Participants:

Jacob Munkholm, 20140479

Jens Jakob Mikkelsen, 201506215

Yang Hanshou, 201902791

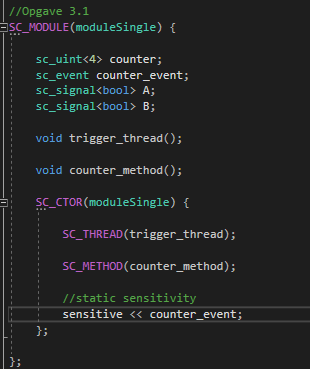
**Embedded Real Time Systems – Assignment 1**

**System level modeling using SystemC**

# Modules, threads, methods and events

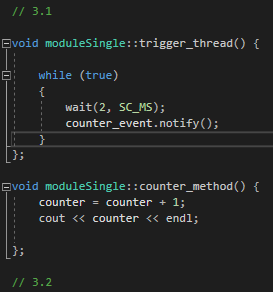
## 3.1:

The module ’moduleSingle’ is created:

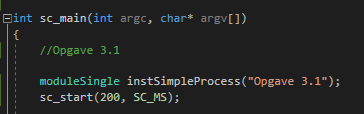


The thread to continuously notify the method, and the method to count, is initialized in the constructor. Static sensitivity is used, so counter\_method() is called everytime the event is raised.

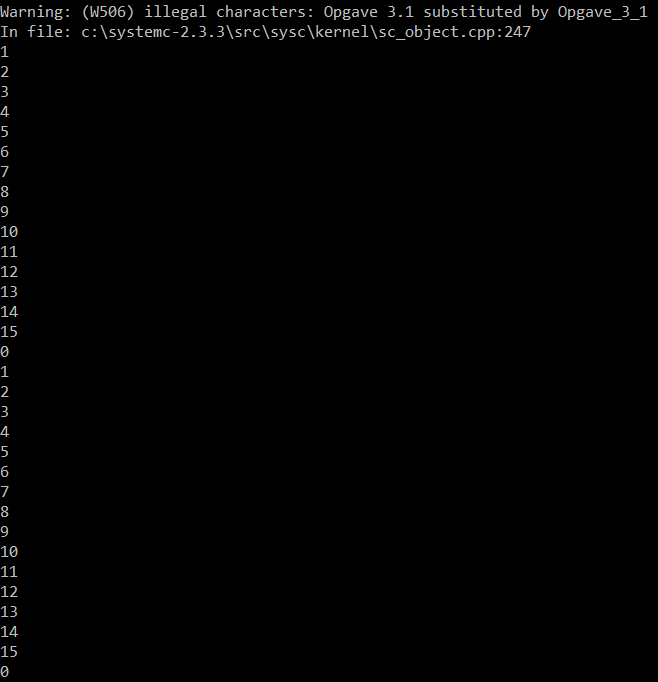
’Trigger\_thread() og ’counter\_method’ looks like this



’Trigger thread’ is an unending loop, calling counter\_event.notify() every other ms. In the main function, the simulation is limited to 200 ms:



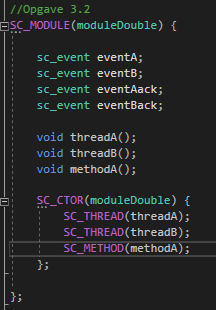
The following is the result:



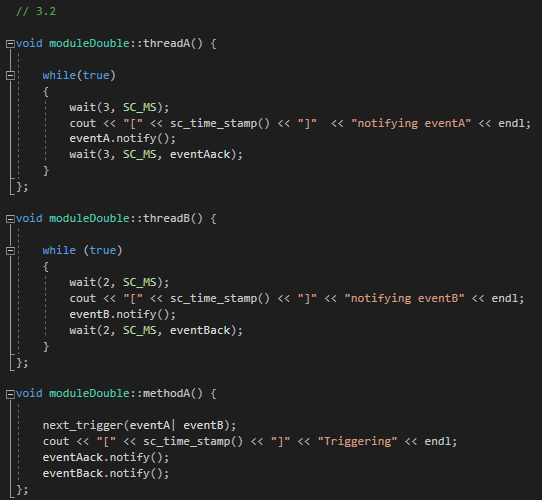
When sc\_uint<4> counter overflows, the program will count from zero again.

## 3.2.

The different threads and events and the method is created:



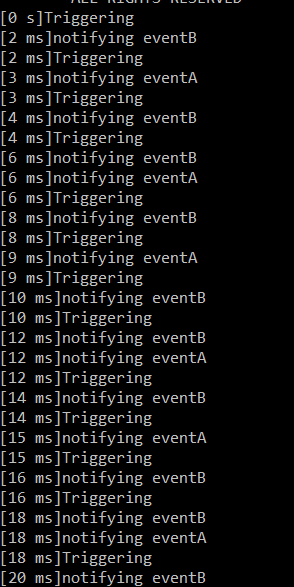
The threads and the method looks like this:



In methodA(), dynamic sensitivity in the form of next\_trigger(eventA | eventB) is used, så the method will be called by if either of the events are called. If the program should work so that methodA knows exactly which event is called, signals would need to be used.

