Digital Logic Laboratory

Report On LAB #6 - ADC Reading

Ana Maghradze

20th December 2021

Contents

1 Description of The Assignment

In this assignment, we should use ADC converter built in the FPGA to read analog values, convert them into digital values and show result on the Seven Segment Display. The ADC uses serial communication and SPI interface for that. It has 8 channels each with 12-bit resolution and we should use one of the channels for this lab. The range of channel analog inputs for the ADC is between 0 and 4.096 volts. Also, we need to have clock frequency of less than 40MHz, as provided in the LTC2308 analog to digital converter datasheet.

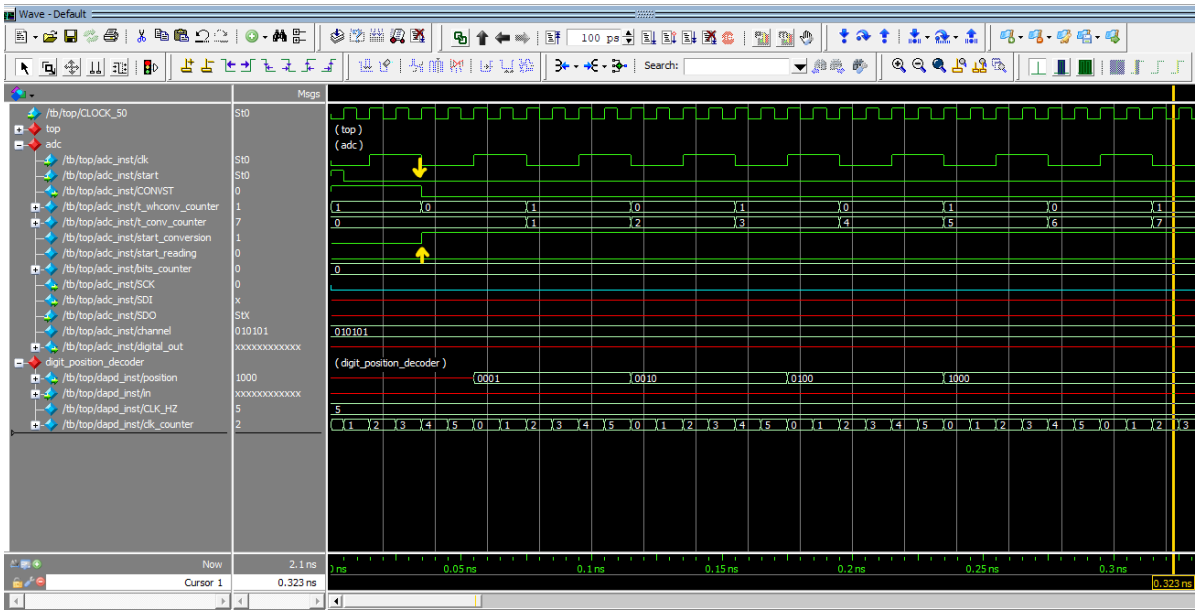
2 Description of The Solution

In the ADC module, I receive already divided clock from top module. when start signal is received, CONVST is set to 1 for few nano-seconds, and on the falling edge of the clock, it becomes zero that is the signal for ADC to start conversion from analog to digital. For conversion, the chip needs about 1.3 - 1.6 micro seconds, so I need to wait for this time and then I'll get the converted values. for this conversion time, I have counter that counts up to 35 that is between the given seconds range and when counter reaches 35, reading process is started. During the reading process, I need to send SCK clock to ADC, and also configuration bits(SDI). As I use channel 4, no sleep mode and unipolar type of conversion, with reference point to GND, my configuration bits are 6'b101010. On the falling edge of the SCK, ADC changes bits, and I should also provide configuration bits and on the positive edge of the SCK, ADC reads my configuration bits and at the same time, I read SDO bits sent from ADC. I use bits counter to count up to 12 to get 12 bits of channel data serially. When reading is finished, the collected 12 bits are sent to digit_at_position_decoder module, that takes the received 12-bit number, separates it into 4 digits and finds out their positions for seven segment display. Then, segment_decoder decodes digit segments and final output appears on the display.
After reading is finished, I begin next cycle of reading, so that when I change the analog values by potentiometer, the corresponding digital values also appear on the seven segment display.

