Loopy. Tutorial 1

Vladimir Rybalkin - rybalkin [at] rhrk.uni-kl.de

The purpose of this tutorial is to explain how to properly build and run the driver generator; compile client application. This includes generation of board-side and host-sided drivers; compiling client application running on the host. For profound understanding of the material of the current tutorial we highly encourage you to use "HOPP Driver Generator Documentation" [1], which was used as a base for the current tutorial.

1) **Short introduction.** This part explains main terms used in the tutorial.

a) **Driver.** The complete software product is called the driver. It enables communication between software running on the computer and the hardware platform on the board.

b) **Board / board-side.** The driver is split in two parts, one of which has to be uploaded and executed on the hardware platform itself. This part of the driver is referred to as board-side driver. Sometimes, the terms *server* and *server-side* might be used instead, since this driver acts similar to a server.

c) **Host / host-side.** In contrast to the board-side driver, the host-side driver is located on the communicating computer. This part contains the actual API, embedded developers will work with. Sometimes, the terms *client* and *client-side* might be used instead, since this driver part acts more like a client.

The overall architecture of the driver is depicted on Figure 1. Data is sent from an embedding host application to the host-side driver. This driver communicates over some transport medium with the board-side driver, which in turn distributes the received data to corresponding hardware components on the FPGA. Results are sent back through the same chain.

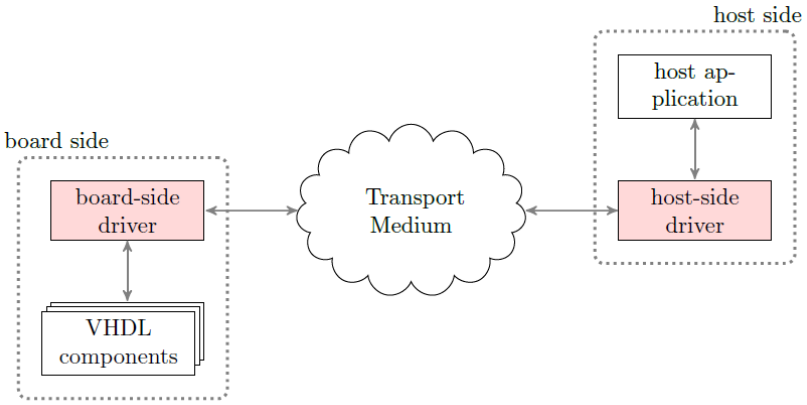

Figure 1: A high-level view of the data flow from a host application to hardware components on the board through the generated driver

2) **Setup.** In the following, the tools required to build and execute the driver generator, and compile a client application are introduced. Please note that all tools have to be executable from the command line. You can use your own computer with the following tools installed or use *zmk.uni-kl.de* servers or *finance.eit.uni-kl.de* server with all the tools already installed. Remember, in order to use the *finance.eit.uni-kl.de* server you need a distinct user account; contact the responsible person to create an account. We recommend you to check the correctness of the setups by following the steps below. To connect to the servers from your own computer you can use *ThinLinc client* available for downloading: http://www.cendio.com/downloads/clients/

a) **Java.** The generator is implemented in Java. Consequently, JDK version 6 or higher is required. Run *"javac –version"* command in order to check the JDK version. The sample output *"java version 1.6.0_14"* signifies that JDK version 6 is installed.

b) **Gradle.** Gradle is a build tool required for building the driver generator. In order to be able to run gradle from any place, add the directory of the tool to *PATH* environment variable. Use *"setenv PATH ${PATH}:/software/gradle-1.5/bin"* in the case of *zmk.uni-kl.de* servers. In the case of *finance.eit.uni-kl.de* server use *"export PATH=$PATH:/opt/gradle-1.6/bin".*

c) **Xilinx Toolsuit.** In order to generate a *.bit* and an *.elf* files that are used to program an FPGA, the Xilinx toolsuite is required. Both XPS and the Xilinx SDK have to be accessible from console using the commands *"xps"* and *"xsdk"* respectively. In order to be able to invoke the tools from command line, use *"source /software/sources/XACT.rc/ise14.4".* You can skip this step, if you use *finance.eit.uni-kl.de* server.

d) **GitHub.** GitHub is a web-based hosting service for software development projects that use the Git revision control system. The sources of the driver generator are located in a GitHub repository. Check if the tool is installed by running *"git"* command in the console.

e) **GCC.** GCC version 4.7.2 or higher is required. Run *"gcc --version"* command to find out the current version of the tool. Nowadays only *finance.eit.uni-kl.de* server has the required version of the *GCC*. We recommend you to use your own computer for client

application development, so make sure that the right version of the tool is installed. However, you can also use the *finance.eit.uni-kl.de* server.

3) **Checkout.** The project has to be checked out. As of now, the project is available at the GitHub repository. The complete command for the initial checkout is *"git clone https://github.com/tukl-msd/msdlib.loopy/ <target dir>"* Later the target directory is referred as a root directory.

