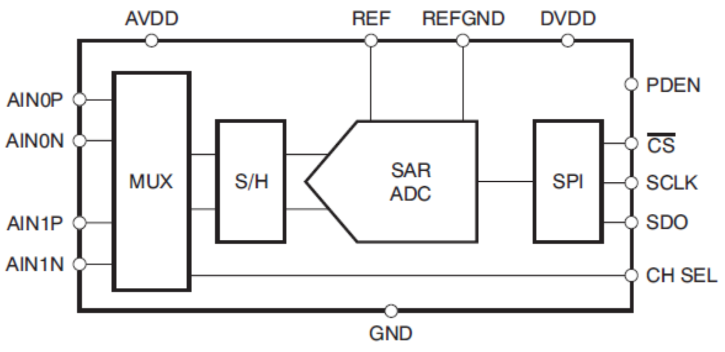
IE4-DS LAB SESSION #1: LECTURE PREPARATION

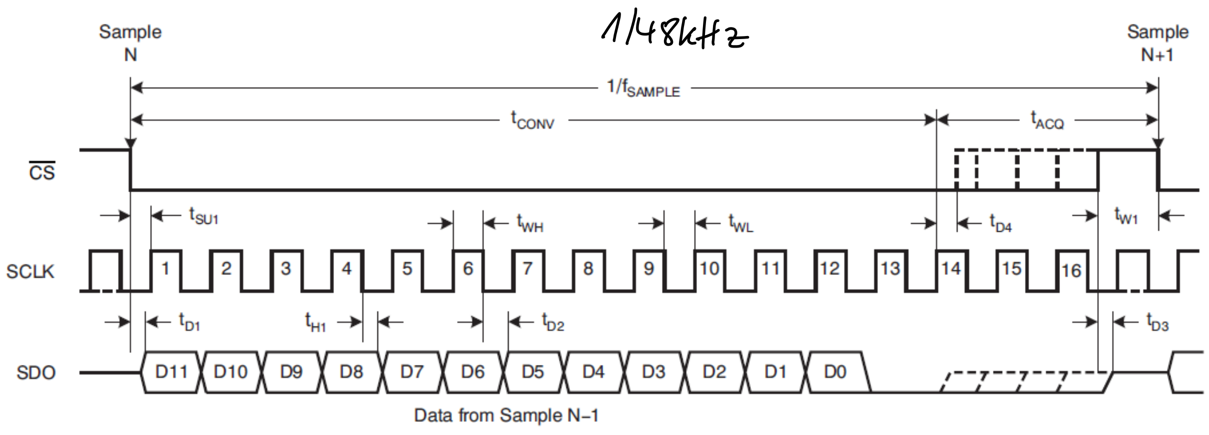
Surname, First Name	Group	Team
1. all of us	1 ☐	A B C D E F
2.	2 ☐	
3.	3 ☐	

TASK

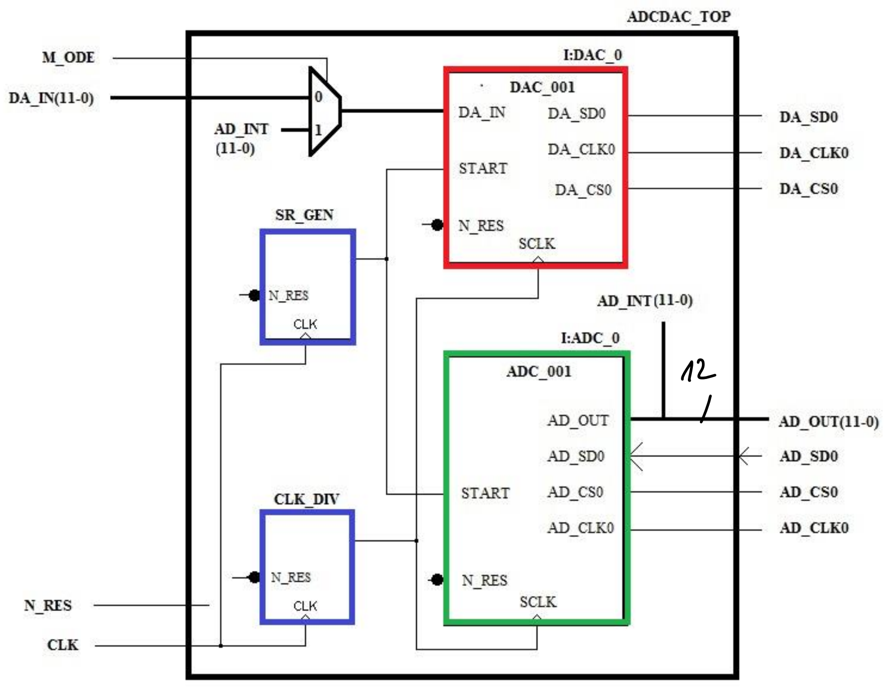
Design a digital module ADCDAC_TOP in VHDL that reads data from an ADS7947 (http://www.ti.com/lit/ds/symlink/ads7947.pdf). The communication to the FPGA (Virtex 7 FPGA on an Xilinx ML507 board) is accomplished by a 3-wire synchronous serial transmission (SCLK, SDO, -CS).



