Design of a SPI receiver

(2010, SzP, LJ)

The goal of the lab is, to design a digital thermometer, which is connected to the FPGA through an SPI interface. The temperature should be displayed first on the LEDs, afterwards the 7-segment display will be used.

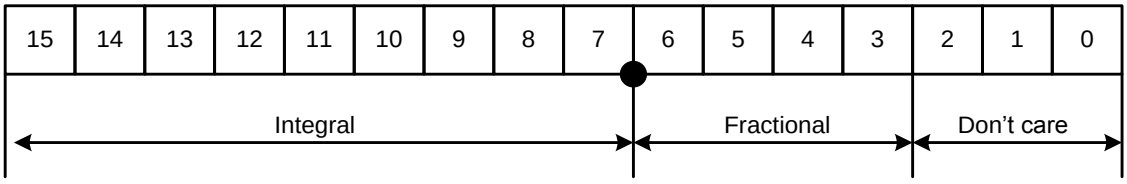
The "TMP121" digital temperature- used during this lab- meter is produced by Texas Instruments. Full datasheet is available on the Lab's website. This document summarizes only the relevant information.

TMP121 Data format

The serial data of the TMP121 consists of 12-bit plus sign temperature data (sum 13 bits) followed by three high-impedance (don't care) bits. (All together 16 bits) Data is transmitted in Binary Two's Complement format. Next figure describes the output data of the TMP121.

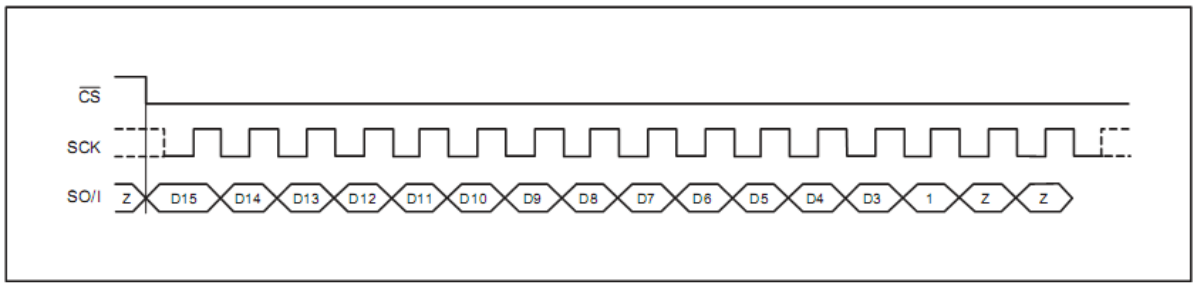
TEMPERATURE (°C)	DIGITAL OUTPUT(1) (BINARY)	HEX
150	0100 1011 0000 0000	4B00
125	0011 1110 1000 0000	3E80
25	0000 1100 1000 0000	0C80
0.0625	0000 0000 0000 1000	0008
0	0000 0000 0000 0000	0000
−0.0625	1111 1111 1111 1000	FFF8
−25	1111 0011 1000 0000	F380
−55	1110 0100 1000 0000	E480

It can be seen that the change in least significant bit, responds to 0.0625 °C temperature change. As 1/0.0625 = 16, the fractional part can be stored on 4 bits, while the integral part is (13-4=9) 9 bits in width.



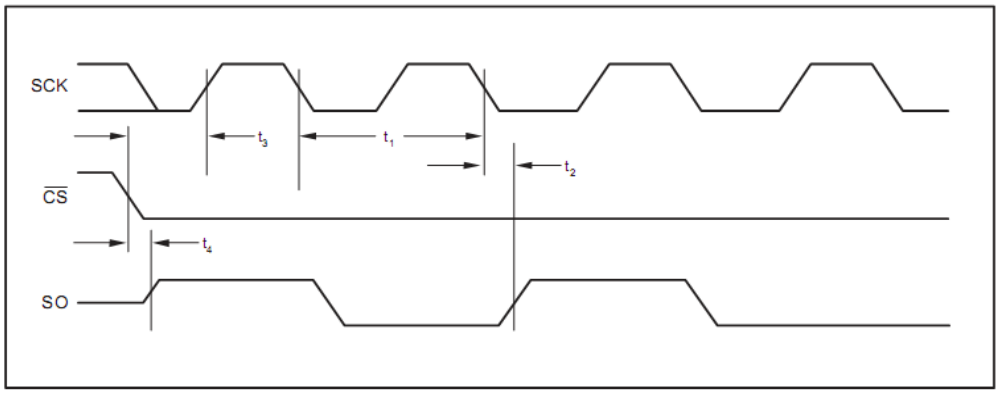
TMP121 SPI interface

The TMP121 uses SPI-compatible interface. Following figures describe the various timing requirements, with parameters defined in Table.