Both event threadA and threadB wait for their event to be called. If no event is called, the thread will timeout and restart. This it done in the wait function by specifying eventAack and eventBack as parameters.

The result looks like this:

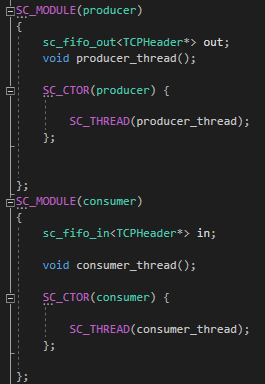


At 6, 12 and 18 ms the events overlap, but eventB is seen to be quickest everytime (because it is called first).

# Channels, signals, hierarchy, communication

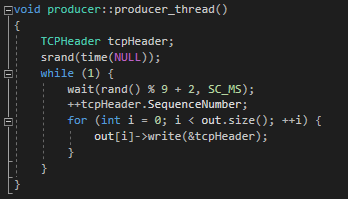
## 3.3

A producer and consumer-thread is created:

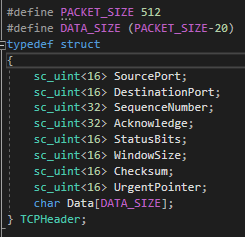


The main object to be used is the sc\_fifo\_in/out, through which data is transferred.

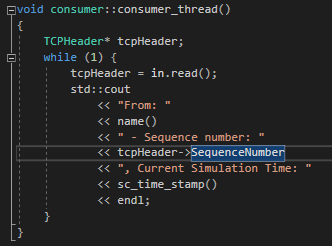
The producer thread is written like this:



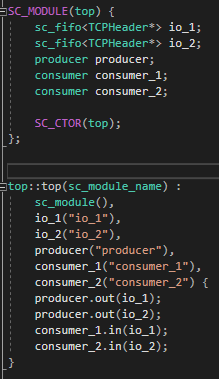
The thread will wait a random amount of time between 2 and 10 ms, increment the sequence-number (the whole TCP segment structure is not used) and write the sequence to the queue. The TCP-struct is defined, but only the sequenceNumber is used to demonstrate the concept:



The consumer-thread read the queues continuously and then writes out the sequenceNumber:

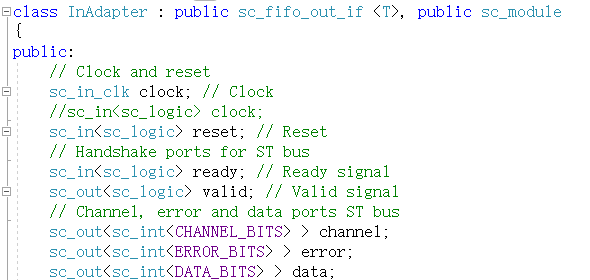


In the top-module, the producer, consumer and fifos are created. More consumers and producers could be added like this, if need be.

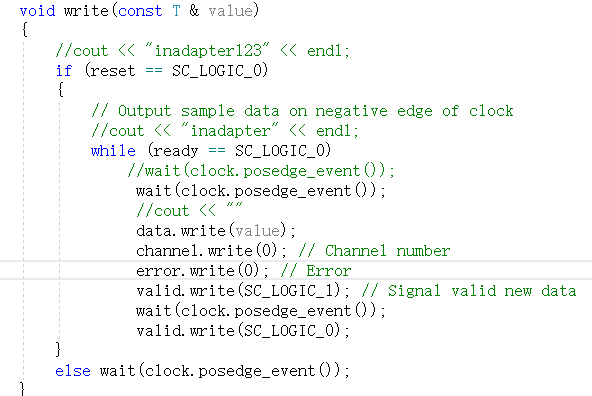


## 3.4

3.5



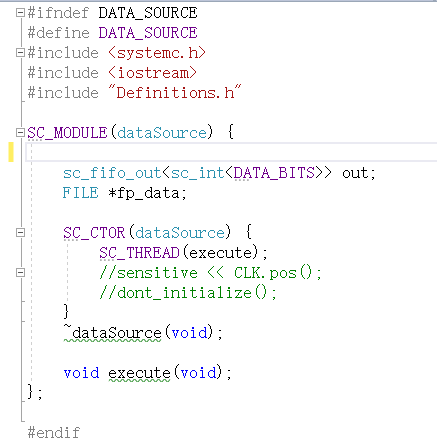
Definition of the InAdapter. The job of the InAdapter: Receiving data from Master, receiving ready signal from Salve, transfer the data to the Slave, output the signal error, channel to the Salve, and also receive the command reset and clock.