The timing of the three signals is as follows (to read a 12-bit ADC value).



The FPGA module ADCDAC_TOP shall have the following blocks:



The module ADC_001 that reads data from the ADC and converts them into a parallel 12bit word AD_OUT(11:0), a clock divider CLK_DIV for that generates a 3.125 MHz clock signal (which is clock speed of FPGA 100MHz / 32) and a sample rate generator module SR_GEN that generates every 1/48 kHz a START pulse that triggers a new reading of the external ADC. The DAC module will be done by yourself during prep of the lab.

TASK 1: DESIGN OF THE CLOCK DIVIDER CLK_DIV

Write a VHDL module CLK_DIV with a clock divider and simulate the module.

SPECIFICATION (VERBALLY)

- the clock divider shall generate SCLK with 3.125MHz generated from a global CLK with 100MHz
- a modulo-32 (5bit) counter shall be used of which the MSB is the SCLK output
- the module shall be reset by an active low asynchronous reset

SIMULATION TEST CASES (VERBALLY)

- verify for a least 3 periods that a 3.125 MHz output is generated
- verify that the reset is asynchronous by applying a reset not aligned to clock edges and that it is active low and that the CLK_CNT value is “00000” when reset
- verify that CLK_CNT increments correctly

VHDL MODEL

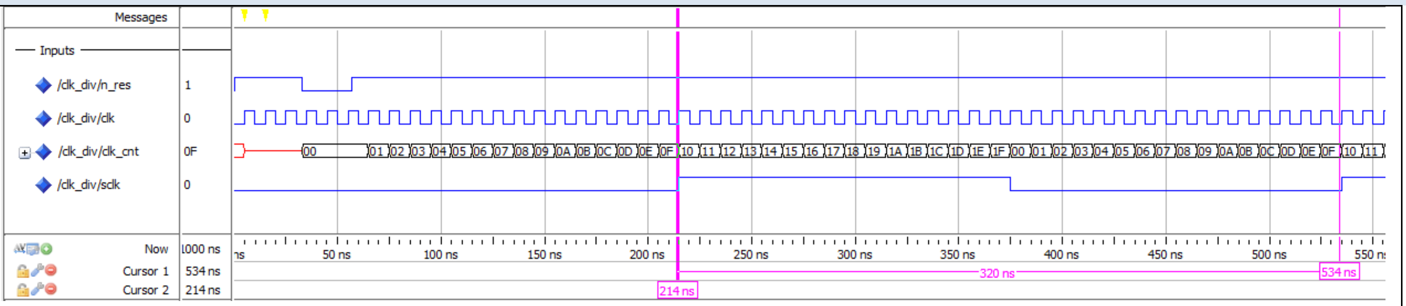
```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.ALL;


entity CLK_DIV is
        port ( CLK, N_RES: in  bit;    -- 100 MHz; async, active low reset
            SCLK      : out bit); -- 3.125 MHz clk out
end CLK_DIV;


architecture BEHAVIOUR of CLK_DIV is

signal CLK_CNT : std_logic_vector(4 downto 0);
```

```
begin

-- clock divider
CLK_DIV : process(N_RES,CLK)
  begin
      if N_RES ='0'  then
               CLK_CNT <= (others =>'0');
      elsif CLK  = '1' and CLK'event then
            CLK_CNT <= CLK_CNT + '1'  ;
      end if;
end process CLK_DIV;

SCLK <= to_bit(CLK_CNT(4)) ;

end BEHAVIOUR ;
```