Transfer start is indicated by pulling the **Chip Select (CS)** low. Data is shifted out into wire **Serial Output (SO)** in the following 16 clock cycles (**SCK**). The 16 bit format is the one described above; the last 3 bits are don't care. The bits are shifted out on the **SCK falling-edge**.

The timing requirements (from the datasheet) as follows:



PARAMETER		MIN	MAX	UNITS
SCK Period	t_1	100		ns
SCK Falling Edge to Output Data Delay	t_2		30	ns
$\overline{CS}$ to Rising Edge SCK Set-Up Time	t_3	40		ns
$\overline{CS}$ to Output Data Delay	t_4		30	ns
$\overline{CS}$ Rising Edge to Output High Impedance	t_5		30	ns

As a summary:

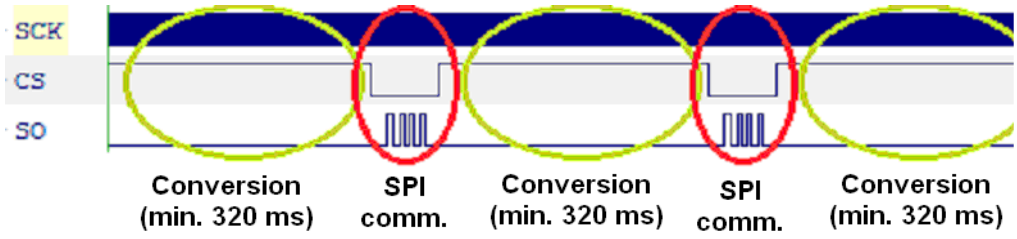
- **t1**: The period of SCK is minimum 100 ns. (Therefore the max. frequency is 10 MHz).
- **t2**: SCK Falling Edge to Output Data Delay is 30 ns. As a conclusion data should not be sampled earlier by the FPGA.
- **t3**: There must be a 40ns delay between the falling edge of CS, and the rising edge of SCK.
- **t4**: The data appears on the output after CS falling edge not later than 30 ns.
- **t5**: After the riding edge of CS, data line is driven active at maximum 30 ns.

After the evolution of the datasheet let us make some design consideration.

The main clock input for the FPGA is 50MHz. The SPI may work at maximum 10 MHZ (**t1**). In order to simplify the design, let us choose SPI clock speed equal to 1/8 of the main system clock (6.25 MHz, period 160 ns).

- Because of **t3** constraint, CS should be changed on the **SCK rising-edge**. (160 ns/2 > 40 ns).
- Because of **t2** constraint it is a good idea to sample the SO on the **SCK rising-edge.**

The datasheet explains that temperature conversion is performed while the CS is logic high. For continuous operation, the minimal time required for the conversion is 320 ms, so we have to ensure that CS is high at minimum 320 ms long.



The minimal time required for the conversion is 320 ms. The main clock is running on 50 MHz, with a period of 20 ns. The **minimal** cycles required are: 320ms/20ns = 16.000.000.
In order to simplify the design, we will use a 24 bit counter, with a bit longer period of (2^{24} * 20 ns= 16.777.216 * 20 ns) 335.5 ms.

SPI interface design

As mentioned above the SPI interface timing will be controlled by a 24 bit counter, counting from 0-up till (2^{24}-1) running from the main input 50 MHz clock. (Resulting in 335.5 ms period.) The SPI communication will be performed during some of the first clock cycles. Let us call this counter *"cntr".*

The SPI clock is running 8 times slower compared to main clock input. One can use the second bit of the cntr (cntr[2]), as the clock source for the SPI communication, while it produces 8 times slower clock. (*Remark: bit 0 (cntr[0]) would have 2 times slower, bit 1 would have 4 times slower etc…*)

