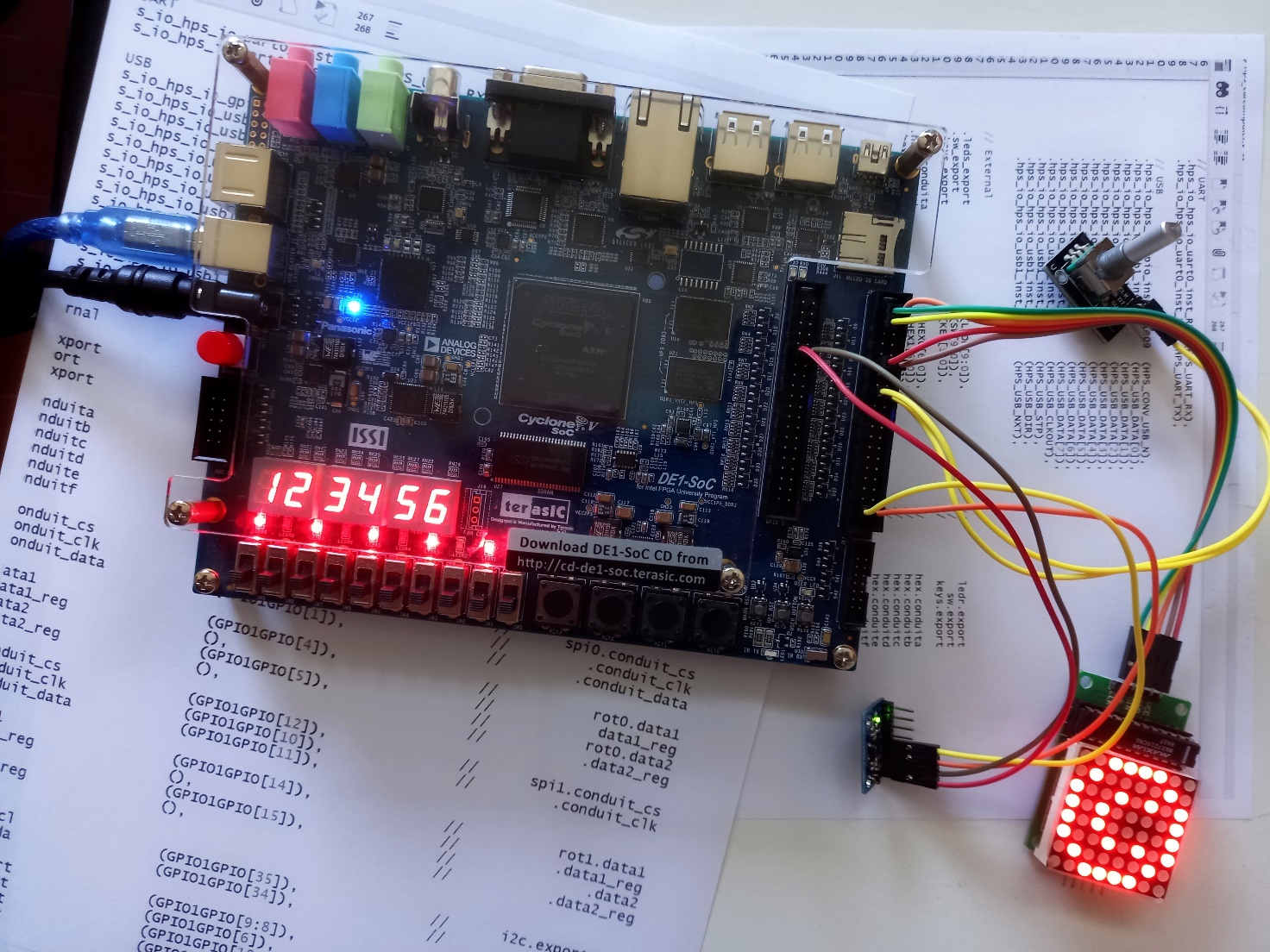
Electronic Systems Design

# Prof. Marsi Stefano - University of Trieste Academic Year 2025/26

Tutorial 3



**Design of a complete hierarchical system.**  
**Hardware** used: Terasic DE1-SoC Board  
**Software** used: Quartus 22.1, System Builder

Tutorial 3

Realization of a simple DE1-SoC "frequency meter"

Description: We want to create a device that, given a digital input signal, is able, by comparing it with a reference signal, to establish the duration of its period and its "duty cycle", displaying this information on the seven-segment display available on DE1-SoC.

Purpose: To develop a complete hierarchical system composed of several synchronous blocks.   
To Familiarize with the peripherals available on the board and their interfaces.

To analyze some tools available to force and monitor signals inside the FPGA.

Expected learning:

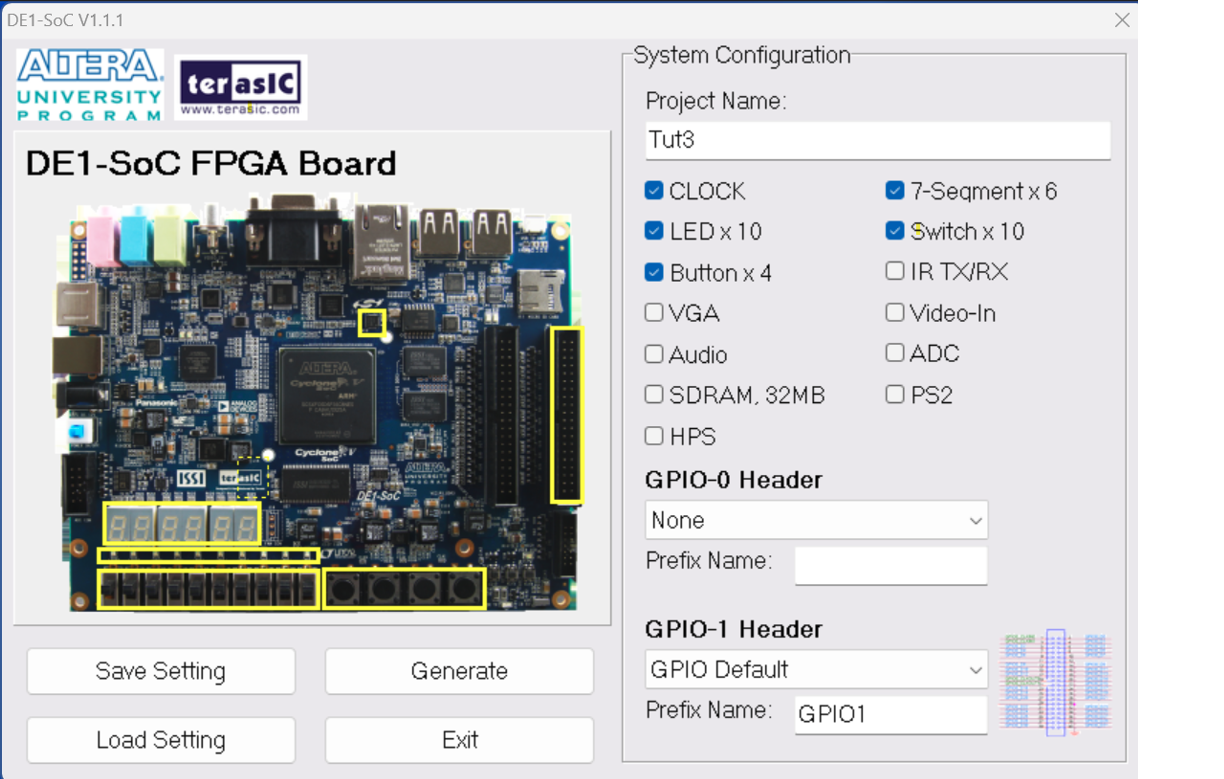
* Access to GPIO interface signals.
* Development of a hierarchical system using different input tools.
* Deep dive into VerilogHDL code
* To familiarize with the "In System Source & Probe" tool to check and observe internal signals.

# Procedure

#### Introductory phase

The company that produces the development board (Terasic) has developed a tool (System Builder), useful for easily starting a project. This software, based on the required periphals, creates the skeleton of a project to be developed later on Quartus, accompanied by constraint files both for the pinout and for the definition of clock constraints.

* run System Builder
* check on the peripherals you intend to use   
  in particular (LED / SWITCHES / BUTTONS / 7-Segment x6 / CLOCK )
* Decide to use GPIO1 in Default mode and assign it a reference name



>> Generate

* Save the project in an appropriate directory

#### Definition of the system architecture

* Open the newly generated project with Quartus

The project already has a VerilogHDL file that represents the "TOP Level Entity" of the final project and a .SDC (Synopsys Design Constraints) which contains the various design constraints. These Files will be used later.

The entire system could be described with the following specifications:

* The signal to be analyzed is processed to establish the instant in which the rising edge and the falling edge occur, generating pulses at these instants
* These pulses can be used to reset an appropriate counter to establish how many clock pulses have elapsed between the various edges.
* This count, to be easily visualized, should be performed in a decimal base
* The numerical value must be thus converted (digit by digit) into the signal with which to illuminate the different LEDs to display with the corresponding character.
* Furthermore, it is correct that the display changes its status ONLY when the count is completed and that it does not continuously show the evolution of the counter, therefore the pulses that determine the arrival of a front will also be used to "freeze" the value to be shown on the display till the arrival of the next front.