TEST BENCH

```
# show input signals
add wave -divider -height 32 Inputs
add wave -height 30 -radix default N_RES
add wave -height 30 -radix default CLK

# show internal signals
add wave -height 30 -radix default CLK_CNT

# show output signals
add wave -height 30 -radix default SCLK

# generate input stimuli
force CLK     0  0ns, 1  5ns -r 10ns
force N_RES   1  0ns, 0 33ns, 1 57ns
run 1000ns
```

SIMULATION RESULTS


Messages
Inputs
/clk_div/n_res 1
/clk_div/clk 0
/clk_div/clk_cnt 0F
/clk_div/sclk 0
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
Now 1000 ns
Cursor 1 534 ns
Cursor 2 214 ns
ns 50 ns 100 ns 150 ns 200 ns 250 ns 300 ns 350 ns 400 ns 450 ns 500 ns 550 ns
320 ns
534 ns
214 ns

TASK 2: DESIGN OF THE SAMPLING RATE GENRATOR SR_GEN

Write a VHDL module SR_GEN that generates the START_TE pulse at a rate of 48kHz and simulate the module.

SPECIFICATION (VERBALLY)

- sample rate generator shall generate a START pulse of 320ns duration (32/100MHz) at a frequency of 48 kHz
- a counter shall be used to generate period and duty cycle of START signal
- the module shall be reset by an active low asynchronous reset

SIMULATION TEST CASES (VERBALLY)

- verify for a least 3 periods that a pulse every 1/48kHz =20.8333…µs is generated
- verify that the reset is asynchronous by applying a reset not aligned to clock edges and that it is active low and that the CLK_CNT value is “00000” when reset
- verify that CLK_CNT increments correctly