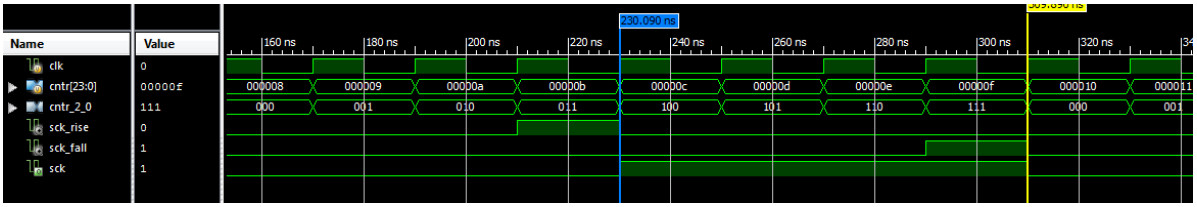
The generated SPI clock has a **rising edge** when:

- The main system clock has a rising edge **AND**
- *cntr[2:0]==3'b011*

Similar to this, the SPI clock has a **falling edge** when

- The main system clock has a rising edge **AND**
- *cntr[2:0]==3'b111*

This can be seen on the following figure: (We will need this two signals later…)



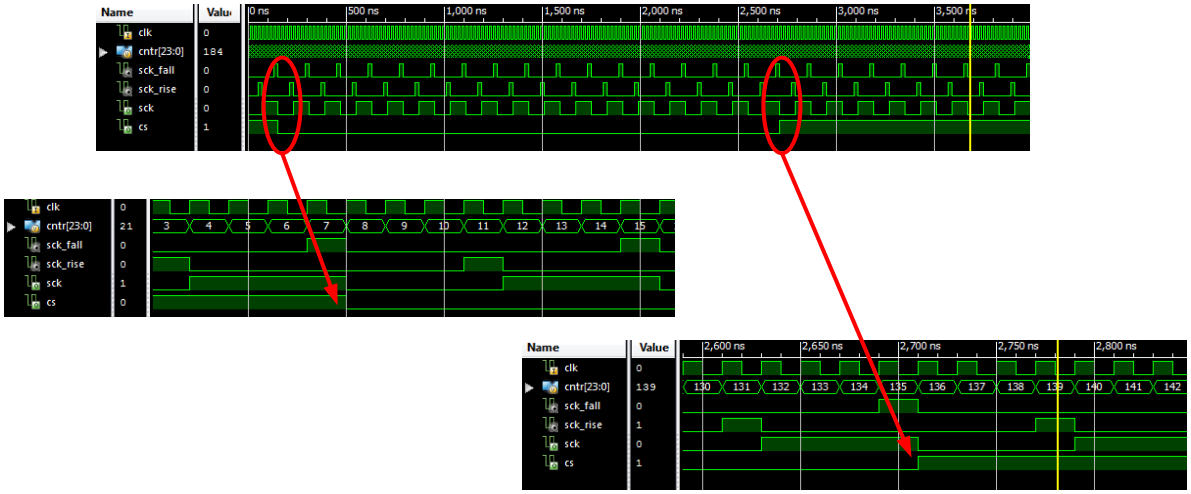
- *cntr[23:0]* is the above mentioned 24 bitcounter
- *Cntr_2_0* is this counter's last 3 bits.
- *sck_rise* and *sck_fall* is the decoded rising and falling edge. Please notice, that they are 1 clk pulse in length, and they align with the generated SCK signal.

(Remark: We could use the generated SCK signal as an input for other modules (like the data shift register) but it would result a non-synchronous design. So please forget this!!)

The full SPI communication takes 16 clock cycles, all together 16*8=128 main clock cycles.

The **Chip Select (CS)** should be driven low at the SCK falling edge, as concluded earlier. After 16 SPI clock cycles, it should return to logic high.

- The counter value is equal to 7 (the first falling edge…) at the moment when the CS is driven low is.
- The counter value is equal to 7+16*8=135 at the moment when the CS is driven high again.
- During the reaming clock cycles, the CS remain high, the TMP121 will generate the new temperature value.

