



Creating a User-Friendly DUT GUI

Agenda

- What is a GUI?
- System Overview
- Environment Setup
 - Dependencies
 - Running the Program

Note

This section focuses on creating a messaging system between a Python GUI, a C++ server, and an example C++ DUT replicating the functionality of our [FPGA AXI counter](#) in software instead.

What is GUI

A **graphical user interface (GUI)** allows a user to interact with a computer program using a mouse cursor or other pointing devices only. A GUI allows a user to interact with programs without prerequisite knowledge of the underlying system architecture. This is very useful for simplifying functions for use by a general audience.

A GUI program is very different from a program that uses a command line interface which receives user input from typed characters on a keyboard.

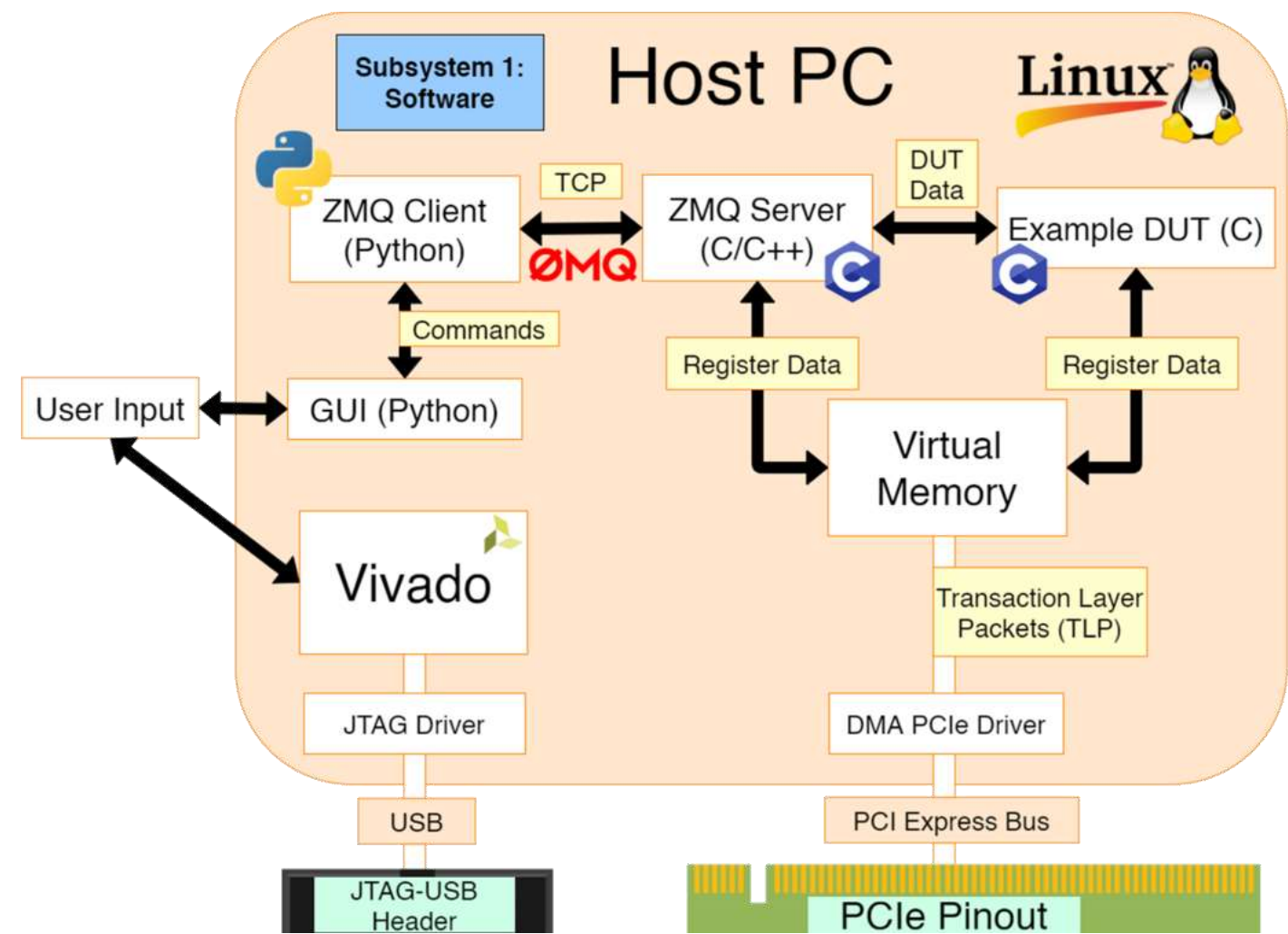
Typically programs that use a command line interface perform a series of tasks in a predetermined order and then terminate, utilizing the Windows or Linux terminal and relying on the user's knowledge of standard Unix syntax. In contrast, a GUI program creates the icons that are displayed to a user and must wait for the user to interact with them. The order that tasks are performed by the program is under the user's control – not the program's control! This means a GUI program must keep track of its own internal state and respond correctly to user commands that are given in any order the user chooses. For example, after setting an initial value for a counter, the GUI must keep track of any subsequent changes to this value if the user decides to increment or decrement the counter.