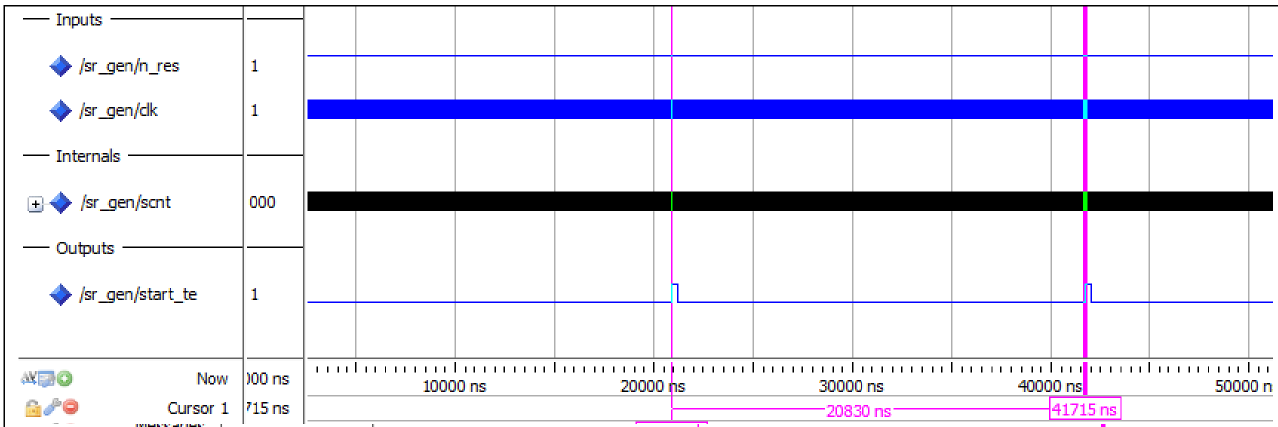
VHDL MODEL

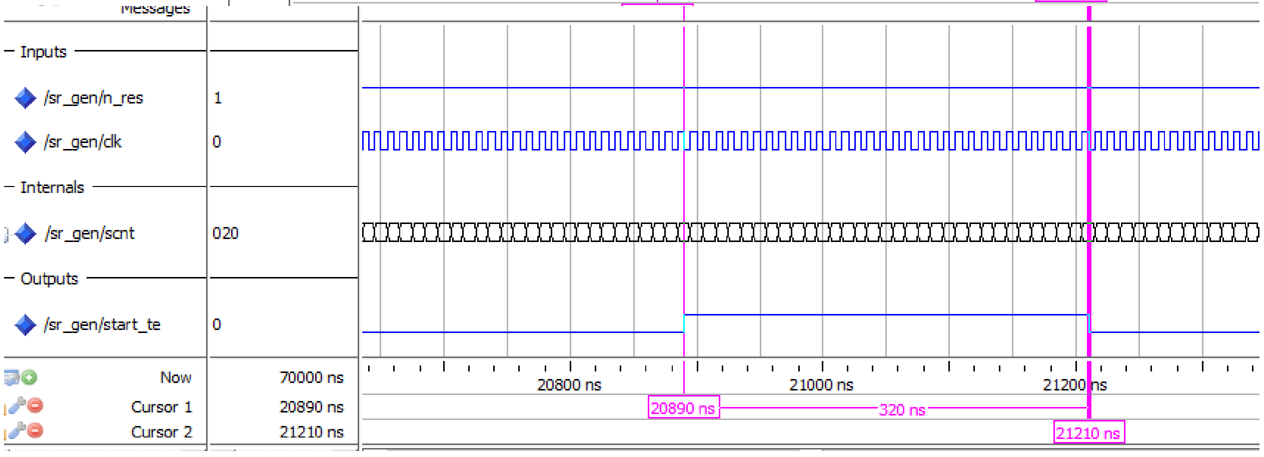
```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.ALL;


entity SR_GEN is
      port ( CLK, N_RES : in bit; -- 100MHz; async, active low reset
          START_TE   : out bit);
end SR_GEN;


architecture BEHAVIOUR of SR_GEN is

signal SCNT : std_logic_vector(11 downto 0);

begin

SR_GEN:process(N_RES,CLK)
  begin
     if N_RES ='0'  then
             SCNT <= (others=>'0');
     elsif CLK  = '1' and CLK'event then
         if SCNT = "100000100010" then -- 100MHz/48kHz=2083.33
                                       -- range 0...2082
             SCNT <= (others=>'0');
         else
                SCNT <= SCNT + '1'  ;
             end if;
     end if;
  end process SR_GEN;
  START_TE <= '1' when SCNT < "000000100000" else '0' after 5 ns;

end BEHAVIOUR ;
```

TEST BENCH

```
# show input signals
add wave -divider -height 32 Inputs
add wave -height 30 -radix default N_RES
add wave -height 30 -radix default CLK

# show internal signals
add wave -divider -height 32 Internals
add wave -height 30 -radix default SCNT

# show output signals
add wave -divider -height 32 Outputs
add wave -height 30 -radix default START_TE

# generate input stimuli
force CLK     0  0ns, 1  5ns -r 10ns
force N_RES   1  0ns, 0 33ns, 1 57ns
run 70000ns
```