The system we want to create should therefore contain the following modules:

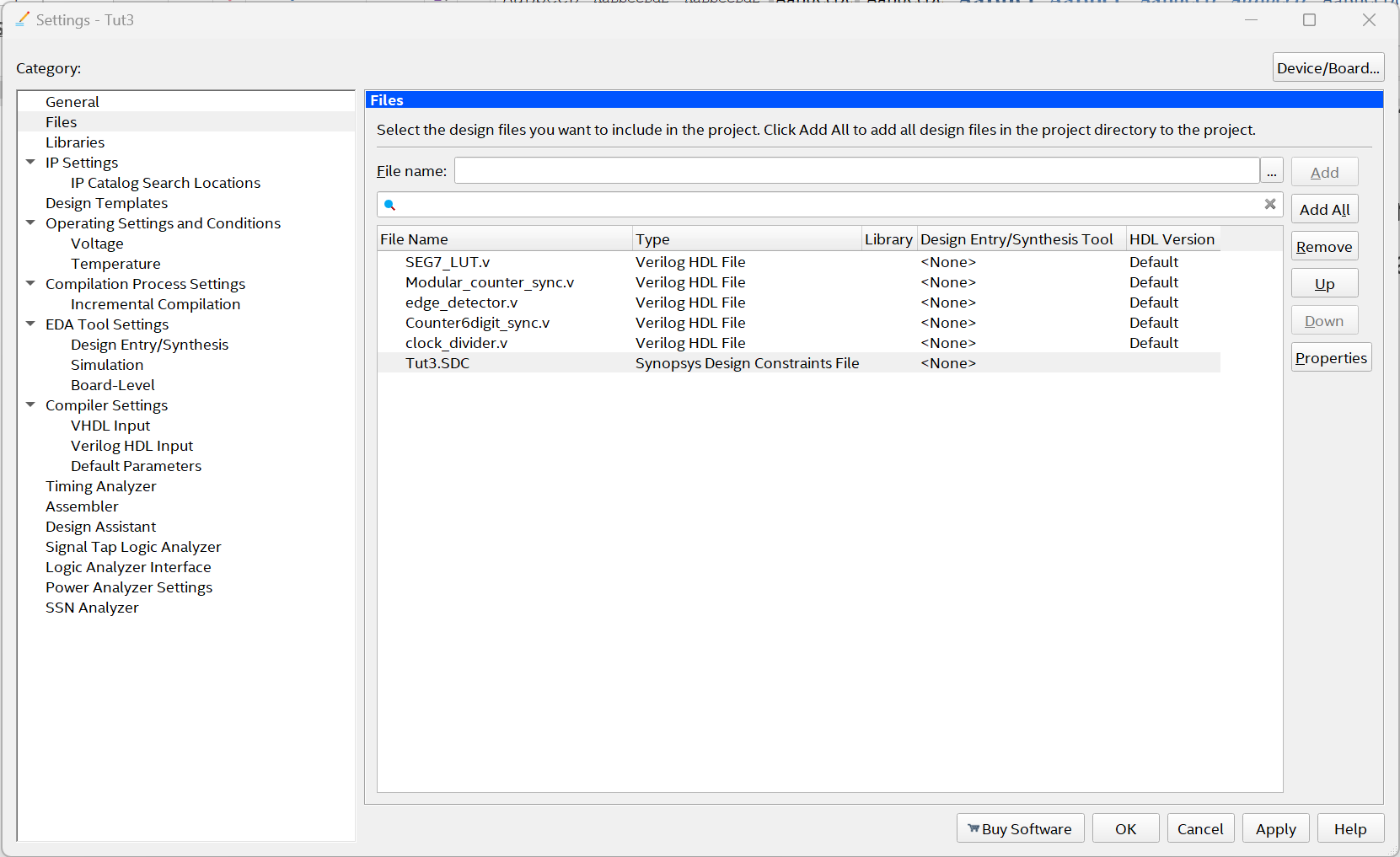
* A system designed to recognize the rising and falling edges of a signal
* A 6-digit BCD counter (to take advantage of the entire display) equipped with a synchronous reset that resets the count at each pulse on it.
  + 6 modules counting in binary between 0 to 9 (BCD)
* Six BCD\_to\_7SEG converters to drive the 6 displays capable of storing the data and altering it only in the presence of an appropriate control signal
* In addition, to make the first tests in the absence of an external signal, a signal generator will be created capable of creating a periodic signal where both the frequency and the duty-cycle can be controlled.

The files for the living modules are available on the course website[]:

Then import the following files into the project:

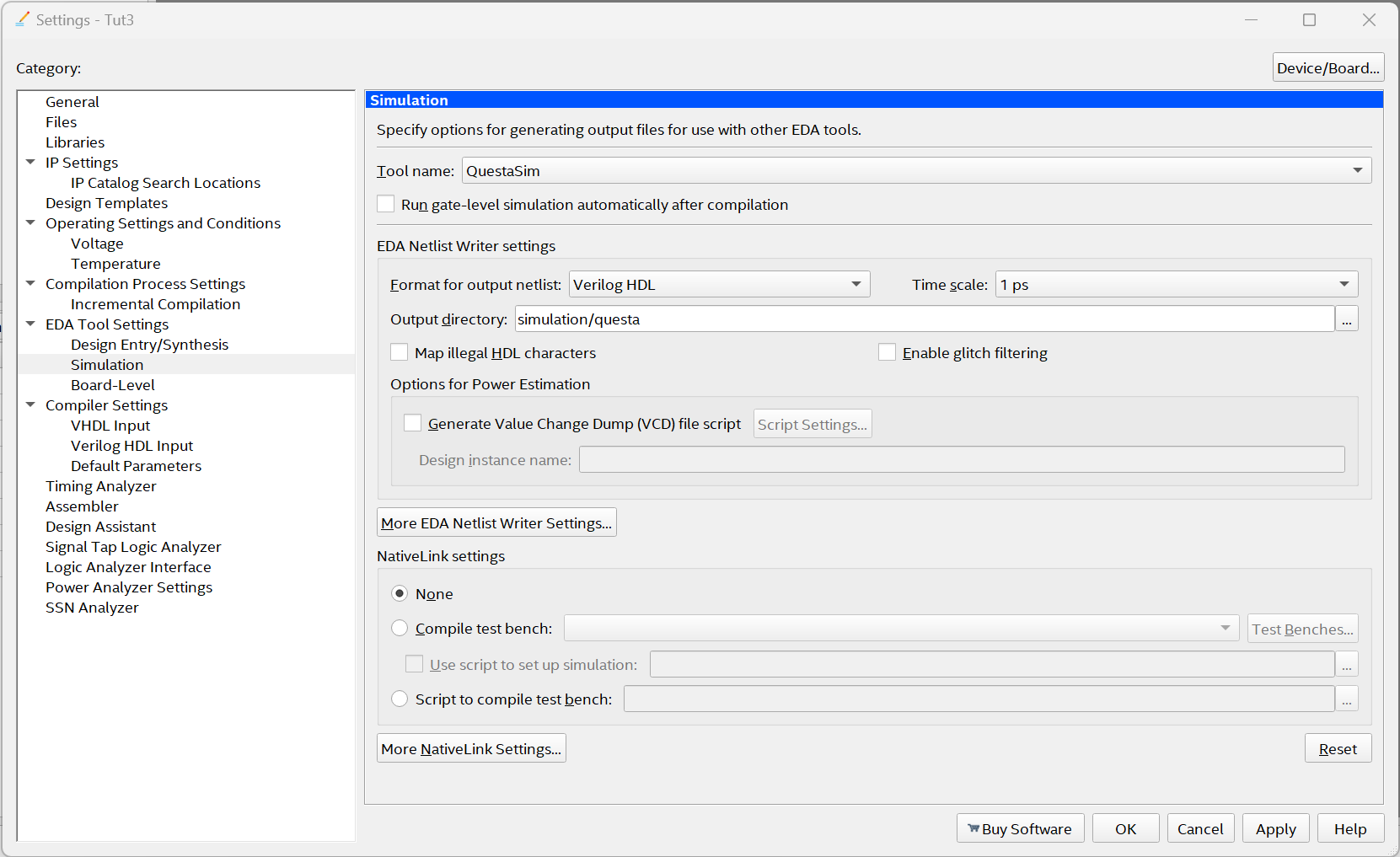
> Project > Add/Remove Files in Project ...

And import the files respectively as shown in the figure.