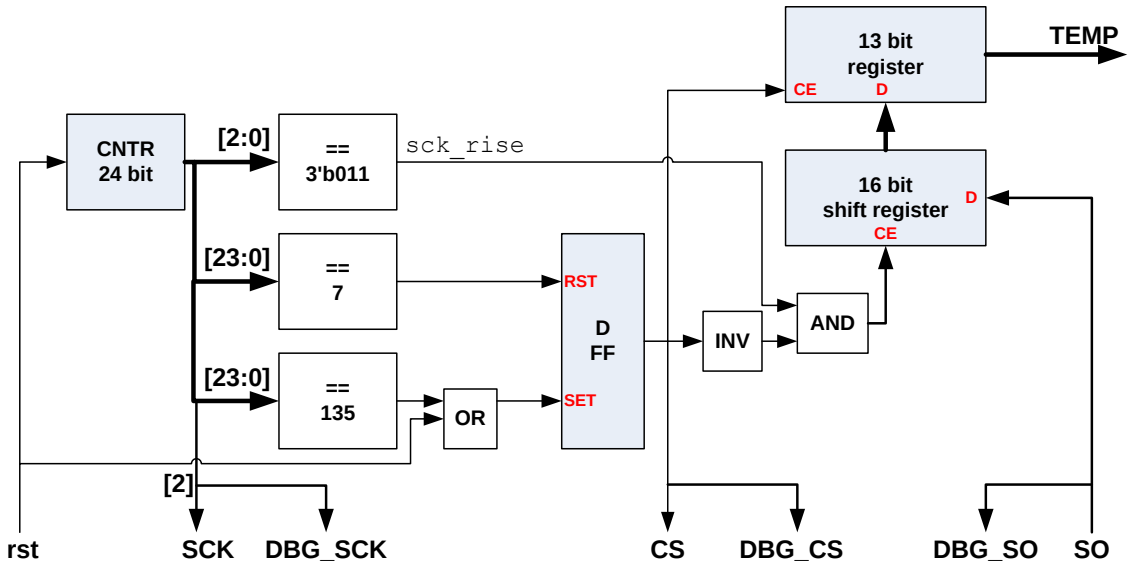


On the **Serial Ouput (SO)** data line the converted temperature value arrives in 16 SPI clock cycles. As concluded earlier data should be sampled on the rising edge.

The TMP121 outputs the **Most Significant Bit (MSB)** first, so we will need a **16 bit left-shifter**.
It is a good idea to copy the result into another register at the end of the conversion, in order to keep the last value until it is updated. (The shift register contains wrong temp values during the SPI communication.) This output register could update its value, when there is no ongoing SPI communication → When CS is in logic high.

The output register can be 13 bits in width, storing only the 13 temperature value. (Not storing the don't care bits.)

Based on the above considerations, the block diagram could be the following.



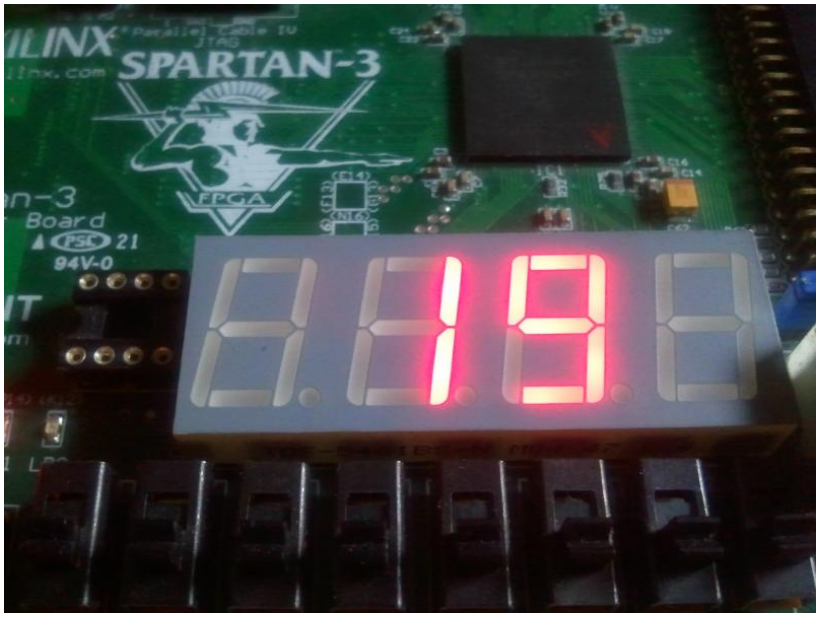
Notations:

- Grey boxes: Sequential logic. All FF are running from the main 50 MHz clock. (Not shown.)
- White boxes: Combinatorial logic
- Red texts: Inputs of the functional block
    - RST: FF reset input → clears the FF into logic low.
    - SET: set input → sets the FF into logic low.
    - CE: FF clock enable input.
    - D: FF data input.
- SCK: SPI clock output. (Output of the FPGA)
- CS: SPI Chip Select. (Output of the FPGA)
- SO: SPI Serial Data Output. (Input of the FPGA)
- Signals starting with "DBG_": They are debug signals, which will be connected to the logic analyzer.