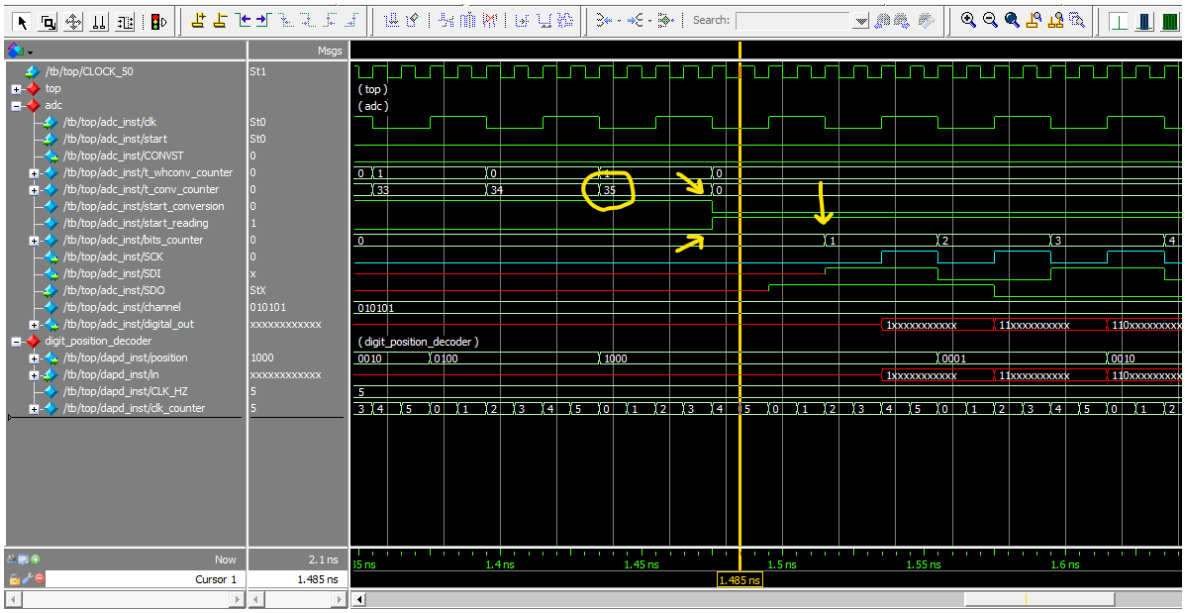
For simulation, I have module SDO_bits_sender, that just outputs hardcoded 12 bits one by one.

