Virtual Platform

1. Overview

The NVDLA virtual platform provides a register-accurate system on which software can be quickly developed and debugged. The virtual platform is based on GreenSocs QBOX (https://git.greensocs.com/qemu/qbox), a solution for co-simulation with QEMU and SystemC. Fig. 89 below shows the top level diagram of the NVDLA virtual simulator. A QEMU emulator of ARMv8 'virt' SoC board is included to provide high performance CPU emulation and generic devices. The emulator is wrapped inside a standard SystemC module with a set of TLM-2.0 interfaces to interact with other SystemC modules.

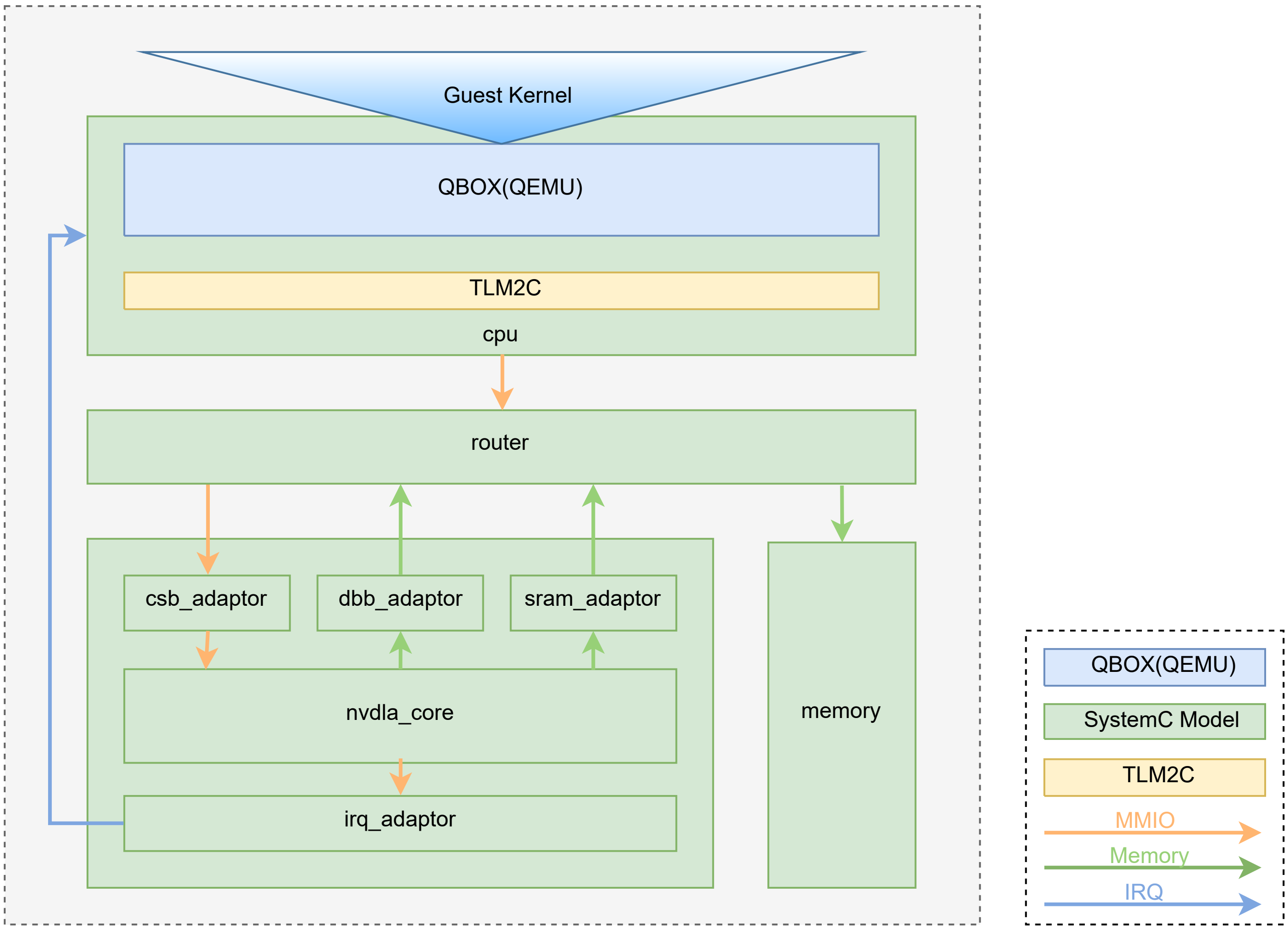

Fig. 89 - NVDLA Virtual Platform.