Hexadecimal display

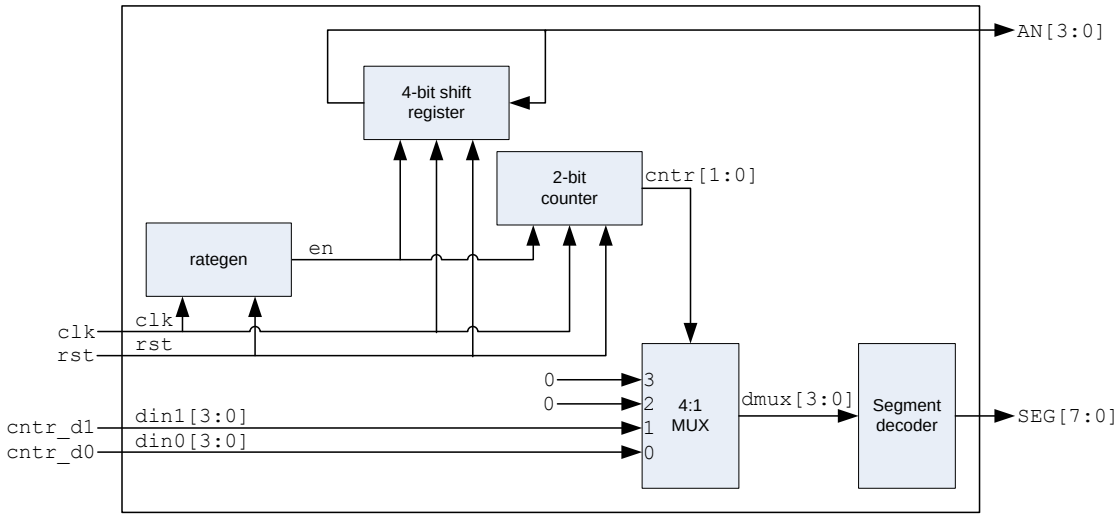
A 7-segment display will be used to display the **integer** part of the measured temperature.

- **Left digit:** Sign (- sign or nothing).
- **Middle two digits:** Integer part of the temperature in **hexadecimal** format.
- **Right digit:** Nothing



The temperature on the picture above is positive (the leftmost digit shows no negative sign), the measured temperature is 0x19 = 25 °C

Block diagram of the 7-segment display used on the 2. measurement lab, was:



This unit was designed to display only decimal characters (0..9), but currently we would like to display hexadecimal characters too (0x0…0xF). Additionally we would like display a ”negative-sign” or alternatively switch all segments off if needed. As a conclusion 18 different combinations can be displayed. (16 + “-“ + blank)
The module implemented on Lab 2. had 4 bit wide data inputs for all digits, but as 18 different combination is needed we have to extend it to 5 bits. (The display used had, 2 inputs only, it should be increased to four.)

The coding of the new 7-segment display should be:

- If the first bit (MSB) = 0, then the last 4 bits equal to the hex character.
- If the first bit (MSB) = 1,
    - **AND** last bit (LSB)= 0: Nothing on the display
    - **AND** last bit (LSB)= 1: Negative sign is on the display

(The “dot” next to the 7-segment display is handled separately. SEG_DP is assigned to 1, meaning switch off.)