#### Functional simulation of individual blocks

Configure the simulator you intend to use   
> Assignament > Settings (Ctl-Shift-E)

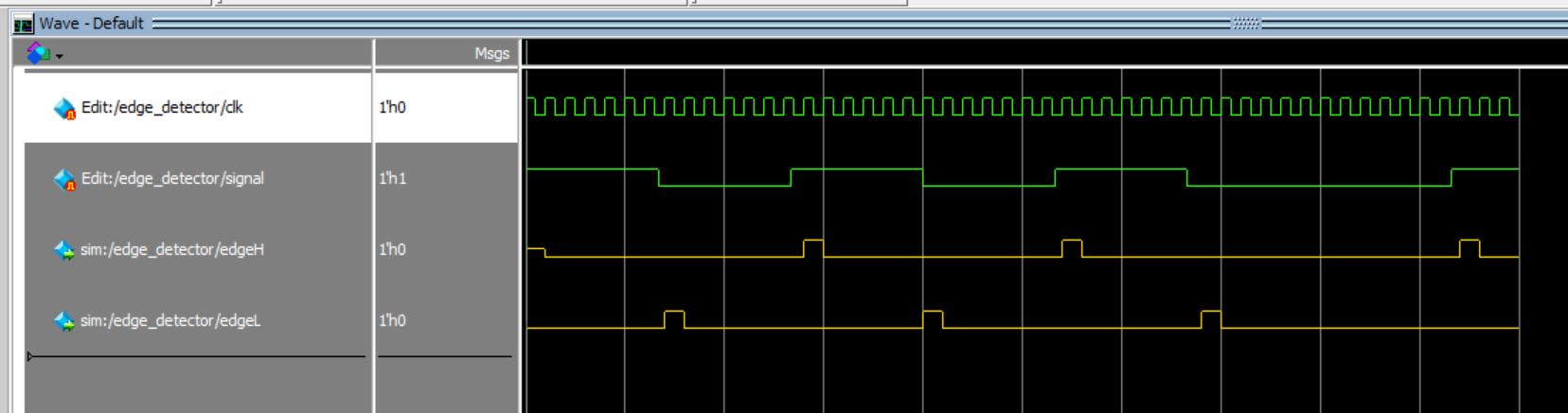


And under Simulation define simulator, parameters, options mode ...

* By setting (temporarily) the various imported modules as "Top Level Entity" (Ctl-Shift-V)
* Start Analysis & Synthesis (Ctrl-K)
* Tools > Run Rimulation Tool > RTL Simulation

We then move on to simulate the various modules of interest by providing appropriate input waveforms and analyzing the results provided by them, for example as follows:

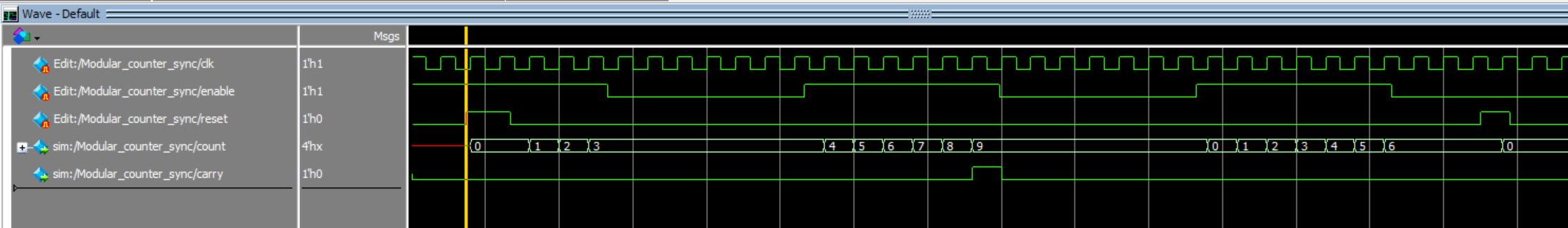
##### Edge Detector



Note in particular:

* The presence of output pulses when input presents an edge
  + But delayed until the first rising edge of the clock
  + Duration of exactly one clock period

##### Modular Counter Sync

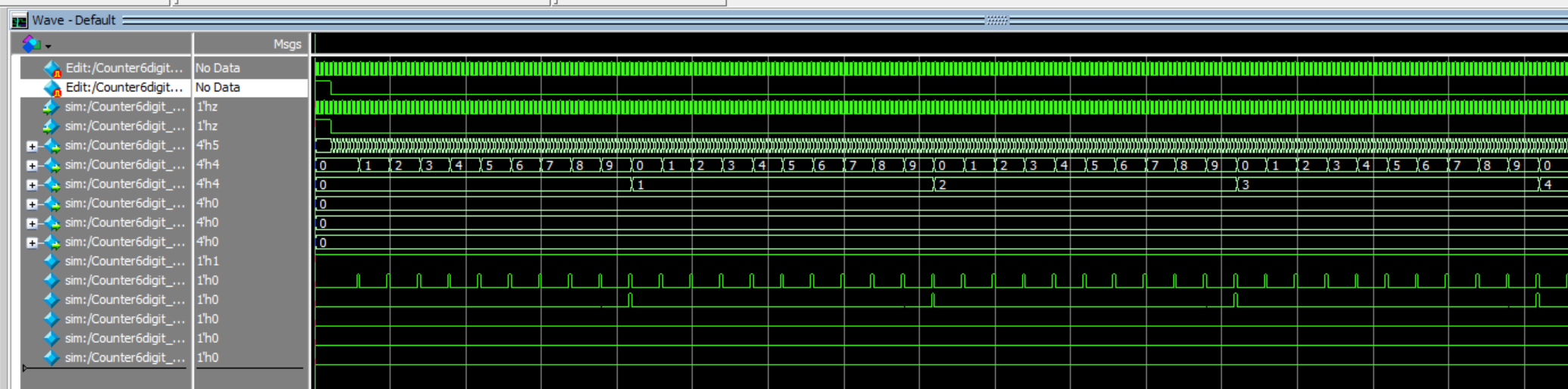


Note in particular:

* The use of reset and enable signals
* The generation of a pulse on the Carry signal when the maximum value has been reached, so that such signal can be used to progress (enable) the count to the next digit.
* The count is in BCD format (from 0 to 9)