SIMULATION RESULTS



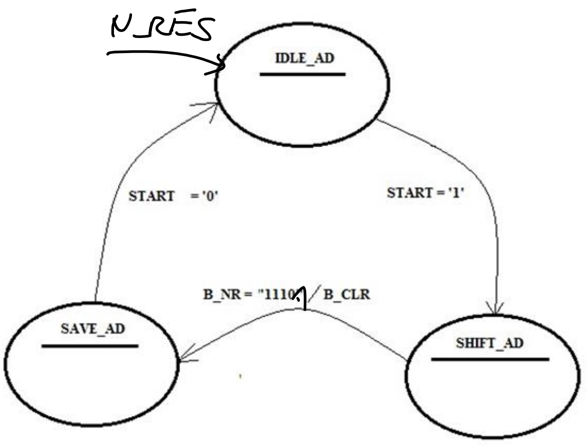


TASK 3: DESIGN OF THE ADC COMMUNICATION MODULE

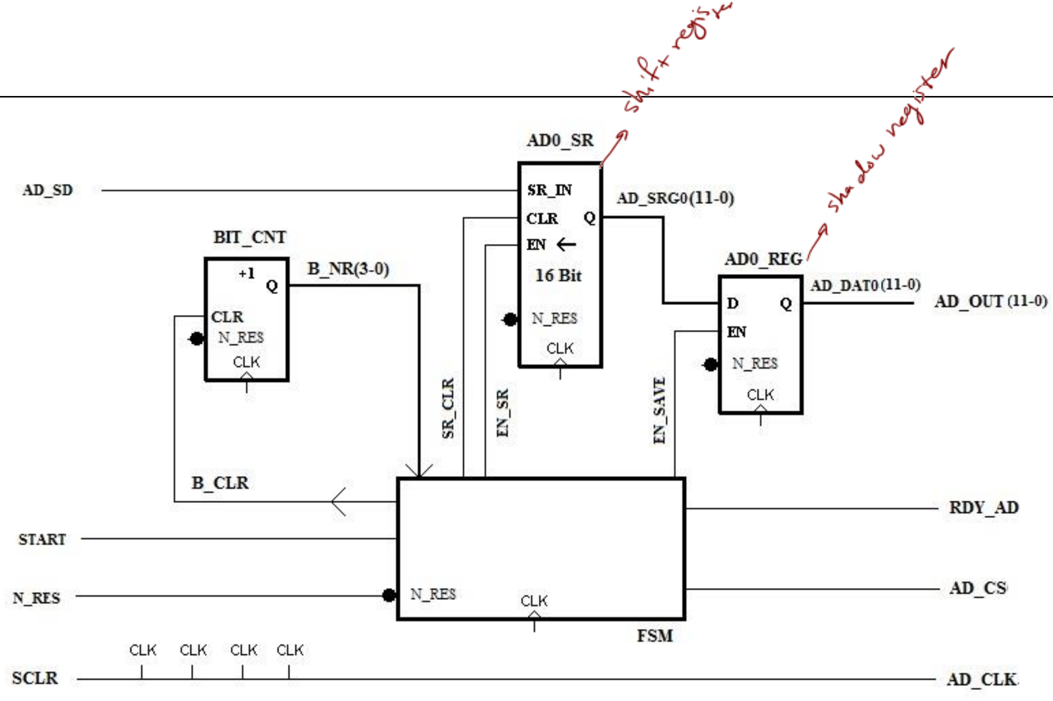
Write a VHDL module ADC_001 that that controls AD_CLK0 and AD_CS0 of the ADC and receives data AD_SD0. The module shall convert the serial bit stream from the ADC into a parallel 12bit data word.

SPECIFICATION (VERBALLY)

- The serial bit stream is initiated and read by an FSM and a counter BIT_CNT
- When the FSM is in AD_IDLE it waits for a next sampling point (START='1')
- After START='1', -CS is pulled low and 16 bits are read in (12 data bits plus 4 zeros for acquisition phase) in state Shift_AD



- Received values are shifted into shift register AD0_SR (MSB first)



- -CS is pulled high afterwards and contents of shift register is copied to AD0_REG (highest 12 bits) in state SAVE_AD
- SR_CLR clears the shift register
- Shifting is done when EN_SR=1

SIMULATION TEST CASES (VERBALLY)

- After reset the FSM is in IDLE_AD
- With START=1 the FSM transits from IDLE_AD to SHIFT_AD
- In state SHIFT_ADD 16 values are read in correctly from ADC (input voltage needs to be known), 4 zeros correctly appended
- After reading 16 values, the FSM transits to SAVE_AD
- In SAVE_AD the 12bit value is shifted to AD0_REG
- BIT_CNT is correctly cleared when ADC value was read in
- Next 16bit ADC sample is correctly read in after sample period 1/48kHz