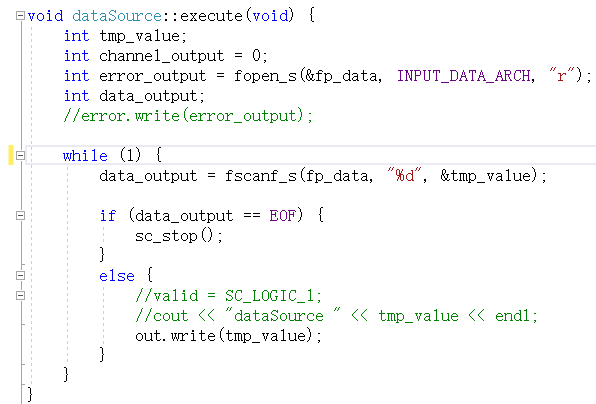


The write function of the InAdapter:

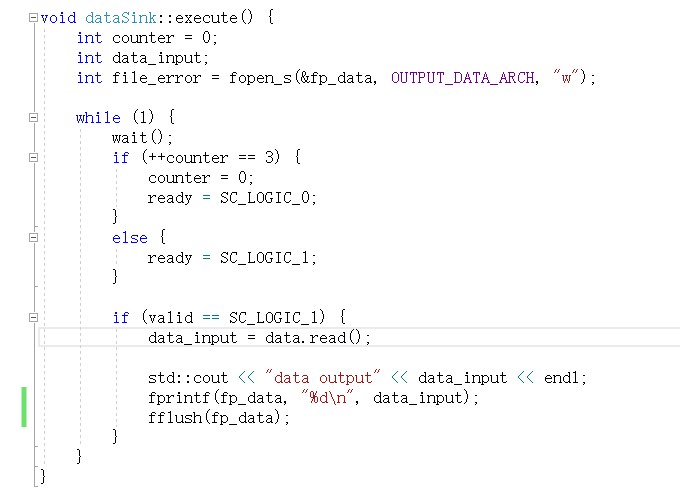
If it receive the command “reset”, then it would be idle and wait for next clock;

If it do not receive the “reset”, wait for the “slave” to be ready, and when the Slave is ready, send the “data”, “channel”, “valid” to the Slave.

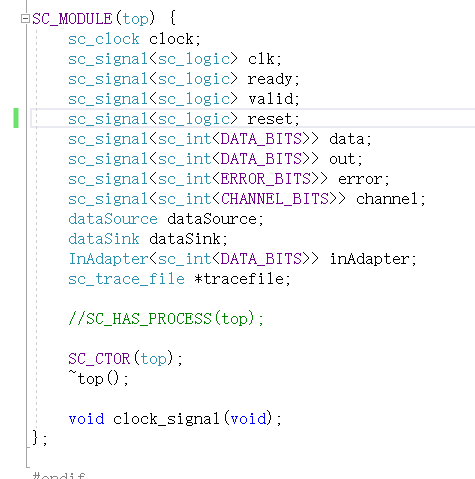


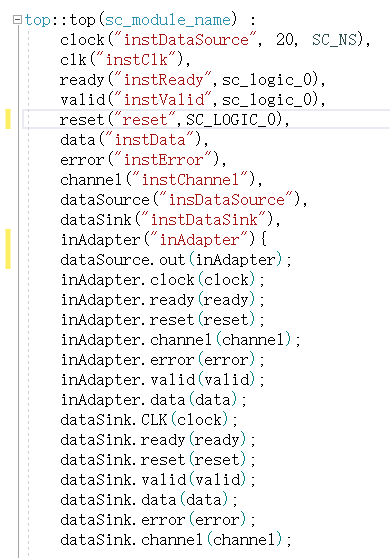


The definition and implement of “master”: it only does the job of reading data from input file and sending data to the InAdapter.



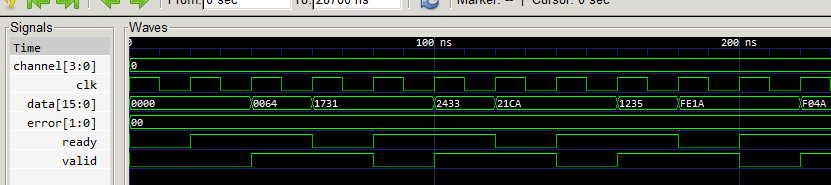
The “Slave”, which is “dataSink” is the same as the on in ex 3.4, but with a new line fflush(fp\_data); just clean up the space cache for the new data waiting to be written to the output file.





The definition and implement of the Top: Since InAdapter inherit from the class sc\_fifo\_out\_if (as stated in the definition) and the “master” has a sc\_fifo\_out, they can be connected to each other directly (dataSource.out(inAdapter));

And then, InAdapter does the same job as the “master” did in the ex3.4: connecting with the “slave”.



Simulation result:

As the code stated in InAdapter, when the InAdapter receives the “ready” signal, it would sent signal “valid” with value 1 after another clock. As stated in the simulation, the “valid” is one clock lagged behind the “ready”.

Since the “ready” signal is always change, and the InAdapter has to wait for the “Slave” ready to transfer the data to it, hence, in the simulation picture, some data`s transfer period is long while the others` is short, it is because they have to wait the “slave” to be ready.

The “channel” and the “error”, as stated in the code in InAdapter, always the same value.