2. Using the Virtual Simulator

It is recommended that you use the pre-built virtual simulator binary included in the docker (https://hub.docker.com/r/nvdla) image. The 2.6 Running the Virtual Simulator From Docker section describes the detail. You are also welcome to build the virtual simulator from source code and use that.

2.1 System Requirements

It is recommended to build the virtual simulator in a system that meets the following requirements:

- OS: Ubuntu 14.04

- g++ 4.8.4
- git > 1.8.2
- systemc 2.3.0
- cmake > 2.8
- libboost > 1.34
- python dev
- glib2 dev
- pixman dev
- lua 5.2 dev
- swig
- libcap dev
- libattr1 dev

Note

virtual simulator also can run with CentOS 7, details refer to Virtual Platform On AWS FPGA (vp_fpga.html)

Several tools like Java and Perl are also required to build the NVDLA hardware tree. Please refer to Environment Setup (hw/v1/integration_guide.html#env-setup) for details.

2.2 Download the Virtual Simulator

The NVDLA virtual simulator source code can be downloaded from github (https://github.com/nvdla/vp).

```
$ git clone https://github.com/nvdla/vp.git
$ cd vp
$ git submodule update --init --recursive
```

2.3 Install Dependencies

2.3.1 Install required tools and libraries

```
$ sudo apt-get update
$ sudo apt-get install g++ cmake libboost-dev python-dev libglib2.0-dev libpixman-1-dev liblua5
```

2.3.2 Download and install SystemC 2.3.0

Please be noted that SystemC 2.3.1/2.3.2 is not supported currently.

```
$ wget -O systemc-2.3.0a.tar.gz http://www.accellera.org/images/downloads/standards/systemc/sys
$ tar -xzvf systemc-2.3.0a.tar.gz
$ cd systemc-2.3.0a
$ sudo mkdir -p /usr/local/systemc-2.3.0/
$ mkdir objdir
$ cd objdir
$ ../configure --prefix=/usr/local/systemc-2.3.0
$ make
$ sudo make install
```

2.3.3 Download and install perl package required

We need to install perl package YAML.pm and Tee.pm to build NVDLA CMOD.

```
$ wget -O YAML-1.24.tar.gz http://search.cpan.org/CPAN/authors/id/T/TI/TINITA/YAML-1.24.tar.gz
$ tar -xzvf YAML-1.24.tar.gz
$ cd YAML-1.24
$ perl Makefile.PL
$ make
$ sudo make install
$ wget -O IO-Tee-0.65.tar.gz http://search.cpan.org/CPAN/authors/id/N/NE/NEILB/IO-Tee-0.65.tar.
$ tar -xzvf IO-Tee-0.65.tar.gz
$ cd IO-Tee-0.65
$ perl Makefile.PL
$ make
$ sudo make install
```

2.3.4 Download and build NVDLA CMOD

Please refer to Tree Build (hw/v1/integration_guide.html#tree-build) for details on building the NVDLA hardware tree, and make sure the required tools listed in Environment Setup (hw/v1/integration_guide.html#env-setup) are installed first.

```
$ git clone https://github.com/nvdla/hw.git
$ cd hw
$ make
$ tools/bin/tmake -build cmod_top
```

The header files and library will be generated in *hw/outdir/<project>/cmod/release*.

2.4 Build and Install the Virtual Simulator

2.4.1 Cmake build under the vp repository directory

```
$ cmake -DCMAKE_INSTALL_PREFIX=[install dir] -DSYSTEMC_PREFIX=[systemc prefix] -DNVDLA_HW_PREFI
```

install dir is where you would like to install the virtual simulator, *systemc prefix* is the SystemC installation directory, *nvdla_hw prefix* is the local NVDLA HW repository, and *nvdla_hw project name* is the NVDLA HW project name.

Example:

```
$ cmake -DCMAKE_INSTALL_PREFIX=build -DSYSTEMC_PREFIX=/usr/local/systemc-2.3.0/ -DNVDLA_HW_PREF
```

2.4.2 Compile and install

```
$ make
$ make install
```

2.5 Running the Virtual Simulator

2.5.1 Prepare Kernel Image

A demo linux kernel image is provided in the github release. You can run this image in the virtual simulator, and run the NVDLA KMD/UMD inside it.

If you would like to build a linux kernel on your own, please refer to 3. Building Linux Kernel for NVDLA Virtual Simulator.

After the image is ready, modify the *conf/aarch64_nvdla.lua* for the image and rootfs file location.

2.5.2 Standard QEMU Arguments

The configuration of the virtual simulator is defined in *conf/aarch64_nvdla.lua*. You can change the standard QEMU arguments in *extra_arguments* inside the lua file.