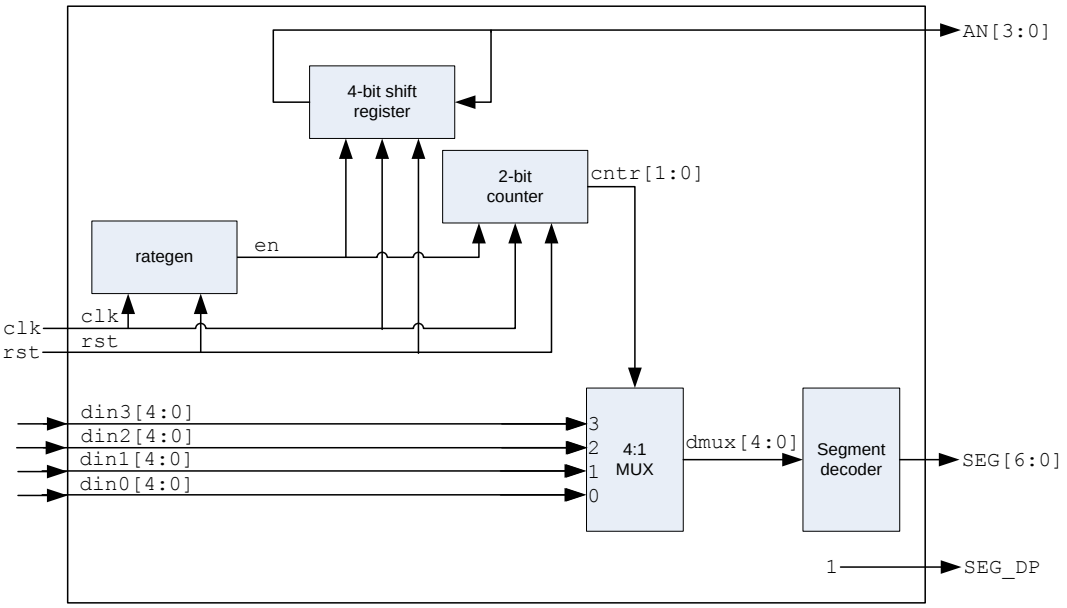
The new decoder code:

```
reg [6:0] SEG_DEC;
always @( * )
 case (dmux)
   5'b10001 : SEG_DEC <= 7'b0111111;   // -
   5'b00000 : SEG_DEC <= 7'b1000000;   // 0
   5'b00001 : SEG_DEC <= 7'b1111001;   // 1
   5'b00010 : SEG_DEC <= 7'b0100100;   // 2
   5'b00011 : SEG_DEC <= 7'b0110000;   // 3
   5'b00100 : SEG_DEC <= 7'b0011001;   // 4
   5'b00101 : SEG_DEC <= 7'b0010010;   // 5
   5'b00110 : SEG_DEC <= 7'b0000010;   // 6
   5'b00111 : SEG_DEC <= 7'b1111000;   // 7
   5'b01000 : SEG_DEC <= 7'b0000000;   // 8
   5'b01001 : SEG_DEC <= 7'b0010000;   // 9
   5'b01010 : SEG_DEC <= 7'b0001000;   // A
   5'b01011 : SEG_DEC <= 7'b0000011;   // b
   5'b01100 : SEG_DEC <= 7'b1000110;   // C
   5'b01101 : SEG_DEC <= 7'b0100001;   // d
   5'b01110 : SEG_DEC <= 7'b0000110;   // E
   5'b01111 : SEG_DEC <= 7'b0001110;   // F
   default  : SEG_DEC <= 7'b1111111;   // OFF
 endcase

assign SEG = SEG_DEC;

assign SEG_DP = 1’b1;
```

Based on it the new decoder block will look like this:



Measurement task

1. Implementation of the SPI interface. Displaying the integer part of the temperature on the LEDs.

1.1 Implementation of the SPI Interface

Implement the SPI interface in Verilog based of the previous assumptions and block diagram.
Let us call the module *spi_temp*, with the following interfaces:

- clk: input System 50 MHz clock input
- rst: input System reset, clears all FF into logic low.
- cs: output SPI Chip Select signal
- sck: output SPI Clock
- so: input SPI Data
- temp: output The measured temperature 13 bit in width (integer + fractional)
- DBG_CS, DBG_SCK, DBG_SO outputs: Connection to the logic analyzer.

1.2 Simulate the SPI interface

Create a new Verilog Test Fixture file in Xilinx ISE, which tests the module implemented in 1.1. Let us call this fixture: *tb_spi_temp*.
Implement the test fixture file, simulate the code, and make sure it runs correctly. You can use the following code blocks.

Modify the created *initial* block to the following.

```
initial begin
   clk = 1;
   rst = 1;
   #102 rst = 0;
end
```

Generate a 50 MHz test clock source. (#10 → 10ns, 10ns *2 =20 ns → 50MHz)

```
always #10 clk = ~clk;
```

The TMP121 will be simulated quite simply. We have to provide 16 bits during the 16 SCK clock cycles. The simulated temperature data will be stored in 16 bit wide register.
During the SPI transfer, the SCK clock cycles will be counted with a 4 bit wide counter. The counter will be reset to 15, on the start of the SPI transfer (Falling edge of CS), and count down on each SCK clock cycle. The data appearing on the SO wire is decoded by selecting one bit from the 16 bit wide register. The selection is done with the 4 bit counter. As the counter is counting downwards it is the MSB bit appearing on SO when the transfer starts.

