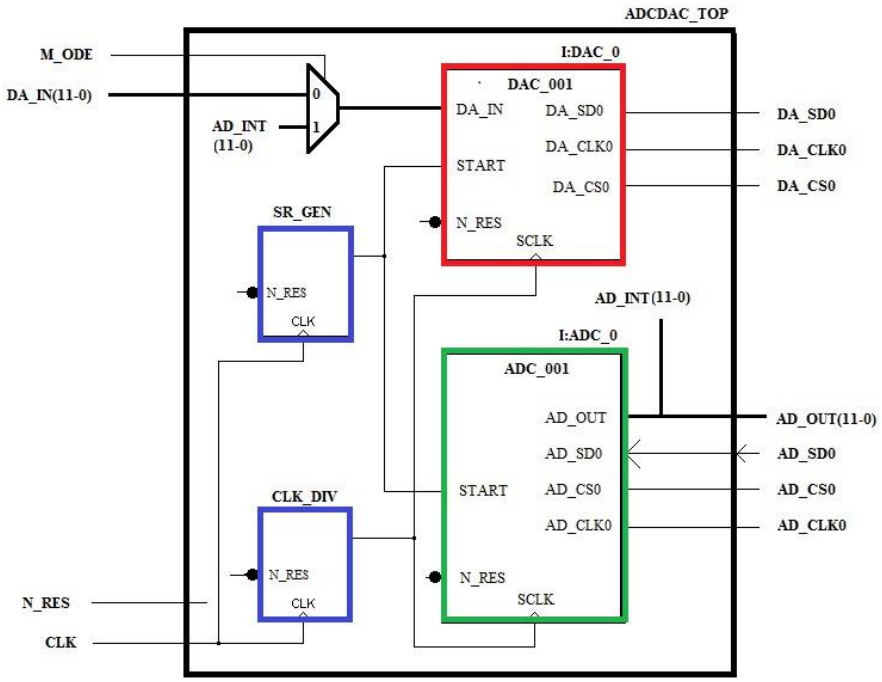
IE4-DS LAB SESSION #1

Surname, First Name	Group	Team
1. **Shahab ShafieiHajiabady** 2. **Bahram Samavatian** 3. **Ali Emami**	1 x 2 ☐ 3 ☐	A B C D E F

This preparation sheet must be filled and uploaded to EMIL by every student before the lab session. Cooperation within the lab team (of 2-3 students) is possible (same solution). However, no cooperation across lab teams.



PREP TASK 1.1

PREP TASK 1.1: Design the module DAC_001 that communicates with the DAC8801. Use an FSM that sends the bits serially to the DAC after the START=1 pulse occurred. The bits are read from a 16bit shift register SR_DAC0.

SPECIFICATION (VERBALLY)

- DA_IN is a 12-bit input digital value , that should be sent to the DAC_001.
- DA_IN input signal can be chosen by different Modes (0,1)
- A full 16- bit Data word can be loaded into the serial register, but only last 14 bits are transferred to the DAC register when low active chip select goes HIGH.
- At the falling edge of low active CS, the bits of 14-bit data word are sent one by one (MSB first).
- FSM will remain in the IDLE_DA state as long as the START signal is '0'
- Once the START is '1' , FSM will send the bits serially to DAC.

SIMULATION TEST CASES (VERBALLY)

- Count number of clock cycles
- Check if the states are changing correctly
- Check if chip select and reset is working properly