2.5.3 Running Kernel Image In the Virtual Simulator

Start the virtual simulator. Pay attention that the environment variable 'SC_SIGNAL_WRITE_CHECK' should be set to 'DISABLE' before we run the virtual simulator.

```
$ export SC_SIGNAL_WRITE_CHECK=DISABLE
$ ./build/bin/aarch64_toplevel -c conf/aarch64_nvdla.lua
Login the kernel. The demo image uses account 'root' and password 'nvdla'.
```

Some demo tests are provided in the *tests* directory, you can run them after login as root.

```
# mount -t 9p -o trans=virtio r /mnt
# cd /mnt/tests/hello
# ./aarch64_hello
```

You should be able to see 'Hello World!' printed in the screen. You are now ready to try out the NVDLA software in the virtual simulator! Please refer to Software Manual (sw/contents.html) for details.

If you want to exit the virtual simulator, press 'ctrl+a x'.

2.6 Running the Virtual Simulator From Docker

```
$ docker pull nvdla/vp
$ docker run -it -v /home:/home nvdla/vp
$ cd /usr/local/nvdla
$ aarch64_toplevel -c aarch64_nvdla.lua
Login the kernel with account 'root' and password 'nvdla'
```

The NVDLA software is also provided in the docker image, please refer to Software Manual (sw/contents.html) on how to run the NVDLA software.

2.7 Debugging the Virtual Simulator

Before debugging the virtual simulator, you need to build the debug version of the simulator:

```
$ cmake -DCMAKE_INSTALL_PREFIX=[install dir] -DSYSTEMC_PREFIX=[systemc prefix] -DNVDLA_HW_PREFI
$ make
$ make install
```

2.7.1 Log Output Control

The log output of SystemC simulator is controled by a configuration string that can be set in two ways:

- The command line option '-s *control_string*' or '–sc_log *control_string*'

- The environment variable 'export SC_LOG=*control_string*'

If both control strings are set, the simulator will pick the one set by environment variable *SC_LOG*.

The format of the control string is:

```
"outfile:<log_file>;verbosity_level:<info_level>;<msg_string>:<report_level>"
<log_file>: log output file name
<info_level>: info verbosity -- sc_none/sc_low/sc_medium/sc_high/sc_full/sc_debug
<msg_string>: message string specified in sc_report.
<report_level>: sc report level -- info/warning/error/fatal/enable/disable
```

Here are some useful control string examples:

```
$ export SC_LOG="outfile:sc.log;verbosity_level:sc_debug;csb_adaptor:enable" -- print the regis
$ export SC_LOG="outfile:sc.log;verbosity_level:sc_debug;dbb_adaptor:enable;sram_adaptor:enable
```

You should be able to see logs like:

```
Info: nvdla.csb_adaptor: GP: iswrite=0 addr=0x300c len=4 data=0x 00000000 resp=TLM_OK_RESPONSE
Info: nvdla.dbb_adaptor: GP: iswrite=1 addr=0xc0001e80 len=64 data=0x abcd01b0 abcd01b1 abcd01b
```

2.7.2 GDB

You can also use GDB to debug the virtual simulator. First run the simulator, then get the PID of the process and use GDB to attach to it.

```
$ ps -ef | grep aarch64_toplevel
$ gdb attach <PID>
```

3. Building Linux Kernel for NVDLA Virtual Simulator

The NVDLA virtual platform is based on QEMU aarch64 virt machine, so building a linux kernel is the same as building one for QEMU aarch64 virt machine. Here's an example of using buildroot to build a linux kernel for NVDLA virtual platform.

3.1 Download

Download the buildroot from https://buildroot.org/download.html (https://buildroot.org/download.html). This example uses the version buildroot-2017.11-rc1.

3.2 Configure

Use *qemu_aarch64_virt_defconfig* as base config, then set the customized configurations:

```
$ make qemu_aarch64_virt_defconfig
$ make menuconfig
* Target Options -> Target Architecture -> AArch64 (little endian)
* Target Options -> Target Architecture Variant -> cortex-A57
* Toolchain -> Custom kernel headers series -> 4.13.x
* Toolchain -> Toolchain type -> External toolchain
* Toolchain -> Toolchain -> Linaro AArch64 2017.08
* Toolchain -> Toolchain origin -> Toolchain to be downloaded and installed
* Kernel -> () Kernel version -> 4.13.3
* Kernel -> Kernel configuration -> Use the architecture default configuration
* System configuration -> Enable root login with password -> Y
* System configuration -> Root password -> nvdla
* Target Packages -> Show packages that are also provided by busybox -> Y
* Target Packages -> Networking applications -> openssh -> Y
```

3.3 Build

```
$ make -j4
```

When it’s done, you can find the kernel image and rootfs in *output/image*.