SystemC tutorial

https://github.com/AleksandarKostovic/SystemC-tutorial#what-is-systemc

What is SystemC

SystemC is a collection of classes and libraries that provide event driven simulation for a system modeling language called SystemC. Its a way to enable hardware modeling functionality within C++. SystemC is based on C++, which gives it speed and flexibility. TLM 2.0 is not covered here, but it comes inside SystemC folder you download.

Being based on C++, SystemC doesnt require any special EDA tool in order to use it. All you need is a C++ compiler that you can link your installation to.

Knowledge of C++ and basic hardware concepts(like clocks, gates and waveforms) is required in order to understand this.

Repository structure

This repository contains few sub-directories

```
 SystemC-tutorial             # top directory
 └── examples                 # directory with examples
     ├── counter              # counter example with Makefile, not
synthesizable by HLS tools
     ├── hello-world          # hello-world example with Makefile, not
synthesizable by HLS tools
     └── synthesizable        # directory containing examples synthesizable
by HLS tools
```

Installation

Dependencies:

- For Debian/Ubuntu and Ubuntu based distros

```
sudo apt install build-essential make wget git gcc g++
```

- For Arch, Manjaro and other Arch based distros

```
sudo pacman -S make wget git gcc g++
```

SystemC can be downloaded, free of charge, from Accellera's website.

Steps:

1. Open terminal and type

```
wget http://www.accellera.org/images/downloads/standards/systemc/systemc-2.3.3.gz
```

This downloads the SystemC tarball

2. Unpack the package and make directories

```
tar -xzf systemc-2.3.3.gz
sudo mkdir /usr/local/systemc-2.3.3/
cd systemc-2.3.3 && mkdir objdir && cd objdir
```

3. Final installation

```
sudo ../configure --prefix=/usr/local/systemc-2.3.3/
sudo make -j$(nproc)
sudo make install
```

Compile the program

For Linux

First clone this repo

```
git clone https://github.com/AleksandarKostovic/SystemC-tutorial.git
```

Then run the command to compile SystemC into executable called hello

```
g++ -I. -I /usr/local/systemc-2.3.3/include -L. -L/usr/local/systemc-2.3.3/lib-linux64 -Wl,-rpath=/usr/local/systemc-2.3.3/lib-linux64 -lsystemc -lm -o hello hello.cpp
```

In the examples I provided makefiles so all you have to do is type make in the example directory

Now, type `./hello` to run the executable and you should get this:

```
        SystemC 2.3.3-Accellera --- Nov 15 2018 12:20:10
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
Hello World!
```

Keywords introduced and some building concepts

Ports

Ports in SystemC are similar to those found in HDL's. They are either inputs, outputs or bidirectional ports. They are designed as:

- `sc_in` - Input port
- `sc_out` - Output port
- `sc_inout` - Bidirectional port

There are ports for clocks like `sc_in_clk` but it is recommended to use regular ports even for clock. For example `sc_in<bool> clock` is just an input port with Boolean nature. Its either high or low(when thinking about clock) - 1 or 0.

sc_main

`sc_main` is the master function. When building a system based on SystemC, `sc_main` is going the whole system's main function. You can build multiple functions, but `sc_main` must be present. Like in hello world example:

```
#include <systemc.h>

SC_MODULE (hello) {  // module named hello
  SC_CTOR (hello) {  //constructor phase, which is empty in this case
  }

  void say_hello() {
    std::cout << "Hello World!" << std::endl;
  }
};

int sc_main(int argc, char* argv[]) {
  hello h("hello");
  h.say_hello();
  return 0;
}
```

We have two functions. The `say_hello` is responsible for outputting text, while the `sc_main` is passing the `say_hello` function and returning 0(success).

SC_MODULE

`SC_MODULE` is meant to be a declaration of a complete module/part. It has the same intention as `module` in Verilog, but just in SystemC style.

SC_CTOR

`SC_CTOR` is macro file for a SystemC constructor. It does several things:

- Declares sensitivity list.

In SystemC a sensitivity list is part of constructor which declares which signals are most sensitive. For example:

```
sensitive << clk.pos();
```

This tells the module that the design is sensitive to clock, the positive edge in this case.

- Register each function as a process happening in a module.
- Create design hierarchy if you are including serval modules to give whole design sense of module usage.

Threads

Thread is a function made to act like a hardware process. It has a few features:

- Runs concurrently - Multiple processes can be started at the same time(note that every function in systemc is a process)
- They are sensitive to signals.
- They are not called by user, but rather always active.
- There are three types of threads: `SC_METHOD`, `SC_THREAD`, `SC_CTHREAD`.

`SC_METHOD`

- Limited to single clock cycle. Fine for simple sequential logic
- They execute once every sensitive event
- They run continuously
- Are synthesizable
- Are comparable to Verilog's `always @` block

`SC_THREAD`

- Runs once at the start of the simulation, than suspends ifself when done.
- Can contain infinite loop
- Comparable to Verilog's `@ initial` block
- Not synthesizable
- Typically used in test benches to describe clocks

`SC_CTHREAD` - clocked threads

- Synthesizable
- Not limited to one cycle
- Can contain continuous loops
- Can contain large blocks of code for control or code with operations
- Used for behavioral synthesis
- Run continuously
- Can take more clock cycles to execute a single iteration
- Used for 99% of SystemC designs
- Similar to Verilog's `always @ (pos/negedge clock)`