##### Counter\_6digit\_sync

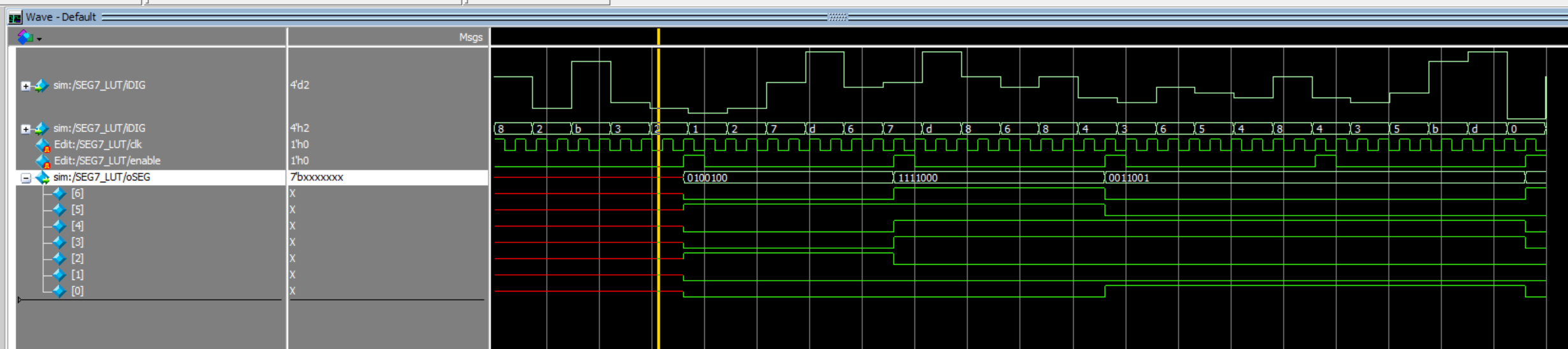




Note in particular

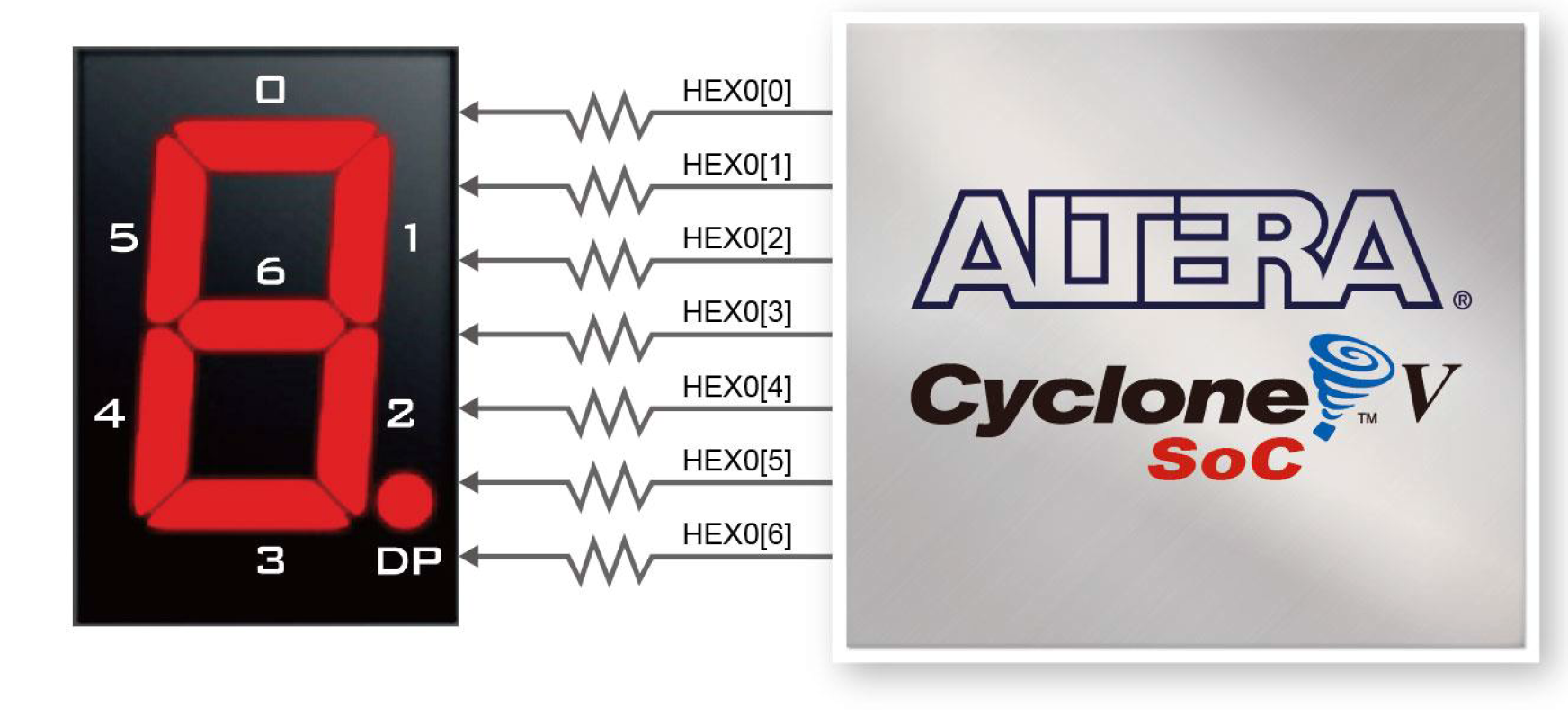
* How an Enable signal presented at the input causes the output to increase only at the next step
* The importance that all the Enable signals on the various digits occur even at different frequencies but all at the **same time** (i.e. when **all** the previous DIGITS are set to 9) Hence the importance of NOT synchronizing the generation of this signal on the clock.

##### SEG7\_LUT

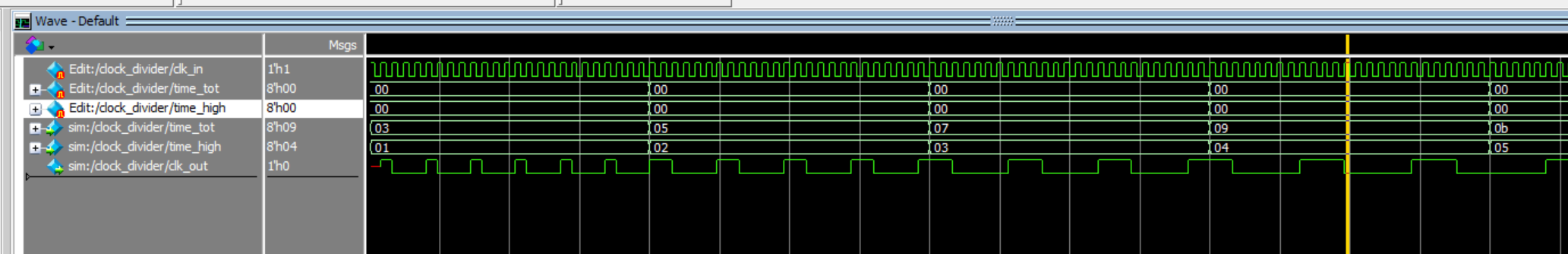


Note in particular

* The congruity of the figure to be displayed with the LEDs that light up   
  ex. N=2🡪 0100100 = 2 LEDs (HEXn[2] and HEXn[5]) **OFF** **(1)** and the others 5 **ON (0)**   
  ex. N=7 🡪 1111000 = 3 LEDs **ON (0)** HEXn[0..2] and the others 4 **OFF** **(1)**
* The storage of the output result only in the presence of Enable=1.



##### Clock\_divider



Note in particular

* How through the two input signals it is possible to modify the output signal both in frequency and as a duty-cycle.

#### Blocks Connections

Eventually all the defined blocks are connected together within a final system respecting the planned structure of the circuit. This can be done in two different ways: through a schematic or through a VerilogHDL description. For purely didactic reasons, we will see both solutions hierarchically.

##### Using schematic enter

A system should be created that includes the 6-digit counter developed above with as many BCDto7SEG converters suitable for correctly showing the numbers on the display.

Before proceeding, you need to create a "symbol" for **each** of the elements to be included in the schematic (in our case for *Counter6digit\_sync.v* and *SEG7\_LUT.v* )

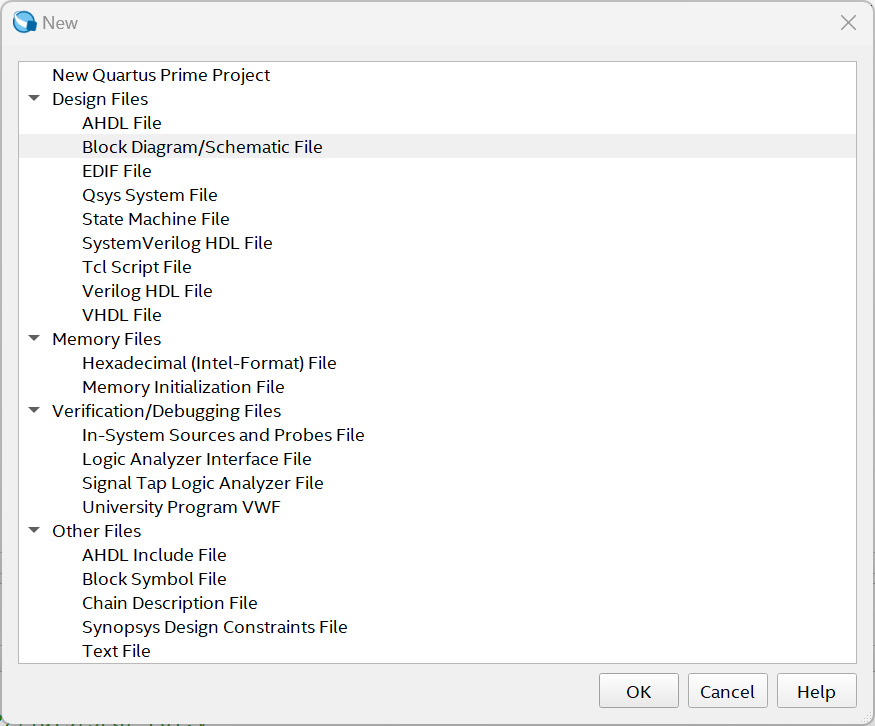
* Inside the "project Navigator" window
* Select the "Files" display mode
* Right-click on the selected file
* Create Symbol File for current File

