

## Exercise 1: SystemC and Virtual Prototyping

### Setting Up Your Environment

*Lukas Steiner*

WS 2023/2024

## Prerequisites for Windows Users

### Installing Windows Subsystem for Linux (WSL)

On Windows platforms we recommend using the **Windows Subsystem for Linux** (WSL). General information about the WSL can be found [here](#). Information on the installation can be found [here](#). To keep things unified please choose **Ubuntu** or **Debian** as your Linux distribution.

### Running Graphical Applications on WSL

In order to run applications with a graphical user interface you have to set up **X11 forwarding** for the WSL. Download and install [VcXsrv](#). Run XLaunch, check the box *Disable access control* in the *Extra settings* window and save your configuration. If you want VcXsrv to be automatically started at system startup create a shortcut of your saved configuration and add it to the startup folder (press Windows key + R and type `shell:startup` to open the folder).

### WSL 1 Specific Setup

Add the following line to the `.bashrc` file (located in the Linux distribution's home directory) and restart the WSL.

```
export DISPLAY=:0
```

---

## WSL 2 Specific Setup

Add the following two lines to the `.bashrc` file (located in the Linux distribution's home directory) and restart the WSL.

```
export DISPLAY=$(awk '/nameserver / {print $2; exit}' \
    /etc/resolv.conf 2>/dev/null):0
export LIBGL_ALWAYS_INDIRECT=1
```

Afterwards, disable the block rule for VcXsrv (TCP) on the public network: open *Windows Firewall* → *Advanced settings* → *Inbound Rules* → *VcXsrv windows x server* (Public, TCP) → *Disable Rule*.

Add a new rule for TCP: right click on *Inbound Rules* → *New Rule...* → *Port* → *TCP port 6000* → select defaults in the remaining windows. Open the properties of your new rule, switch to the *Scope* window, select *These IP addresses* in the *Remote IP address* window and add the address `172.16.0.0/12`.

Additional information and support can be found [here](#) and [here](#).

## Testing the X11 Forwarding

Finally, you can test X11 forwarding by installing the `x11-apps` and executing `xeyes` in the WSL:

```
$ sudo apt install x11-apps
$ xeyes
```

## Remark for Linux and macOS Users

The following sections will demonstrate the installation of required software tools using the **APT** package manager, which is available on **Ubuntu** and **Debian**. If you are using a different Linux distribution or macOS you should use the appropriate package manager, e.g., **YUM** on **Fedora** or **Homebrew** on **macOS**.

---

# Git

**Git** is a free and open source **distributed version control system** designed to handle everything from small to very large projects with speed and efficiency.

If you are not familiar with Git you can find some information [here](#) and [here](#). There are also thousands of tutorials on Youtube and elsewhere on the internet.

You can install Git using the following command:

```
$ sudo apt install git
```

Afterwards configure your Git installation:

```
$ git config --global user.name "FirstName OtherNames LastName"
$ git config --global user.email "email@example.com"
$ git config --global credential.helper 'cache --timeout=3600'
$ git config --global color.ui auto
```

Now you can clone the SCVP artifacts repository:

```
$ git clone https://github.com/TUK-SCVP/SCVP.artifacts.git
```

If you want to fork the repository and push your local changes create an account on [GitHub](#).

# CMake

**CMake** is a widely-used tool for managing the **build process of C++ projects**. CMake is available on different platforms and is **compiler independent**. Many IDEs provide support for CMake or even use CMake for their project management by default. Alternatively the tool can be used from command line. More information is provided [here](#).

Since the SCVP artifacts as well as all exercises are also based on CMake, you should install it using the following command:

```
$ sudo apt install cmake
```

---

# Installing SystemC

First, check if a C++ compiler and Make is available on your system. If not you can simply install the build-essential package:

```
$ sudo apt install build-essential
```

Now download the latest SystemC and SystemC AMS sources from the official websites:

```
$ wget https://www.accellera.org/images/downloads/standards/systemc/systemc-2.3.3.tar.gz
```

```
$ wget https://www.cosedatech.com/files/Files/Proof-of-Concepts/systemc-ams-2.3.tar.gz
```

Build and install both libraries. The following steps are shown exemplarily for the SystemC library. Extract the files, create a temporary directory inside the extracted folder and navigate into it:

```
$ tar -xzf systemc-2.3.3.tar.gz
```

```
$ mkdir systemc-2.3.3/objdir
```

```
$ cd systemc-2.3.3/objdir
```

Run the configure script specifying the directory where the library should be installed at (default is `/opt/systemc/` for SystemC and `/opt/systemc-ams/` for SystemC AMS but you can also choose different directories). Further information about the configuration options is provided in the `INSTALL` files.

```
$ ../configure --prefix=/opt/systemc/
```

Afterwards, build the library and install it (N denotes the number of build jobs, set it to the number of processor cores):

```
$ make -j N
```

```
$ sudo make install
```

As last step export environment variables for SystemC. You can also add the lines to the shell startup script (e.g. `.bashrc`) to automatically export the variables on startup.

```
$ export SYSTEMC_HOME=/opt/systemc/
```

```
$ export SYSTEMC_TARGET_ARCH=linux64
```

If you have chosen a different directory for installation adapt the first variable accordingly.

---

The second variable denotes the target architecture suffix. Check the library folder name inside the installation directory and set the variable accordingly. The library folder name usually starts with `lib-`.

Now repeat all steps for the SystemC AMS library. Finally, export the following environment variable (and adapt it accordingly):

```
$ export SYSTEMC_AMS_HOME=/opt/systemc-ams/
```

## Testing the Installation

All required tools are now installed and you can test if your system is working. Create a build folder inside the SCVP artifacts repository and navigate into it:

```
$ mkdir SCVP.artifacts/build
$ cd SCVP.artifacts/build
```

Create a makefile using CMake and build the project:

[why do we create a separate folder for Cmake and Build?](#)

```
$ cmake ..
$ make -j N
```

Now execute one of the examples, e.g., the `tlm_protocol_checker`:

```
$ ./tlm_protocol_checker/tlm_protocol_checker
```

You should see the following program output:

```
@40 ns Write Data:  103
@80 ns Write Data:  198
@120 ns Write Data: 105
@160 ns Write Data: 115
@200 ns Read Data:  103
@240 ns Read Data:  198
@280 ns Read Data:  105
@320 ns Read Data:  115
```

For an easier and less error-prone development you can also use an IDE to do the exercises, e.g., Visual Studio Code, CLion or Qt Creator. All IDEs support CMake for project management.

---

## Remark for Windows Users

With WSL you can also use your Windows installation of Visual Studio Code and compile and run your code in the Linux environment using a plugin. More information can be found [here](#) and [here](#).