4) **Building the driver generator.** After checking out the project, all sources have to be compiled and packaged. This can be done using the gradle build tool. The command to build the jar package is *"gradle jar".* Simply type it in a shell in the project root directory, where the file *build.gradle* is located. Running the build file will generate an executable jar package under the path *<root dir>/build/libs.*

5) **Running the generator.** The behavior of the driver itself is only influenced by the provided *.bdf* file. Before running the generator, the board description file has to be modified. Navigate to *<root dir>/examples/AxiTwoRNG* directory and modify *system.bdf* file. The entries in the *medium ethernet* block have to be changed with respect to the FPGA board and network settings. You can find *ip* and *mac* address on the stickers attached to the board. The sample configuration:

```
medium ethernet {
mac "00:0a:35:02:31:95"
ip "172.16.13.146"
mask "255.255.0.0"
gate "172.16.254.254"
port 8844}
```

a)

```
medium ethernet {
mac "00:0a:35:02:31:95"
dhcp
port 8844}
```

b)

a) the sample configuration in the case of the local network, when zmk.uni-kl.de servers are used b) the sample configuration when DHCP settings are used (in the case of finance.eit.uni-kl.de server*)*

In the current directory you have to find *build.sh* script that is used to run the driver generator. In order to make the script executable run command *"chmod +x build.sh".* Then run *"sh ./build.sh"* command which executes the script. In order to see the progress of the running driver generator, a verbose flag *"-v"* to *build.sh* script was added. After successful completion of the process, you have to be able to find *temp*, *host* and *board* folders in the current directory. The *board* folder has to contain *system.bit* and *app.elf* files representing the board-side driver, which will be used later for an FPGA programming. The *host* directory has to contain */src* folder that includes all the files required for development of the client application running on the host.

6) **Building the client application.** The next step would be writing a host application that uses the generated host-side driver and API. As it has been already mentioned, you can use your own computer or *finance.eit.uni-kl.de* server for the client application development.

a) In the current project, we recommend to use *Eclipse IDE* for host side application development. Run *"eclipse"* command. Choose a path for workspace directory.

b) Create new project. *File –> New –> C++Project*. Specify the name for the project, for example, *“client_host_app”.* Choose the type of the project, *“Hellow World C++ Project”* with *“Linux GCC”* compiler as a toolchain. Click *“Finish”.*

c) Navigate to *<root dir>/examples/AxiTwoRNG/host/src* and copy all content to *<workspace dir>/client_host_app/src* directory of the eclipse project. Next, replace the content of the *“client_host_app.cpp”* with *<root dir>/examples/AxiTwoRNG /main.cpp.*

d) If DHCP settings are used, provide *startup()* function with IP address. Like this, *startup(“131.246.74.3”).* Use the given IP in the case of running client application with DHCP settings on the *finance.eit.uni-kl.de* server.

e) Apply the following settings for the project:

Do not forget to click apply after every step.

- Choose: *Project –> Properties.*
- Navigate to *C/C++ Build –> Settings –> GCC C++ Compiler –> Miscellaneous* and add *“-std=c++0x -pthread”* to *Other flags* textfield (see Figure 2). Add the same line to *C/C++ Build –> Settings –> GCC C++ Linker –> Miscellaneous –> Linker flags.*
- Now, navigate to *C/C++ General –> Preprocessor Include Paths –> Providers*
- disable everything but *“CDT GCC Build-in Compiler Settings”*
- choose it and disable *“Use global provider shared between projects”* checkbox
- add *“-std=c++0x”* to *“Command to get compiler specs”* textfield that is now active (see Figure 3). Click *“Apply”* and *“OK”.*

f) Left click on the project name in *“Project Explorer”* then *Project –> Build Project*. If you see errors, do *Project –> Clean...* then build project again.

g) Now, the project is built and can be run. However, in order to see the result of communication between the board and the host, first proceed with step 7. If you used your own computer for client application development, copy the executable file “*client_host_app”* to the servers either *zmk.uni-kl.de* (if a local network is supposed to be used for communication*) or finance.eit.uni-kl.de.*