Or

> File > Create/Update > Create Symbol File for current File

Next, once you have created the necessary symbols

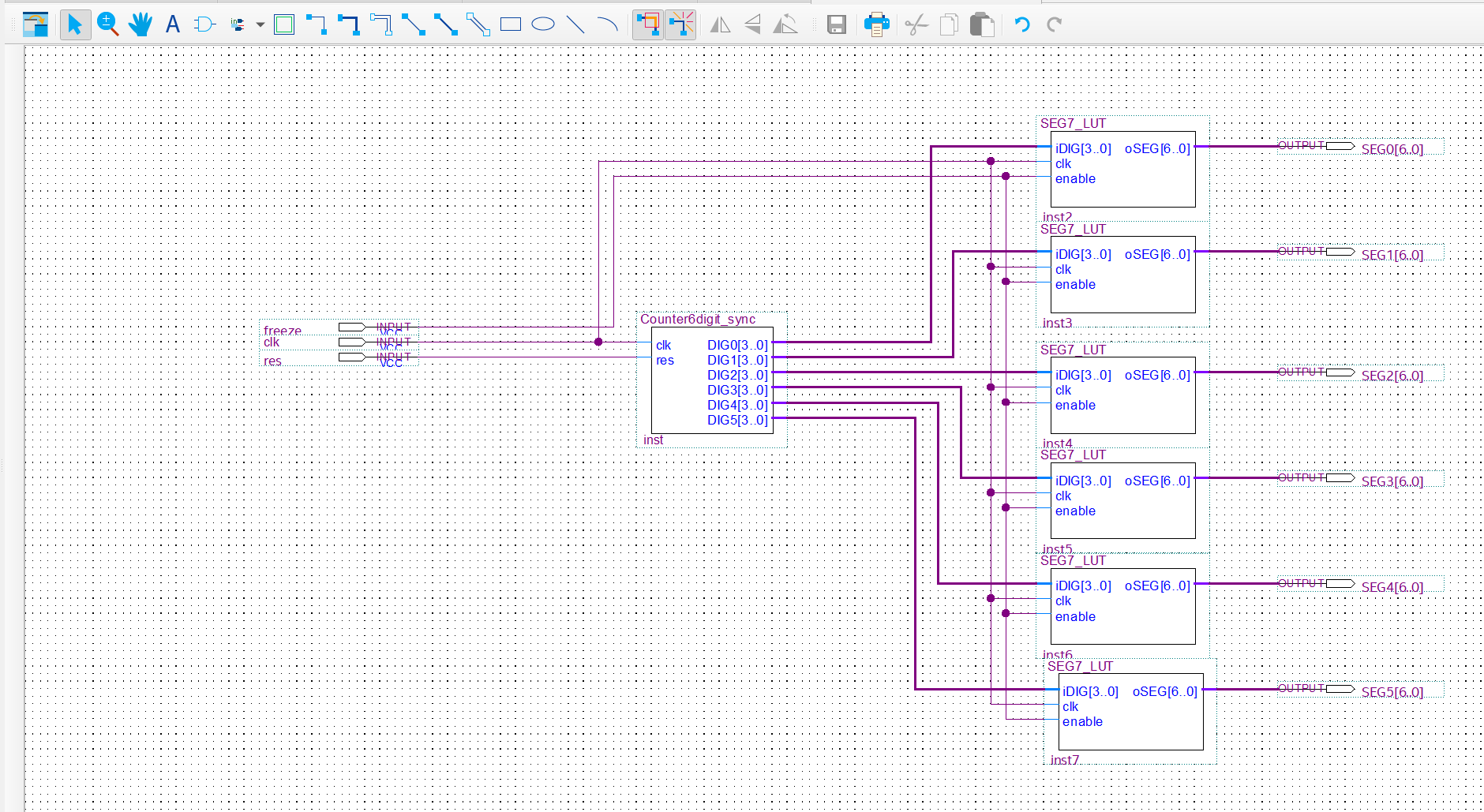
> >New File



> Block Diagram /Schematic File

<OK>

With the Graphic System available, draw a scheme similar to the following:



* Including the blocks generated in the previous step
* By connecting them appropriately (pay attention to the difference between bit connections (thin) and bus connection (BOLD) )
* Adding I/O pins
* Providing them with a correct and appropriately sized name especially in the case of BUSes

Save the schematic with an appropriate name.

Although the schematic thus created is perfectly integrated into Quartus, if you want to simulate or to integrate it into higher level VerilogHDL files, you are forced to generate an alternative version in VerilogHDL.

After highlighting the file in the "Project Navigator" window

> File > Create / Update > Create HDL Files from Current File

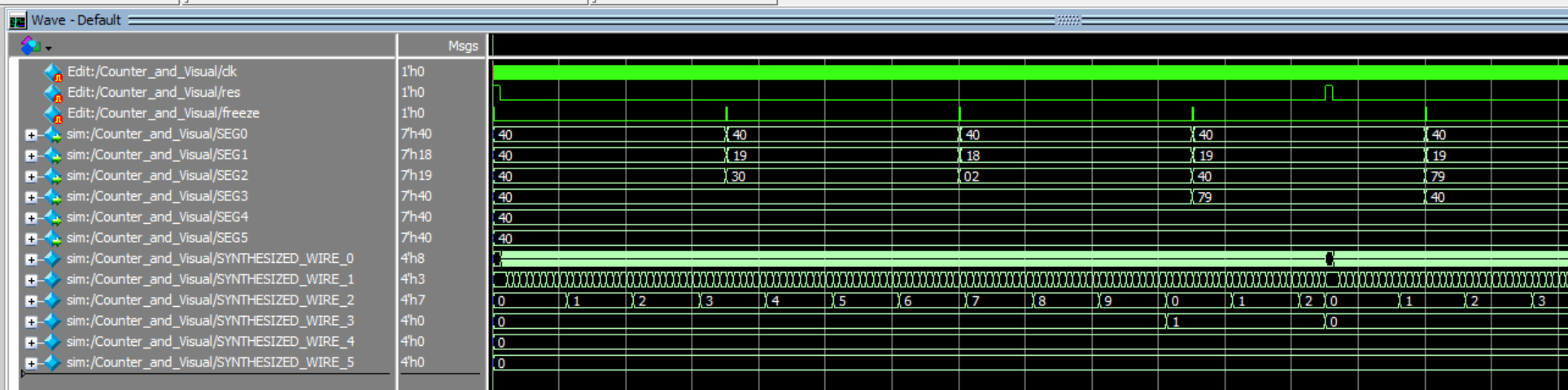
Choosing an appropriate name and the correct language (Verilog HDL) in the window that follows

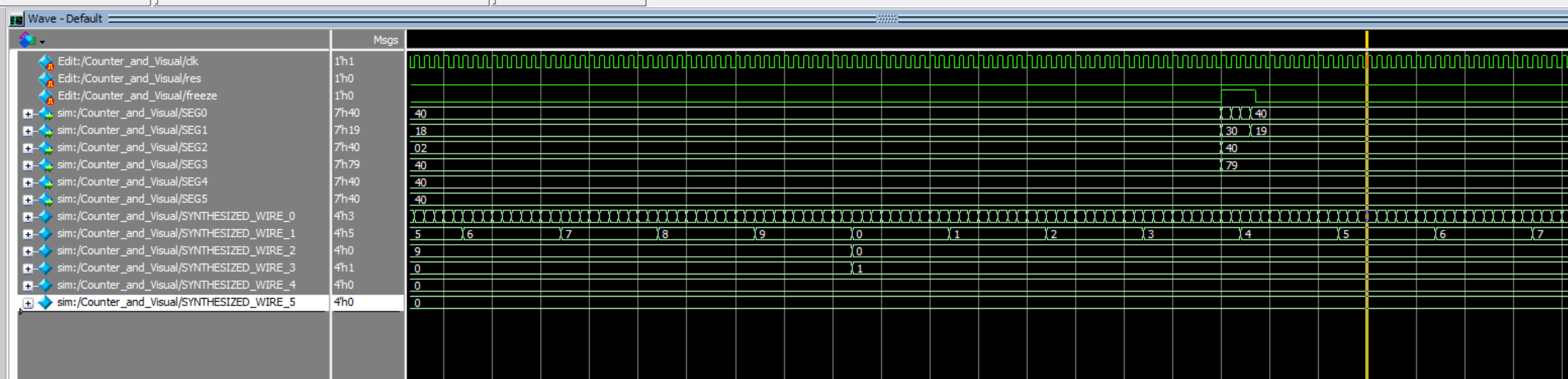


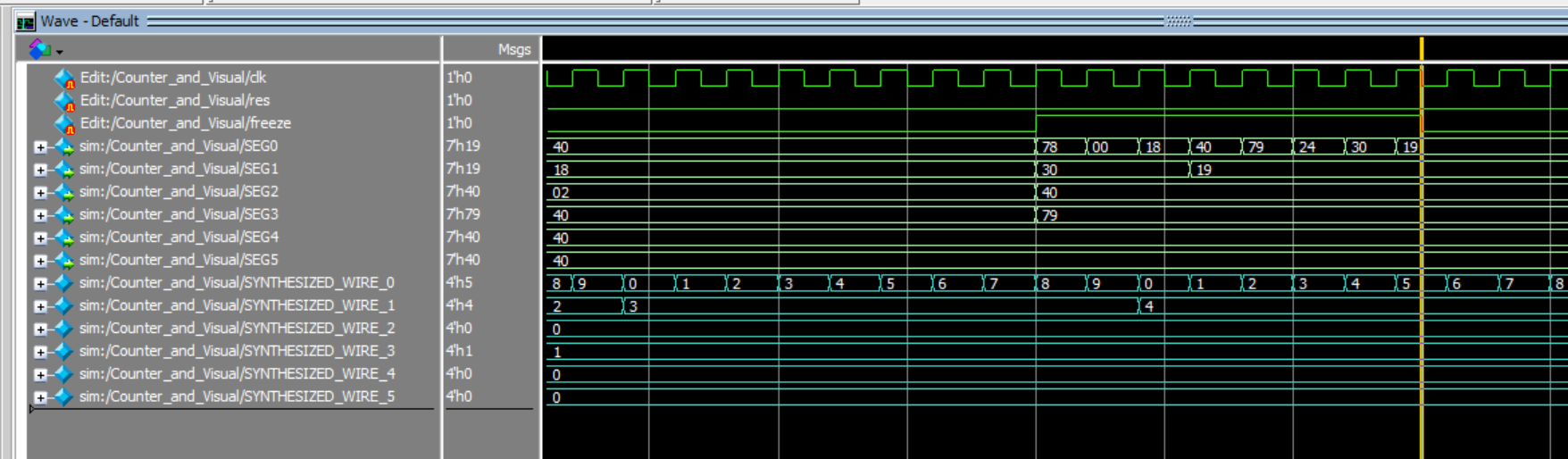
<OK>

To simulate this system, however, it is now necessary

1. Include the newly generated VerilogHDL file in your project
2. Remove the .bdf (schematic) file from the project – to avoid duplication
3. Set this element as "Top Level Entity"
4. Analyze & Synthesis
5. Launch RTL simulation
6. Within Questa/Modelsim provide the desired inputs
7. Analyze the results