VHDL MODEL

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ADC_001 is
       port ( SCLK, N_RES : in bit;
          AD_OUT    : out bit_vector(11 downto 0);
                  START     :  in bit;   -- start signal
                  RDY_AD    : out bit ; -- ready flag
              AD_SD     :  in bit ;
          AD_CS     : out bit ;
          AD_CLK    : out bit );
end ADC_001 ;

architecture BEHAVIOUR of ADC_001 is
-- bit counter
signal  B_NR    : unsigned(3 downto 0):= "0000" ;
signal  B_CLR   : bit   ;

--Control Signals ADC
signal  SR_CLR  : bit;   -- clear shift reg
signal  EN_SR   : bit;   -- enable shift reg
signal  EN_SAVE : bit;   -- store

-- registers for ADC data
signal  AD_SRG0 : bit_vector(15 downto 0):= (others => '0')  ;  -- shift reg
signal  AD_DAT0 : bit_vector(11 downto 0):= (others => '0')  ;  -- save reg

type    State_Type_AD is (IDLE_AD,SHIFT_AD,SAVE_AD);
-- states of ADC communication module
signal  AD_STATE         : State_Type_AD;
```

```vhdl
signal  AD_NEXTSTATE    : State_Type_AD;

begin

AD_OUT <= AD_DAT0;

-- clock to ADC
AD_CLK <=  not SCLK;

-- counter for serial bits ------------------------------
BIT_CNT: process(SCLK)
   begin
     if SCLK = '1' and SCLK'event then
          if B_CLR = '1' then
              B_NR <= "0000" after 5 ns ;
         else
              B_NR <= B_NR + 1 after 5 ns ;
         end if ;
     end if ;
end process BIT_CNT;

---------ADC0 - shift / save regs ----------------------------
AD0_SR: process(SCLK, N_RES)
   begin
     if N_RES ='0'  then
            AD_SRG0 <= (others=>'0')  after 5 ns ;
         elsif SCLK = '1' and SCLK'event then
               if SR_CLR = '1' then
                AD_SRG0 <= (others=>'0')  after 5 ns ;
            elsif EN_SR = '1' then
                AD_SRG0 <= AD_SRG0(14 downto 0) & AD_SD after 5 ns ;
            end if;
     end if ;
end process AD0_SR;

AD0_REG: process(SCLK, N_RES)
   begin
     if N_RES ='0'  then
            AD_DAT0 <= (others=>'0')  after 5 ns ;
         elsif SCLK = '1' and SCLK'event then
            if EN_SAVE = '1' then
                AD_DAT0  <=  AD_SRG0(15 downto 4) after 5 ns ;
        end if;
     end if ;
end process AD0_REG;

-- FSM_ADC comm module ----------------------------------------
FSM_AD_REG : process(SCLK , N_RES)
   begin
     if N_RES ='0'  then
        AD_STATE <= IDLE_AD after 5 ns ;
     elsif SCLK  = '1' and SCLK 'event then
        AD_STATE <= AD_NEXTSTATE after 5 ns ;
     end if ;
end process ;