VHDL MODEL

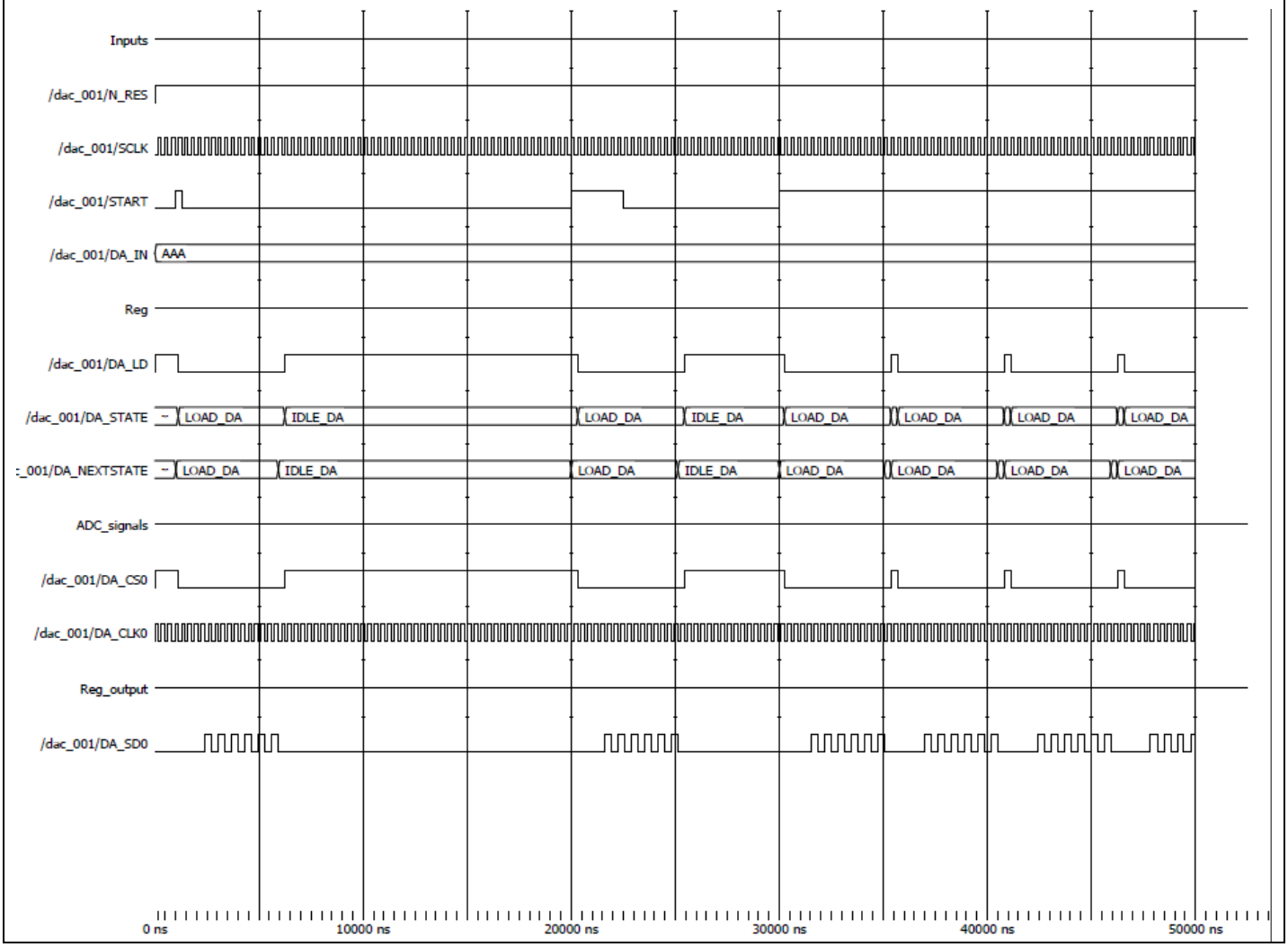
```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity DAC_001 is
port(SCLK, N_RES : in bit;
START : in bit; -- start signal
DA_IN : in bit_vector(11 downto 0) ;
RDY_DA : out bit;
DA_SD0 : out bit ;
DA_CS0 : out bit ;
DA_CLK0 : out bit );
end DAC_001 ;
architecture BEHAVIOUR 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 dont 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 dont clear untill 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 ;
```