 From them we can deduce:

* That the count proceeds correctly (999 🡪 1000)
* That the display/update of the digits takes place only when the activation signal (freeze) is HIGH and if it is maintained to High Level the output are updated every cycle.
* That the display on the LEDs is correct (001044 🡪 0x40,0x40,0x79,0x40,0x19,0x19) – note that it is a **7 bit** sexagesimal representation!
* That the displays are updated with one clock cycle delay with respect to the input data

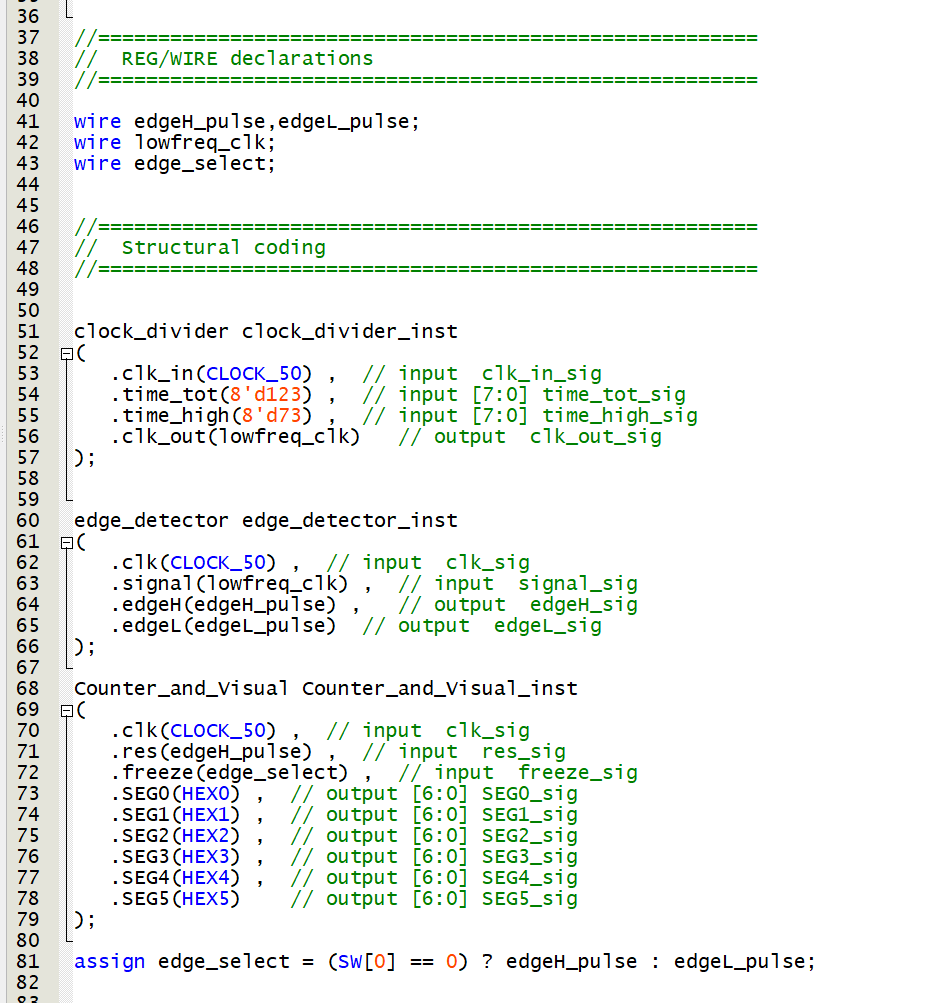
##### Using Structural Description in VerilogHDL

In the previous step, the use of the "schematic editor" was basically used to generate a VerilogHDL file that represent the connections among various blocks (have a look to the generated Verilog File).

Sometimes however it could be much easier to generate a structural file directly in Verilog HDL.

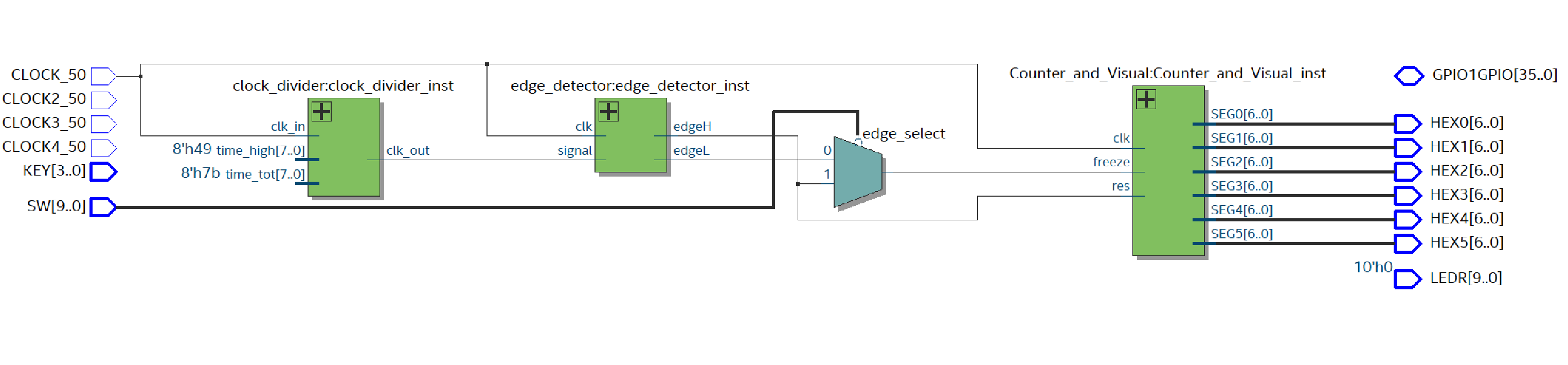
In generating the final file, complete with all the blocks, we will follow this way:

* In the Project Navigator Window, set the view mode to Files
* Define the file that has the name of the initial project (i.e. the one generated by "System Builder") as "Top Level Entity"
* Now switch to the Hierarchy display mode
* Double-clicking on the "Top Entity" opens a Verilog file (generated by "System Builder") which represents a skeleton for the final project (with interface signals and definition of the "TOP module".
* We then move on to define the internal signals and the structural description of the complete system
* To do this it can be useful for the various blocks that make up the project to generate an example of instantiation (In File mode – highlight the file of interest – File > Create/Update > Create Verilog Instanstation ...
* A file named <filename>\_inst.v appears in the project directory that provides an example of how to integrate the module.
* Repeat the procedure for all the blocks of interest
* Include these "instances" in the final file by appropriately modifying the input and output signals and creating the appropriate connections.
* Make the final file which as far as the structural part is concerned could be more or less the following:



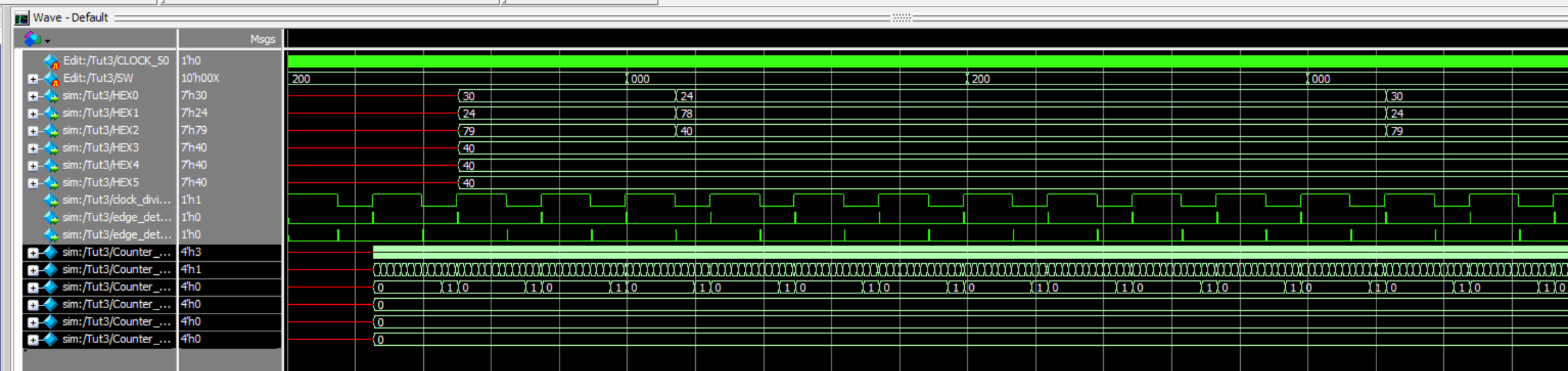
Note in particular

* The definition of signals
* The "clock divider" module is driven by 2 constant values: 123 and 73
* The instantiation of the three modules
* The connection between the modules through signals
* The  *assign* (on line 81) that allows (based on the input on SW[0]) to select which signal to use to update/view the count
  + If SW[0] ==0 the positive edge of the reference signal will be used (and since this is also used as a reset of the count, the number of a50MH pulses contained in an entire period will be indicated.
  + If SW[0] ==1 the negative edge of the reference signal will be used (and since the positive edge is also used as a reset of the count, the number of a50MH pulses contained between the two edges will be indicated (useful for estimating the duty\_cycle)

Eventually the RTL view can be useful to examine if the block are correctly connected:  


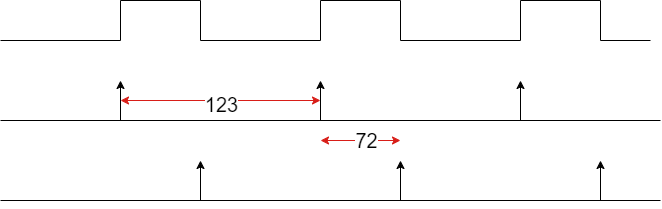
#### Final Simulation

As in the previous steps, it is suggested to simulate the entire circuit to verify its fundamental aspects.



