Assignment for VHDL hands on workshop

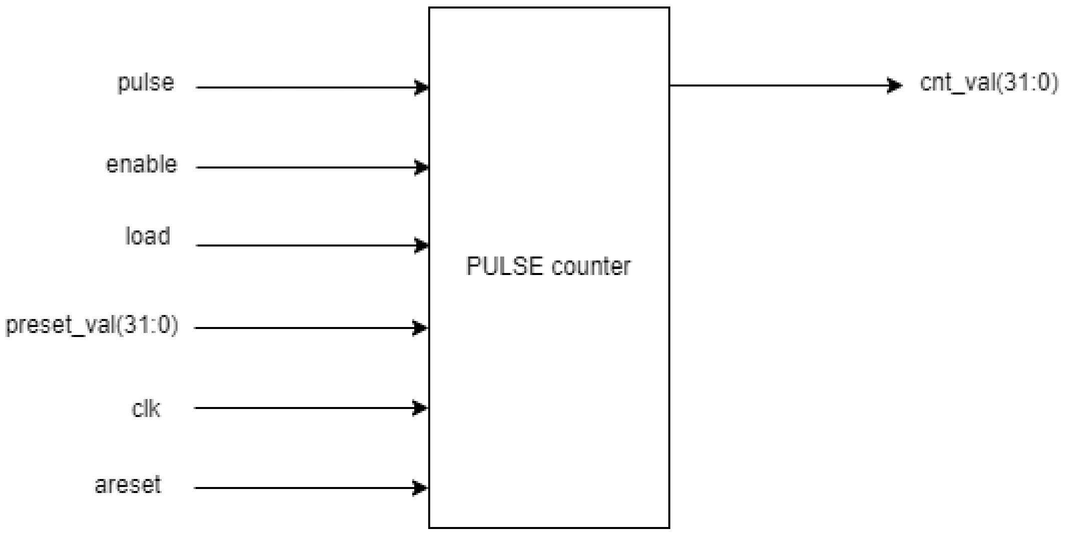
Assignment 1: Counter module

Design

A Geiger counter typically gives out an analog negative pulse when a radioactive particle is detected.

In this assignment we assume that this signal has been turned into a digital signal with positive polarity – i.e. it is high (or digital '1') when a hit is seen, and low (digital '0') otherwise.

The top level of your counter module is given below:



The other specifications are:

1. The counter should be 32 bits (or selectable size by using a *generic*)
2. The input signal *pulse* should be assumed to be asynchronous to the clock – i.e. you need to make a synchronizer.
3. When enable is high the count module should:
   a. Count the number of rising edges on the *pulse* input.
   b. Be able to load a value into the counter to start from a different value than 0
4. When enable is low, the *pulsecounter*'s *cnt_val* should be set to zero.

Tip: Use *std_logic, std_logic_vector* as datatypes for all signals, except for the counter itself. Then use *unsigned* and cast it to a *std_logic_vector* on the output.

Testbench

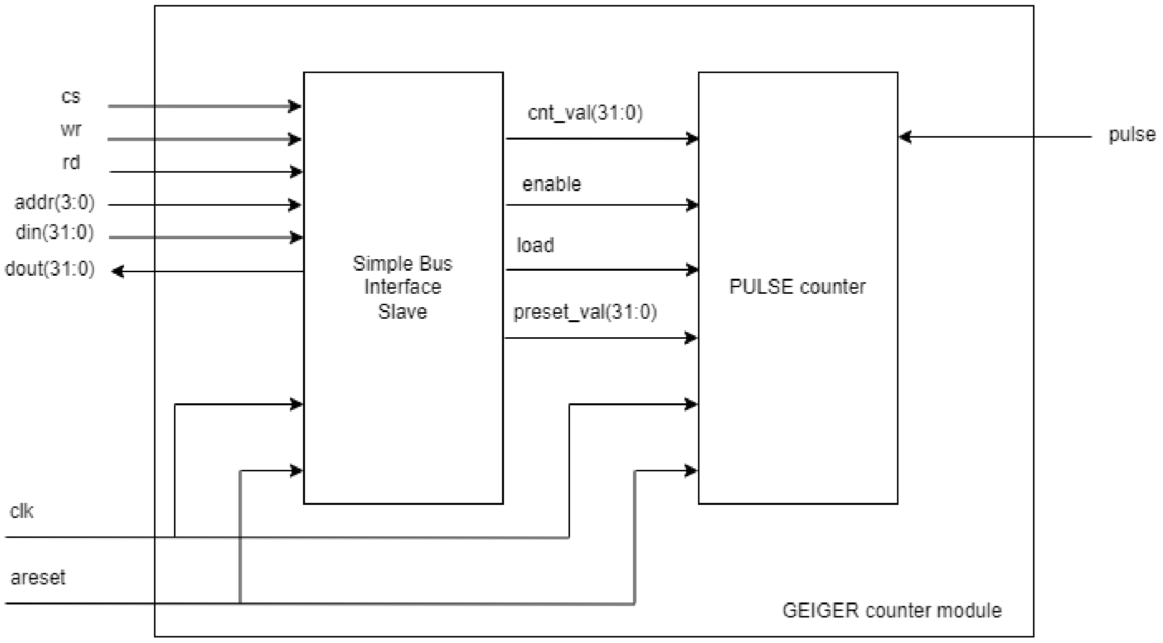
The module should have a testbench that simulates the basic functionality of the module. The best is that this testbench is self checking, i.e. you do not need to look at the waveforms to verify that the tests are successful.