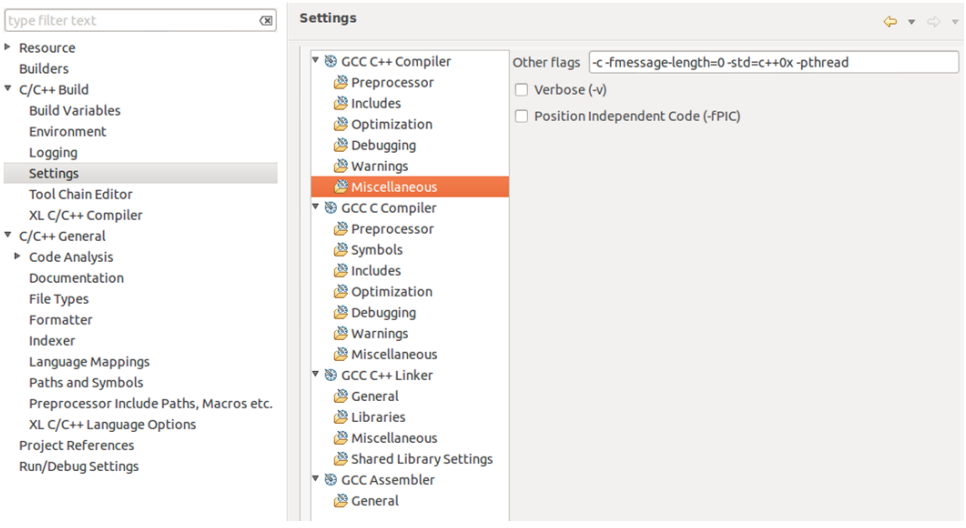

Figure 2: Project settings

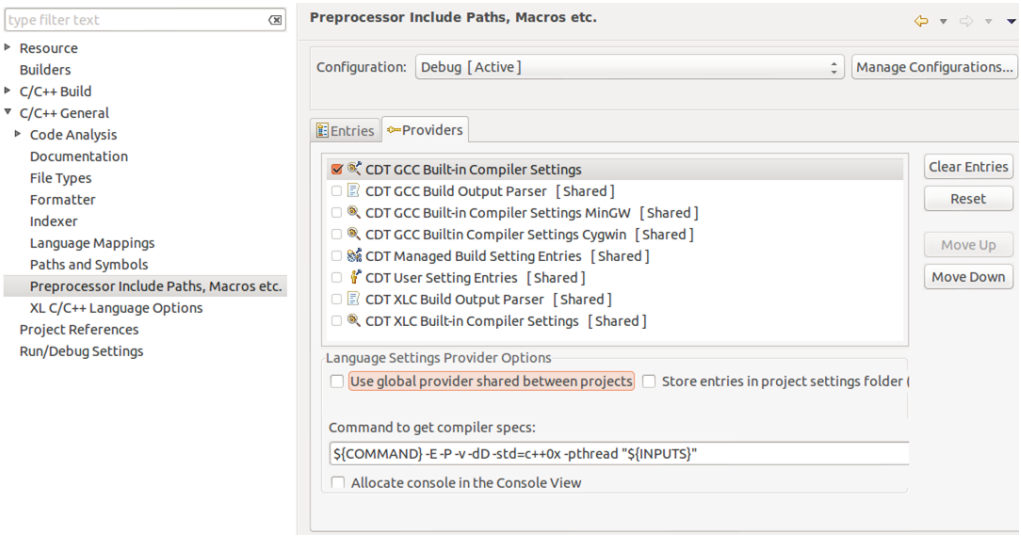

Figure 3: Project settings

7) **Download HW/SW to the FPGA.** In order to download a bitstream and executable files to the FPGA board you will have to use your computer as this step requires physical connection between host computer and board using USB cable. First, you have to install Xilinx ISE Design Suite: WebPACK Edition, which is freely available for downloading on Xilinx webpage: http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html Remember, you have to create account to start downloading.

a) Download the bitstream file *"system.bit"* and executable *"app.elf"* from the server (where you built and ran driver generator) to your host computer (your own computer). Remember that these files are in *the <root dir>/examples/ AxiTwoRNG /board* folder.

b) Move the files to a handy location in your host computer, for example, your home folder.

c) Connect the *Xilinx Virtex-6 FPGA ML605* board to your computer with USB cable. Use USB JTAG (J22) port for connection. Use Ethernet cable to connect the board to Ethernet socket. Use *"172.16.X.X"* socket in the case of local servers and *"131.246.74.X"* in the case of *finance.eit.uni-kl.de* server. Switch power on the board.

d) After Xilinx ISE Design suite installation, open *"ISE Design Suite Command Prompt"* and type *"xmd"* and press enter. You should now be in the MicroBlaze Debug Module.

e) Type *"fpga -f system.bit"* and press enter, this will download the bitstream file to the FPGA.

f) Type *"connect mb mdm"* and press enter, this will connect the MicroBlaze to the Debug Module. In order to proceed, type *"rst"* and press enter.

g) Type *"dow app.elf"* and press enter; this will download the executable to our system.

h) Type *"rst"* and press enter, in order to reset the MicroBlaze.

i) Type *"con"* and press enter; this will start the software program.

j) Run client application on server by navigating to *<workspace dir>/client_host_app/Debug* and typing *"./client_host_app"* in the terminal. If you used your own computer for client application development, navigate to the location on the server, where the executable file was copied at step 6) g.

k) Then, you should receive the first eight random values.

l) You could stop the MicroBlaze by typing *"stop"* in the *XMD* environment. Also, you can find more useful commands in the MicroBlaze documentation.

m) The following information is to speed-up the programming procedure. You could copy all these commands into a file and after initializing XMD, you can *perform "source myfile.txt".*

References:

[1] Thomas Fischer, "HOPP Driver Generator Documentation", Software Technology Group of the University of Kaiserslautern in cooperation with the Microelectronic Design Research Group of the University of Kaiserslautern, July 4, 2013