#### Download on board

At this point, you can proceed to configure the DE1-SoC board and check its operation. In particular, it will be noted that based on the position assumed by SW[0] on the display, two values will be shown that correspond to the number of 50MHz pulses contained between the two sampling moments (whether they are respectively the rising edges of the low frequency clock or the rising and falling edges)



#### Controllability and observability

So far, the values for the signal to be measured have been made fixed and constant through the two assignments:

.time\_tot(8'd123) , input [7:0] time\_tot\_sig

.time\_high(8'd73) , input [7:0] time\_high\_sig

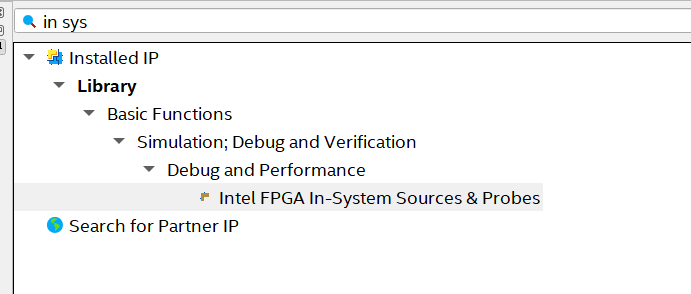
And the only way to modify them is to modify the verilog code and recompile the entire project. Alternatively, external signals (SW, KEY, GPIO) could be used to control them, but first of all these signals are limited, or alternatively they would require some external driving systems.

However, it is possible to interact with the board signals directly within the Quartus system, forcing certain signals to the desired value or even reading the values assumed by certain signals.

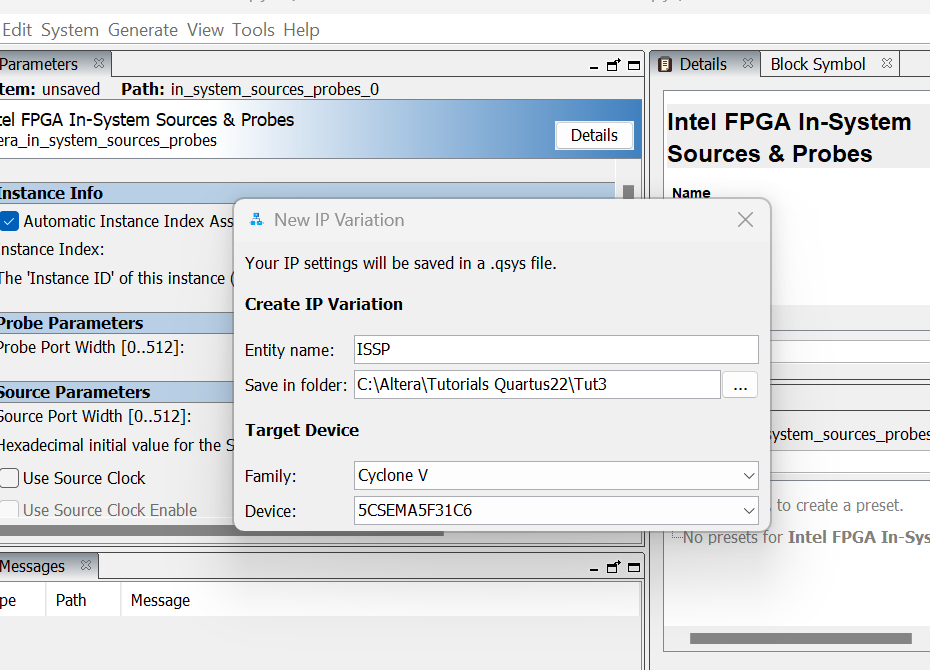
The tool that enables this interaction is the "***In System Source & Probe Editor" (ISSP)***”. To use it, however, additional hardware must be added inside the FPGA fabric to control and observe these signals.   
Such hardware is available among the modules developed by "third parties"

> Tools > IP Catalog

Within the window that is added to the Quartus layout windows, search for "Intel FPGA In-System Source & Probes" (you can help with the search line



(Double click)



In the window that appears, provide a mnemonic name for the entity you are going to create (in this case ISSP)

<Enter>

Now assign the number of signals you want to use as a "Source" (16 or 8+8 in our case) and the Number of signals to be used as a "Probe" (at the moment 1 is enough, for example to monitor the status of the switch)



<Generate HDL>

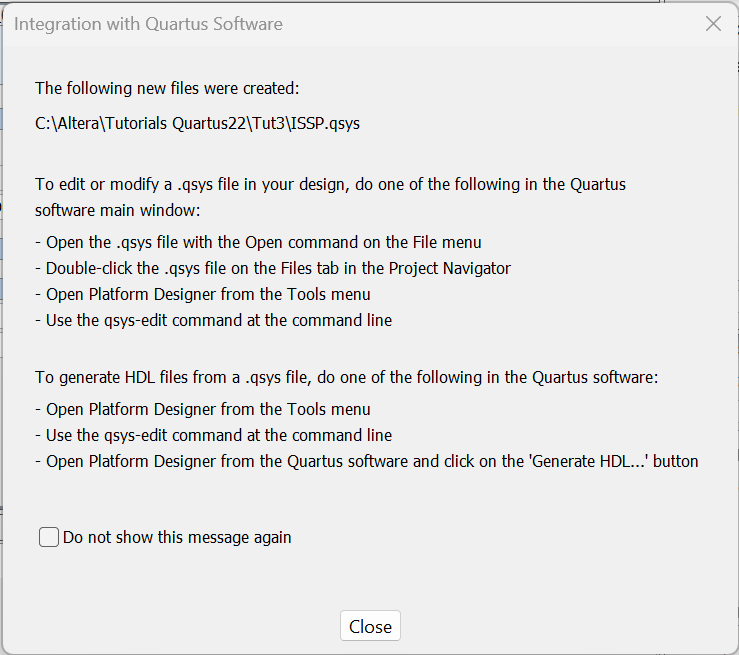
<Generate>

<Close>

<Close>

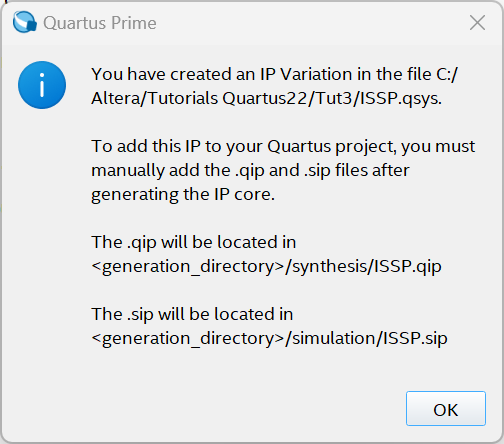
<Finish>

**Take carefully note** of what the system suggests:



<Close>

**Take carefully note** of what the system suggests:



<OK>

So as suggested: "manually" add the newly generated block among the elements making up the complete system:

> Project > Add/Remove Files in Project

Add the newly generated ISSP.qip file to the System Files.

<OK>

In addition, in the directory in which all the various blocks and elements related to this block have been generated, there is also a file called ISSP\_inst.v that can be used as an example for the instantiation of the newly formed block in our project:

ISSP u0 (

.source (<connected-to-source>), // sources.source

.probe (<connected-to-probe>) // probes.probe

);

Therefore, it is suggested to follow this example to integrate the block into our project, making sure that the "source" signals provided by ISSP connect to the signals driving the "clock divider" while the Probe signals connect to SW[0].

For example, add the instantiation of the newly generated ISSP module to the Verilog code for the "Top Level Entity" block

ISSP u0 (

.source ({t\_tot,t\_high}), // sources.source

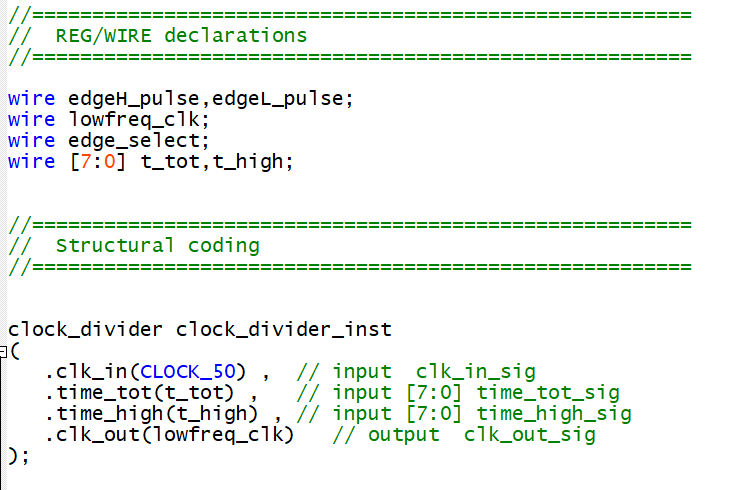
.probe (SW[0]) // probes.probe

);

After having opportunely

- defined the two new signals t\_tot and t\_high 8-bit !