3 ModelSim Simulation

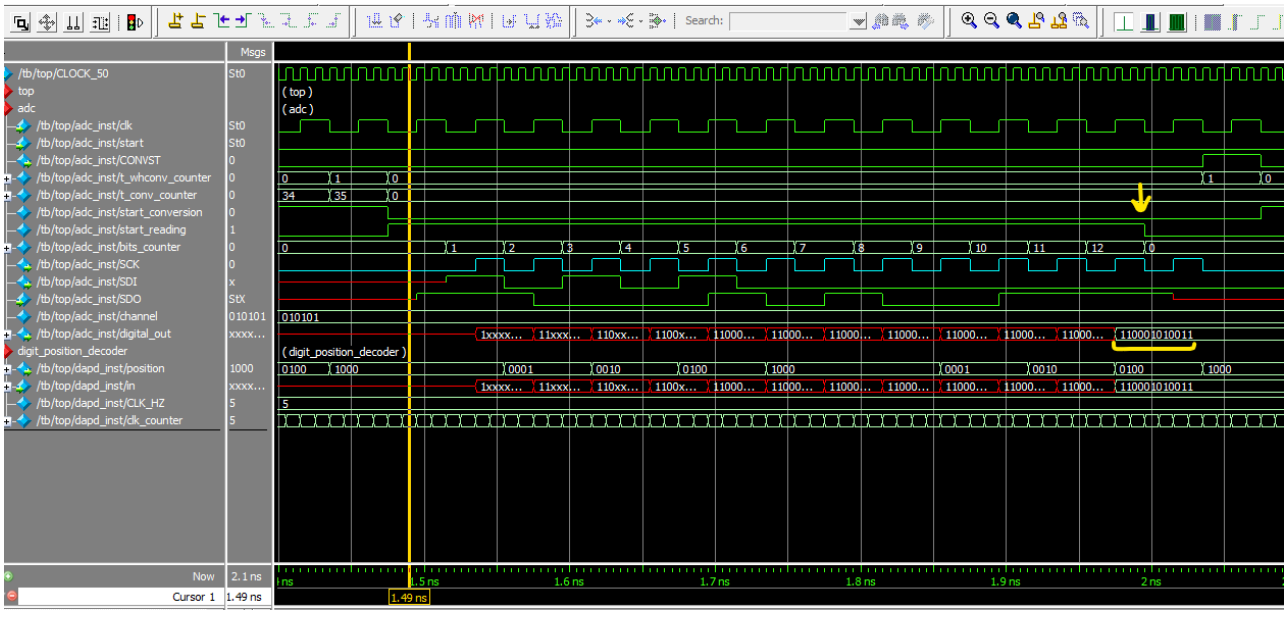
3.1 Send CONVST signal, start conversion



3.2 Send SCK, SDI Start Reading SDO



3.3 Final Output



4 Appendix

4.1 ADC

```
module ADC(
  input logic clk,
  input logic start,
  input  logic SDO,
  output logic SDI,
  output logic SCK,
  output logic CONVST,
  output logic [11:0] digital_out,
  output logic start_reading_SDO
);

/*
channel[0] => S/D = 1;
channel[1] => O/S = 0;
channel[2] => S1  = 1;
channel[3] => S0  = 0;
channel[4] => UNI = 1;
channel[5] => SLP = 0;
*/
localparam [5:0] channel = 6'b010101; // channel #4, unipolar, no sleep mode
localparam T_whconv = 1; // 20ns
localparam T_conv = 35;  // 1 / 1.3e-6 = 769,230.7692, 25mln / 769,230.7692

integer bits_counter = 0;
integer t_conv_counter = 0;
integer t_whconv_counter = 0;

reg clock_25_reg;
reg start_reading = 0;
reg start_conversion = 0;
reg next_reading_cycle = 0;

assign SCK = clk & clock_25_reg;

assign start_reading_SDO = start_reading; // used for simulation

always @(posedge clk) digital_out[12 - bits_counter] <= SDO;

always @(negedge clk)
begin
  clock_25_reg <= start_reading;
  if(start || next_reading_cycle) CONVST <= 1;

  // handling t_WHCONV, start conversion
  if((t_whconv_counter == T_whconv) && ~start_reading)
  begin
    CONVST <= 0;
    t_whconv_counter <= 0;
    start_conversion <= 1;
    next_reading_cycle <= 0;
  end
  else if(~start_reading) t_whconv_counter <= t_whconv_counter + 1;

  // handling t_CONV, start reading
  if(start_conversion)
  begin
    if(t_conv_counter == T_conv)
    begin
      start_reading <= 1;
      t_conv_counter <= 0;
      start_conversion <= 0;
    end
    else t_conv_counter <= t_conv_counter + 1;
  end

  // handling bits count, send config bits
  if(start_reading)
  begin
    if(bits_counter == 12)
```

```
    begin
      bits_counter <= 0;
      start_reading <= 0;
      next_reading_cycle <= 1;
    end
    else
    begin
      bits_counter <= bits_counter + 1;
      if(bits_counter < 6) SDI <= channel[bits_counter];
    end
  end
end

endmodule
```