Remark: *As this code is designed for simulation, it is allowed to write on variable (bit_cntr) in multiple always blocks. This is not allowed when writing code for synthesis.*

```
reg [15:0] spi_data = 16'h1234;
reg [ 3:0] bit_cntr = 15;
wire cs_dl;

assign #2 cs_dl = cs;

always @ (negedge cs)
   bit_cntr <= 15;

always @ (negedge sck)
if (~cs_dl)
   bit_cntr <= bit_cntr - 1;

assign so = spi_data[bit_cntr];
```

***Important*:** The test-fixture generated by the ISE defines *SO as reg*. The definition should be modified to *wire*.

Tasks

- Copy the Verilog source code into the Measurement Report file.
- Using the Waveform View **make sure the simulation was successful**, and the data appearing on the output *temp*, is equal to the expected value.
- Copy the simulation results into the Measurement Report file, and **explain** what is on the picture.

1.3 Implementation of the SPI Interface

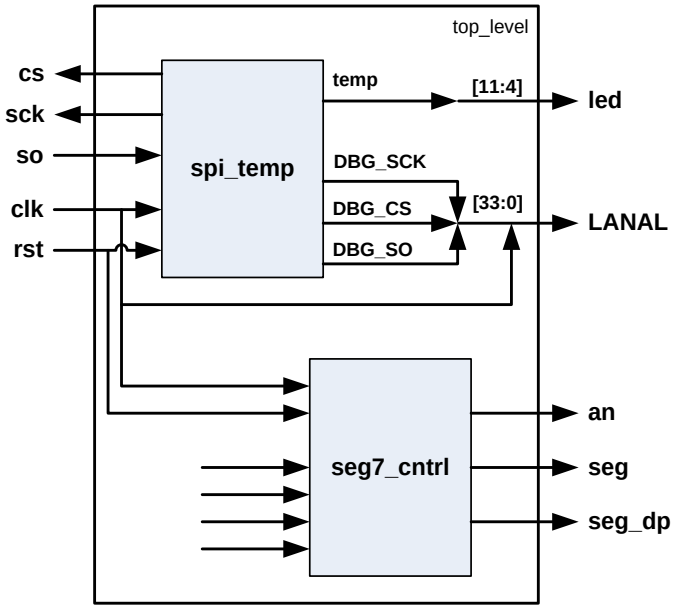
Download the specified ZIP file, which includes the following.

- *top_level.v*: The top level module, which instantiates the 7-segment display, as a submodule.
- *seg7_cntrl.v*: The 7-segment display
- *pins.ucf*: The pin assignment file
- *lanal.ucf*: The constraint file needed when the logic analyzer is used.

Interfaces of the *top_level* module are the following:

- clk: input System 50 MHz clock input
- rst: input System reset, clears all FF into logic low.
- cs: output SPI Chip Select signal
- sck: output SPI Clock
- so: input SPI Data
- led: output 8 bit LED drive
- an: output 4 bit digit select signal
- seg: output 7 bit segment drive
- seg_dp: Display "dot" output
- LANAL: 34 bit connection to the logic analyzer

- Instantiate the previously implemented *spi_temp* module in the *top_level* module. (Check the comments in the code, where it should be done !)
- Connect the ports with the corresponding wires. (If their names are the same.)
- Connect the 13 bit wide *temp* (of the *spi_temp* module*)* to the wire defined in the *top_level* module*.*
- Connect the *temp* [11:4] to the *top_level* 8 bit *led* output. (Drive the 8 LEDs with the bottom-integer part of the measured temperature).
- (At the moment do not care about the instantiated 7-segment display.)



Connect to the 34 bit LANAL output the following. Check comments in the code, where it should be done!)

- The first (MSB) bit should be main system clock.
- The 3 LSB bits should be the 3 debug wires coming from the SPI module
- The remaining 30 bit (in the middle) should be constant 0

Add the two UCF files to the design. Generate BIT file, and download it to the FPGA!
Check if the temperature appearing on the LEDs is ok. (It should be somewhere between 20-30)

Run the Logic analyzer program, and create a screen-shot where **only** the main system clock + the 3 SPI signals are displayed. Check a full SPI transfer!

Tasks

- Copy the Verilog source code into the Measurement Report file.
- Which mode of the logic analyzer you have used? Why?
- Which is the right trigger condition?
- With the proper settings, create a screen-shot where the SPI transfer can be seen. Make a screen-shot from the trigger setup too.