- connected them to the "clock divider" device



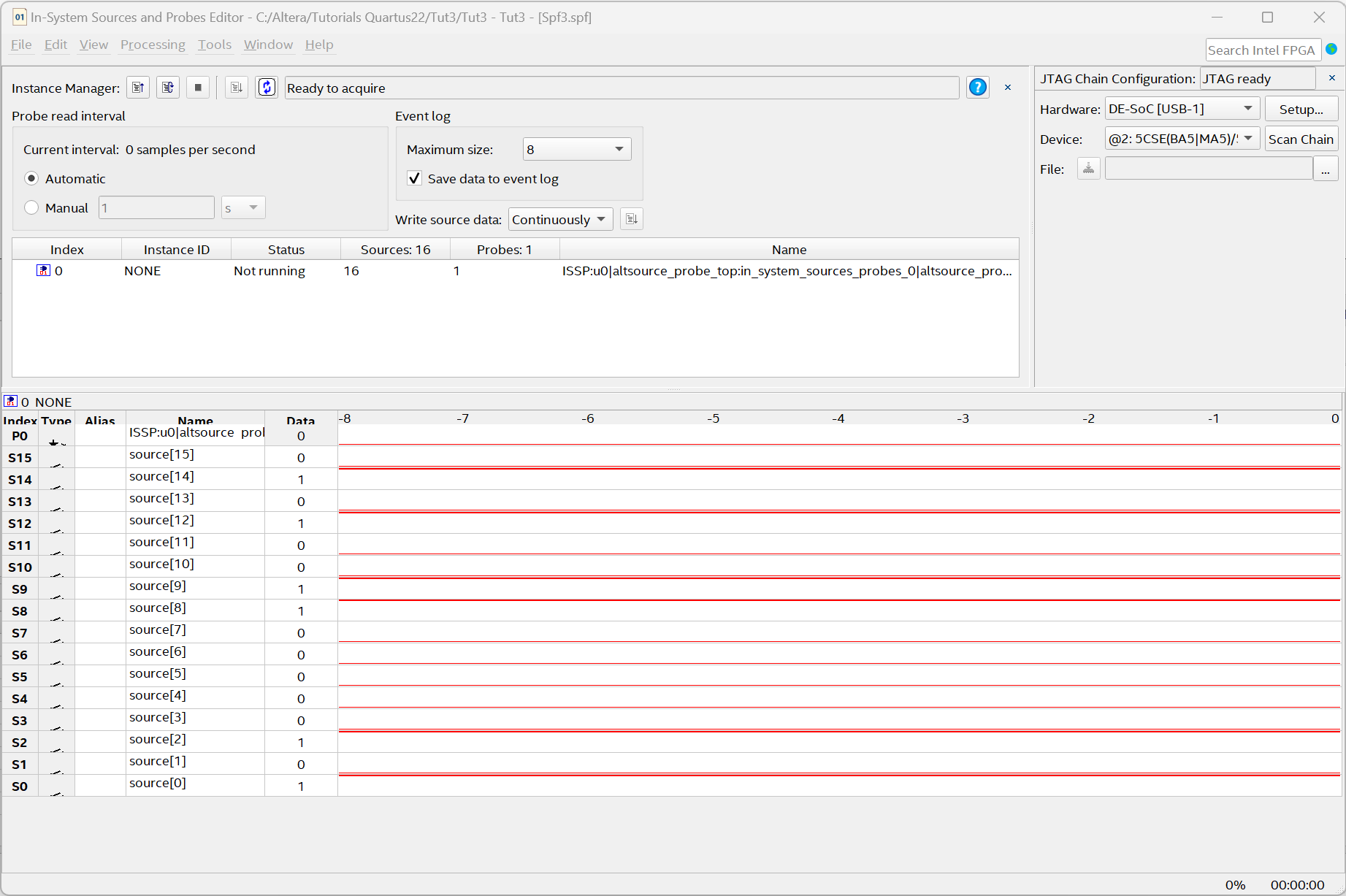
Recompile the complete system and download it to the card. It is also suggested to analyze the various "reports" and the occupation of resources before and after the inclusion of the ISSP block.

At this point, to interact with the signals inside the FPGA, you need to use a specific tool: "In System Source & Probe Editor".

> Tools > In System Source & Probe Editor

In the window that opens, activate the data acquisition:

> Processing > Continuosly Read Probe Data (F6)



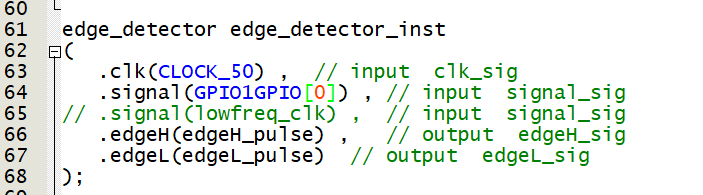
Change the data as desired and check the result on the card:

Note in particular: - that the P0 bit is pure acquisition and displays the status of the SW[0]- that the S[15:8] bits define the period of the signal to be measured.- that the S[7:0] bits define the period for which the signal to be measured is high (and that the last one must be obviously lower than the previous one for correct operation)

#### Real Signal

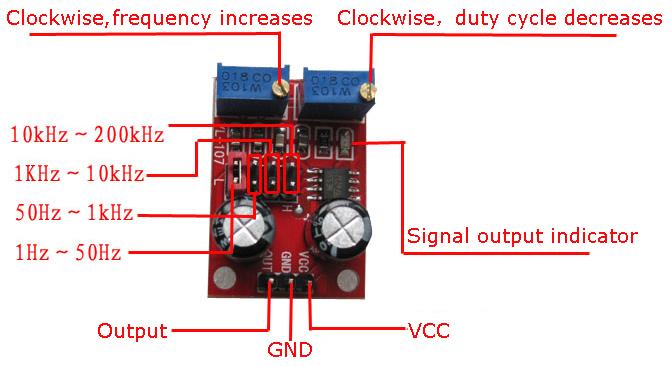
As a last step, try now to interface the circuit with a real signal. A low-frequency signal can be generated through an external device. And connect this through one of the I/O ports (GPIO1GPIO[0]).

Modify the system so that the edge detector block uses this signal as input:



The rest of the circuit can remain unchanged, although some blocks such as "ISSP" or "clock divider" are obviously redundant and therefore can be eliminated from both the project and the "top Level Entity" if desired.

The circuit shown in the figure based on the NE555 integrated chip can be used for generating the digital periodic signal.

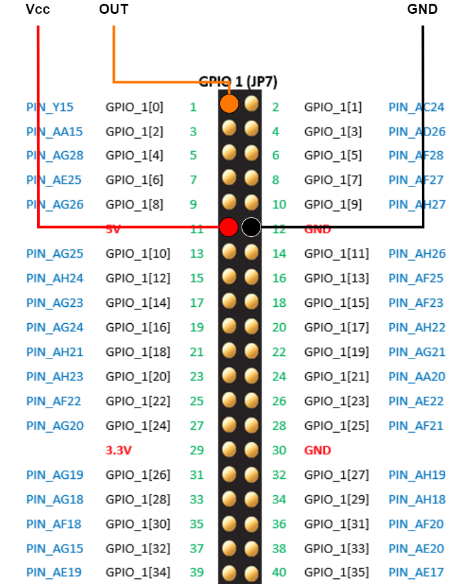


**Adjustments**: It has some adjustments:

* The Jumper allows you to define the frequency range (it is recommended to put it in the maximum frequency position (10 kH – 200 kH) so that the number of 50MH pulses contained are in the order of hundreds/thousands (but other solutions can also be analyzed)
* A Trigger to change the frequency. Note: this trigger scales proportionally both the high and low parts of the signal in order to keep the duty-cycle constant, but in doing so its adjustment changes both the duration of the overall period of the signal and the duration of the signal at high level.
* An additional trigger that modifies the duration for which the output signal is high, effectively altering both the duty-cycle and the frequency of the signal itself

**Connections**: The external clock generator needs a power supply and an interface signal.

* **The power supply** can be taken directly from the DE1-SoC, at pins [11] for Vcc and [12] or [30] for GNDs from both JP0 and JP1 (as shown in the image)



* Instead, the **signal** to be measured can be connected, as described in the Verilog code previously developed, from pin [1] (i.e. GPIO1GPIO[0]).

If all connections have been made correctly, the display should now show a number representing the number of 50MHz pulses contained

* + In a complete period of the signal when SW[0] is set to 0.
  + In the high phase of the signal when SW[0] is set to 1.