4.2 ADC_bits_sender

```
// For simulation
// sends 12 bits as SDO
module SDO_bits_sender(
  input clk,
  input start_reading,
  output reg SDO_out
);

localparam [11:0] SDO_bits = 12'b110010100011;
integer i = 0;

always @(posedge clk)
begin
  if(start_reading)
  begin
    if(i < 12)
    begin
      i <= i + 1;
      SDO_out <= SDO_bits[i];
    end
    else i <= 0;
  end
  else SDO_out <= 1'bx;
end

endmodule
```

4.3 de0_nano_soc_baseline

```
module de0_nano_soc_baseline(
  input CLOCK_50,
  output ADC_CONVST,   // comment this for simulation
  output ADC_SCLK,     // comment this for simulation
  output ADC_SDI,      // comment this for simulation
  input ADC_SDO,       // comment this for simulation
  inout [35:0] GPIO_1, // comment this for simulation
  input [1:0] KEY,
  output [7:0] LED
);

reg start = 1;
reg [1:0] clock_25 = 0;

wire [11:0] digital_out;

wire [3:0] digitAtPosition;
wire [6:0] outsegment;
wire [3:0] position;

//wire ADC_SDO;                             // uncomment this for simultation
//wire start_reading;                         // uncomment this for simultation
//SDO_bits_sender bs(clock_25[1], start_reading, ADC_SDO); // uncomment this for simultation

ADC adc_inst(clock_25[1], start, ADC_SDO, ADC_SDI, ADC_SCLK, ADC_CONVST, digital_out, start_reading);
```

```
digit_at_position_decoder dapd_inst(CLOCK_50, digital_out, digitAtPosition, position);

segment_decoder sd_inst(CLOCK_50, digitAtPosition, outsegment);

assign GPIO_1[6:0] = outsegment[6:0]; // comment this for simultation
assign GPIO_1[35:32] = position;      // comment this for simultation

assign LED = digital_out[7:0];

always @(posedge CLOCK_50)
begin
  clock_25 <= clock_25 + 1;
  start <= 0;
end

endmodule
```

4.4 digit_at_position_decoder

```
module digit_at_position_decoder(
  input clk,
  input [11:0] in,
  output reg [3:0] digitAtPosition,
  output reg [3:0] position
);

localparam CLK_HZ = 50000000; // 5 for simulation
integer clk_counter = 0;

integer count = 0;

always @(posedge clk)
begin
  if(clk_counter == CLK_HZ / 1000) // 1 instead of 1000 for simulation
  begin
    case(count)
      0:
      begin
        digitAtPosition <= in % 10;
        position <= 4'b0001;
      end
      1:
      begin
        digitAtPosition <= ((in % 100) - (in % 10)) / 10;
        position <= 4'b0010;
      end
      2:
      begin
        digitAtPosition <= ((in % 1000) - (in % 100)) / 100;
        position <= 4'b0100;
      end
      3:
      begin
        digitAtPosition <= ((in % 10000) - (in % 1000)) / 1000;
        position <= 4'b1000;
      end
    endcase

    count <= count == 4 ? 0 : count + 1;
    clk_counter <= 0;
  end
  else clk_counter <= clk_counter + 1;
end

endmodule
```

4.5 segment_decoder

```
module segment_decoder(
  input clk,
  input [3:0] digit,
  output reg [6:0] outsegment
```

```
);

always @(posedge clk)
begin
  case(digit)
    0: outsegment <= 7'b1000000;
    1: outsegment <= 7'b1111001;
    2: outsegment <= 7'b0100100;
    3: outsegment <= 7'b0110000;
    4: outsegment <= 7'b0011001;
    5: outsegment <= 7'b0010010;
    6: outsegment <= 7'b0000010;
    7: outsegment <= 7'b1111000;
    8: outsegment <= 7'b0000000;
    9: outsegment <= 7'b0010000;
    default: outsegment <= 7'b1111111;
  endcase
end

endmodule
```

4.6 Testbench

```
module tb;

reg clk = 0;

always #5 clk <= ~clk;

de0_nano_soc_baseline top(clk);

endmodule
```