FSM_AD_SN : process(AD_STATE,B_NR,START)
   begin
        AD_NEXTSTATE <= AD_STATE after 5 ns ;
        AD_CS   <= '1' after 5 ns ;
        EN_SR <= '0' after 5 ns ;
        SR_CLR <= '0' after 5 ns ;
        EN_SAVE <= '0' after 5 ns ;
                RDY_AD <= '0' after 5 ns ;
                B_CLR  <= '1' after 5 ns ;
    case AD_STATE is
        when IDLE_AD => SR_CLR <= '1' after 5 ns ;
                                RDY_AD <= '1' after 5 ns ;
                                if START = '1' then
                                  AD_NEXTSTATE <= SHIFT_AD after 5 ns ;
                        end if ;
        when SHIFT_AD => AD_CS   <= '0' after 5 ns ; -- 15 states
```

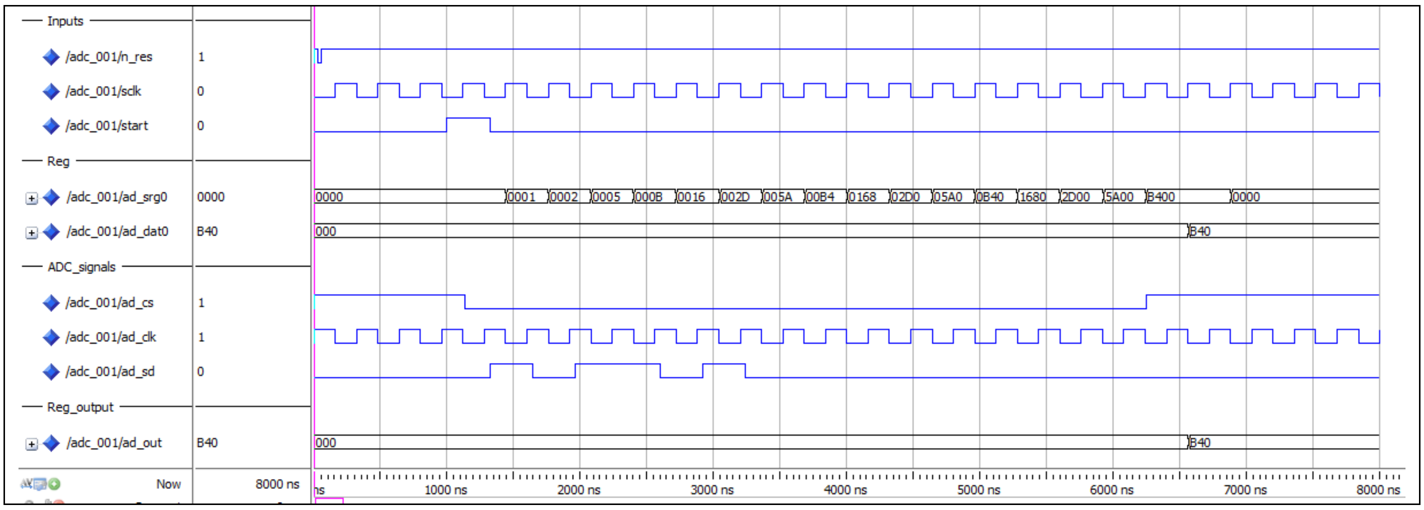
```
                              EN_SR <= '1' after 5 ns ;
                              B_CLR <= '0' after 5 ns ;
                      if B_NR = "1111" then
                        B_CLR <= '1' after 5 ns ;
                                            AD_NEXTSTATE <= SAVE_AD after 5 ns ;
                                          end if;
        when SAVE_AD => EN_SAVE <= '1' after 5 ns ;
                                                          if START = '0' then
                            AD_NEXTSTATE <= IDLE_AD after 5 ns ;
                                                          end if;
    end case;
end process;

end BEHAVIOUR ;
```

TEST BENCH

Does not cover all relevant cases (!)

```
# --- compilation
vlib work
vcom -93 -work work ADC_001.vhd
vsim ADC_001
# --- simulation
# 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

# show reg signals
add wave -divider -height 32 Reg
add wave -height 30 -radix default AD_SRG0
add wave -height 30 -radix default AD_DAT0

# show ADC signals
add wave -divider -height 32 ADC_signals
add wave -height 30 -radix default AD_CS
add wave -height 30 -radix default AD_CLK
add wave -height 30 -radix default AD_SD

# show output of register
add wave -divider -height 32 Reg_output
add wave -height 30 -radix default AD_OUT

# 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
force AD_SD   0  0ns, 1 1320ns, 0 1640ns, 1 1960ns, 0 2600ns, 1 2920ns, 0 3240ns
run 8000ns
```

SIMULATION RESULTS


Inputs
/adc_001/n_res 1
/adc_001/sclk 0
/adc_001/start 0
Reg
/adc_001/ad_srg0 0000
0000 0001 0002 0005 000B 0016 002D 005A 00B4 0168 02D0 05A0 0B40 1680 2D00 5A00 B400 0000
/adc_001/ad_dat0 B40
000 B40
ADC_signals
/adc_001/ad_cs 1
/adc_001/ad_clk 1
/adc_001/ad_sd 0
Reg_output
/adc_001/ad_out B40
000 B40
Now 8000 ns
ns
1000 ns
2000 ns
3000 ns
4000 ns
5000 ns
6000 ns
7000 ns
8000 ns