There are several ways to do this, but the simplest way is to use the *assert* and *report* statements. I.e. you can assert a true statement, if this fails it will report in the log a self defined message with a given severity level.

If there is functionality that will be used several times it is wise to make a procedure and call it, as in normal SW programming.

Assignment 2: Adding a Simple Bus Interface

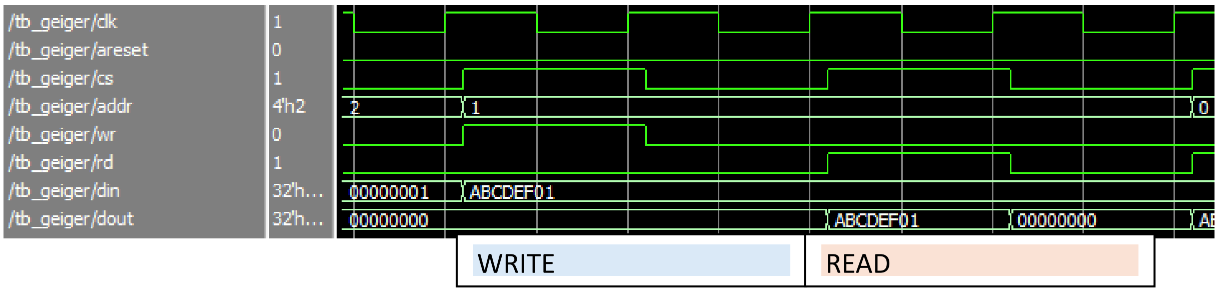
Design



In this assignment you will add a simple bus interface (SBI) to your module from assignment 1. In this way, your module can be connected to a CPU that can act as a master for your module.

The SBI protocol is a simple memory-mapped bus protocol. The term memory-mapped implies that you talk to all the submodules on the bus by giving them individual addresses on which the respond to.

The waveforms for the simple bus interface are given below:



The first transaction is a write to address 0x1, which can be recognized by both the *cs* and the *wr* line being high. The second transaction is a read from address 0x1, where the data from register with address 0x1 is read by the master. You can see that the *dout* gets the value that was previously written by *din*.

This sbi slave address decoder has the following specifications:

- You must decide on a register map, i.e. which addresses you should use to be able to control all the signals between the two modules.
    - The address can be 4 bits wide
    - The data (din and dout) can be 32 bits wide
    - You must decide the access of these addresses, i.e. should the by read and write (RW), read only (RO) or write only (WO).
- You should make two processes, one for writing to the slave, and one for reading from the slave. The write process should be synchronous, while the read process can be combinatorial. Hint: use *case statements* for selecting the address.