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
```

SIMULATION RESULTS



PREP TASK 1.2

PREP TASK 1.2: Create an entity ADCDAC_TOP that instantiates the module DAC_001 SR_GEN and adds the sample rate generator and the clock divider.

SPECIFICATION (VERBALLY)

- The clock divider shall generate SCLK with 3.125MHz generated from a global clock with 100MHz (100 MHz / 32)
- Sample rate generator shall generate a START pulse of 320 ns duration (32/100MHz) at a frequency of 48 kHz.
- The module shall be reset by a an active low asynchronous reset
- A counter shall be used to generate period and duty cycle of START signal

SIMULATION TEST CASES (VERBALLY)

VHDL MODEL

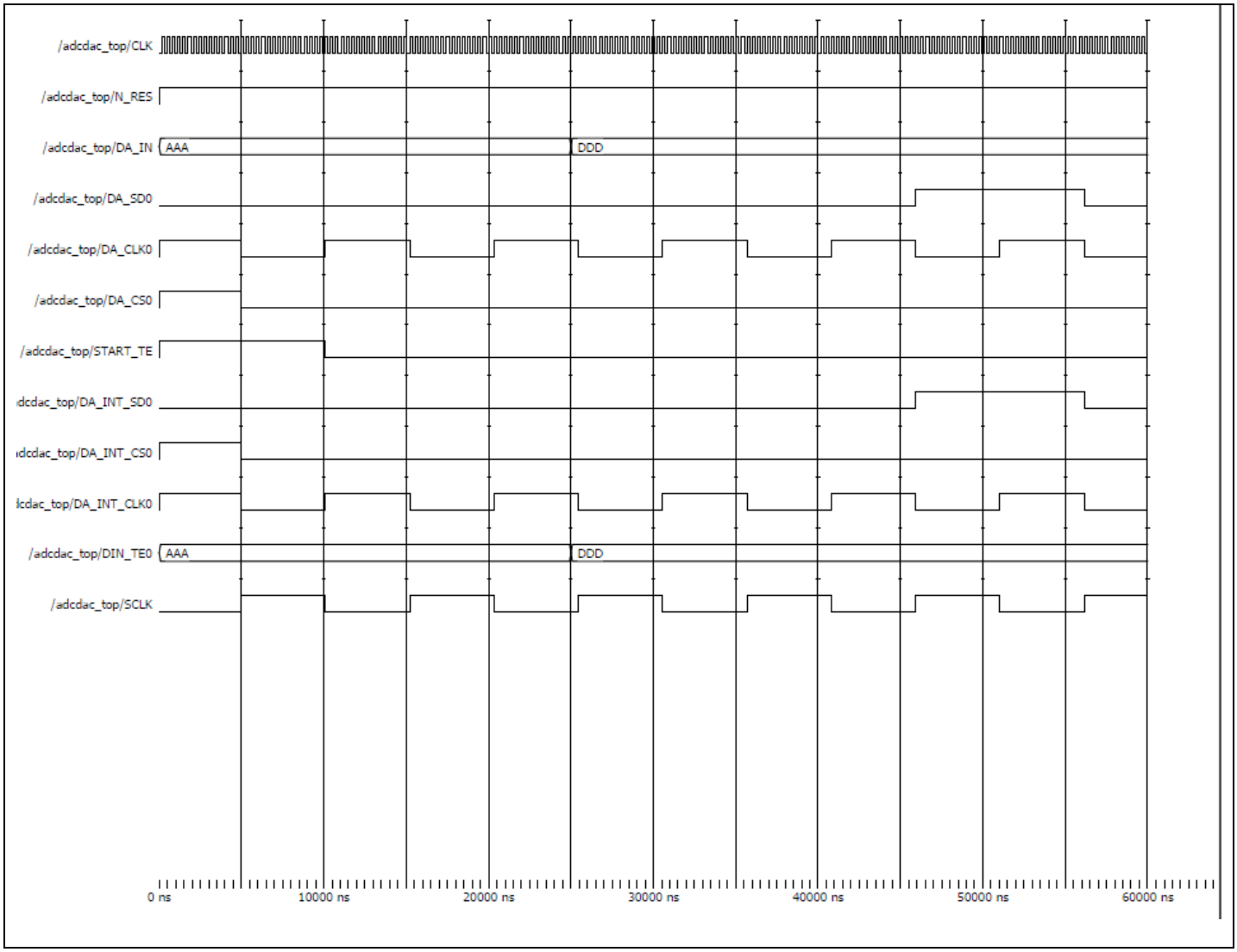
```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.ALL;
entity ADCDAC_TOP is
```

```
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 ;
```

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
```

SIMULATION RESULTS



SIMULATION TEST CASES (VERBALLY)

VHDL MODEL

TEST BENCH

SIMULATION RESULTS