2. Displaying the temperature on the 7-segment display

After displaying the measured temperature on the LEDs, let us use the 7-segment display too.

2.1 Modifications of the 7-segment display

The downloaded 7-segment display contains the following modifications. (Compared to the one used on Lab. 2):

- Four data inputs for 4 digits.
- All inputs are 5 bits wide.
- Coding of the inputs has been changed. (As defined earlier).

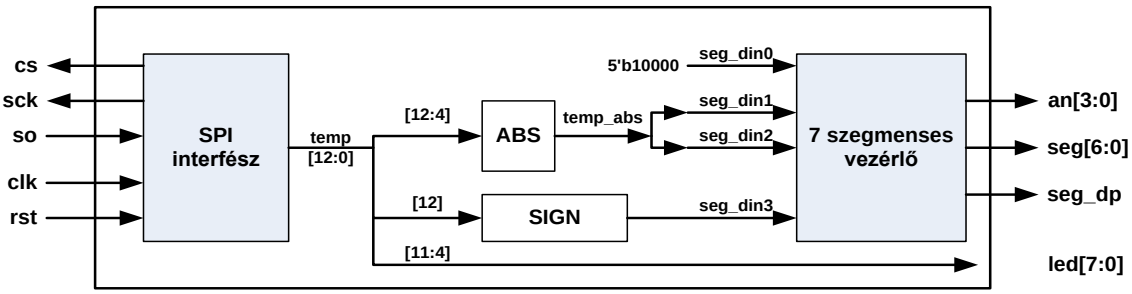
2.2 Driving the display

The *top_level* module from the ZIP file contains the instantiation of the new 7-segment display. Wires connecting to the data inputs are also defined (*seg_din0…seg_din3*), but the wires are not driven correctly. Please check the code, and try to understand!

The temperature arriving from the SPI interface is a 13 bit wide number in two's complement format. On the display we would like to display, the absolute value plus a sign separately, therefore it can not be connected directly.

- Port *din0* (rightmost digit on the board) is not used, so "nothing on the screen" (5'b10000) code should be assigned there.
- *Port din1, din2* should be connected with *seg_din1, seg_din2. Seg_din1, seg_din2* will hold the absolute value of the measured temperature, encoded in 5 bit encode format. As the 7-segment display controller now accepts 5 bit input, when generating *seg_din1* an additional '0' should be concatenated to the "left" (MSB) followed by *temp_abs* four LSB **integral** bits. When *Seg_din2* is built 0 should be concatenated with *temp_abs* four MSB **integral** bits.
- Port *din3* input should be connected with *seg_din3.* 5'b10000 should be assigned to *seg_din3* if the temperature is positive and 5'b10001 if it is negative.

The new block diagram is the following:



The ABS and SIGN blocks are responsible to generate the above mentioned functions. **They should be implemented in the TOP module, do not generate sub modules for these functions.**

Tasks

- What is the period of the 7-segment display?
- Check the 7-segment display code. On which value of the *clk_div* will be the *en* logic high?

- Generate the *temp_abs* signal, which responds to absolute value of the temperature.
- Connect the right values to *seg_din0…seg_din3. Remark: These wires are 5 bit wide. Check coding (the character appearing on the display based on the input) as defined above.*
- Implement the design, Configure the FPGA. Check if the temperature appearing on the display is ok. (Heat up the temperature meter a bit, and notice the changes on the display.)
- Compare the values shown by the LEDs with the display.

Questions

(This you should know before coming to the lecture)

1. Which serial protocol needs clock to be transmitted?
2. Why it is hard to transfer some Mbit/s using UART transfer?
3. Which serial protocol doesn't support bus topology?
4. How to control the four digit 7-segment display?
6. What kind of setup parameters the users have, when configuring a serial port on the PC?
7. We are using a 115200 baud UART connection, with 8N1 setup. (8 bit data, no parity, 1 STOP bit). How long does it take to transfer 100 bytes?
8. If 115200 baud, 8N1 UART is used what could be the difference of a single bit time measured on transmitter and receiver, if successful transmission is needed?
9. Compare serial protocols based on their connection speed!
11. Which are the four different SPI modes?
13. What is START bit on UART protocol? What is the polarity?
14. What does odd parity means? What would be the parity value transmitted when the data is 0xAB (HEX)?
15. How many wires do we need to interface 4 SPI peripherals?