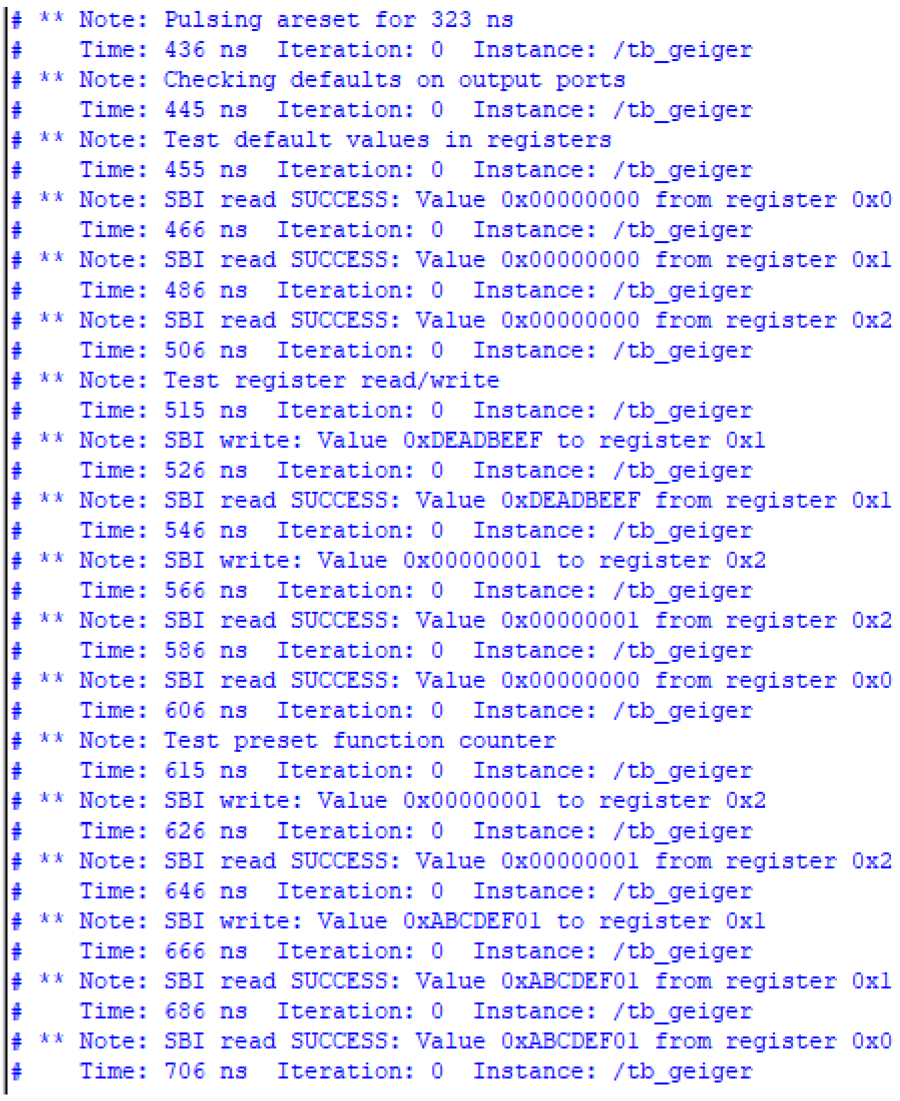
When the SBI slave is done you should make a top level module (*geiger.vhd*) that wraps the two modules like shown in the figure above.

Testbench

You should make a complete testbench for this as well. The testbench should have:

- A process called a test sequencer that defines all the tests you should do and then executes them. Some standard tests are:
    - Check defaults on output ports (after reset)
    - Check defaults on readable registers.
    - Check write and then read on RW registers.
    - Test to write a preset value to the counter, and check if the counter has been set.
    - …
- It is advisable to make procedures for *sbi_write* and *sbi_read*. These are made in the sequencer process, just before the begin word, and should define the sequence of setting the SBI bus for a write and a read respectively.
    - These procedures should be self checking – i.e. – the should test if the value read back from the bus is as expected. Using *assert* and *report* statements the output log could look like shown on the next page.

You will be able to see examples.



```
# ** Note: Pulsing areset for 323 ns
#    Time: 436 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: Checking defaults on output ports
#    Time: 445 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: Test default values in registers
#    Time: 455 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0x00000000 from register 0x0
#    Time: 466 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0x00000000 from register 0x1
#    Time: 486 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0x00000000 from register 0x2
#    Time: 506 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: Test register read/write
#    Time: 515 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI write: Value 0xDEADBEEF to register 0x1
#    Time: 526 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0xDEADBEEF from register 0x1
#    Time: 546 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI write: Value 0x00000001 to register 0x2
#    Time: 566 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0x00000001 from register 0x2
#    Time: 586 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0x00000000 from register 0x0
#    Time: 606 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: Test preset function counter
#    Time: 615 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI write: Value 0x00000001 to register 0x2
#    Time: 626 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0x00000001 from register 0x2
#    Time: 646 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI write: Value 0xABCDEF01 to register 0x1
#    Time: 666 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0xABCDEF01 from register 0x1
#    Time: 686 ns  Iteration: 0  Instance: /tb_geiger
# ** Note: SBI read SUCCESS: Value 0xABCDEF01 from register 0x0
#    Time: 706 ns  Iteration: 0  Instance: /tb_geiger
```

More information

There is a course at UiO that is fully open on the web that our students sometimes do. If you want to learn more about VHDL programming or want to get some videos and slides you can look at and learn for yourself (or while sitting here), it is a very nice alternative.

https://fys4220.github.io/