An GUI program has the following structure:

- Create the icons and widgets that are displayed to a user and organize them inside a screen window.**
- Define functions that will process user and application events.**
- Associate specific user events with specific functions.**
- Start an infinite event-loop that processes user events. When a user event happens, the event-loop calls the function associated with that event.**

Usually, GUIs are created with higher-level programming languages like Java for their rich graphical libraries and ease of use. As such, we will use Python and the simple graphical library Tkinter to create the GUI for our test program.

System Overview



Software system BD

The purpose of this system was to **build a framework that allowed for versatile** communication between the end user and FPGA DUT. Xilinx's DMA/Bridge for PCIe driver arbitrates the mapping of physical memory to virtual memory. This allows for the FPGA board to communicate with the host machine through PCIe by **writing values from physical kernel addresses into the PC's virtual memory**. This framework bridges the gap between the FPGA's stored data in virtual user space and the user themselves. There are many ways to display the FPGA's data to the user, but we chose to **utilize a TCP messaging library called ZeroMQ**. By using **TCP sockets**, the user will be able to write and read to/from virtual memory any amount of data, including PCIe TLPs, binary data, strings, etc.

However, one goal was to test this system without the presence of a board, as the software infrastructure should be able to operate independently without a PCIe device. To accomplish this, we also created **an example DUT in C++ mimicking our simple AXI counter to simulate data transactions within virtual memory**.

Environment Setup

This setup has been tested on both Ubuntu/Debian-based distributions and CentOS/Red Hat. Steps may vary based on the Linux distribution used. Windows and MacOS not tested, although the system should function properly with all dependencies installed.

1. Install CMake.

- For Debian, run `sudo apt-get install cmake g++ make`. Run `apt show cmake` to check if it has installed properly.
- For CentOS, run `sudo yum install cmake`. For all subsequent CentOS commands, `yum` can be substituted for the `dnf` package manager if it is already installed.

2. Install Libsodium.

- For Debian, run `apt-get install libsodium-dev`. If this does not work, you may have to install `libsodium23` or `libsodium18`. Otherwise, download the stable tarball [here](#), unzip it, run `sudo ./configure`, `sudo make`, and `sudo make install` to install Libsodium manually.
- For CentOS, run `yum install libsodium`. You should enable EPEL first (check this [article](#)).
- For CentOS, if `yum` does not work, first run `uname -m` to check if the machine is x86_64 or aarch64. Download the latest release [here](#) (our host machine was x86_64), run `rpm -Uvh libsodium-1.0.18-1.el7.remi.src.rpm` (this is the package filename), and then run `sudo yum install libsodium`.

3. Download the rest of the necessary dependencies. For brevity, we will combine multiple packages together.

- For Debian, run `apt-get install libtool pkg-config build-essential autoconf automake`. Then run `apt-get install libzmq5 libzmq3-dev python3 python3-zmq python3-tk` to install ZeroMQ, Python 3, and Tkinter (if they are not already installed). You can also run `pip install pyzmq` but this is optional.
- For CentOS, run `yum install libtool pkg-config autoconf automake` and then `yum install python3 python3-zmq python3-devel python3-tkinter` if you do not already have Python. Also run `yum install gcc-c++` and `yum install -y ncurses-devel`. Finally, run `yum install zeromq-devel`, which should install `libsodium-devel`, `libunwind-devel`, `openpgm-devel`, and `zeromq-devel`.

⚠ Danger

The source file's `CMakeLists.txt` is currently configured for Debian.

- For CentOS, `libzmq.so` is found in `/usr/lib64` (different from Ubuntu), so after installing all dependencies for CentOS, open `CMakeLists.txt` and edit the line that finds the `libzmq.so` file to `FIND_FILE(ZMQLIB libzmq.so /usr/lib64)`.
- The location of `libzmq.so` will vary, so be sure to use the `find` command in the Terminal (`find /usr -name libzmq.so`).

Running the Program

1. Download and install all dependencies.
2. Download and decompress `gui.zip` into a folder.
3. Within the folder itself, make another temporary folder (this is where your C++ executable will go).
4. Navigate to this temporary folder using the terminal.
5. Compile main.cpp using the command `cmake ..`
 - a. If using **CentOS**, go into another folder on top (like `CMakeFiles`), and copy `zmq.hpp` into the same folder as `CMakeCache.txt` and `cmake_install.cmake` .
 - b. After running `cmake ..` , run `cmake --build ..` in the aforementioned top folder (in this case, `CMakeFiles`) and `./ZmqProject` will be generated in the previous folder (if there is trouble compiling, read this [post](#)).
 - c. If you do not do this, you may get an error about not compiling due to having no cache.
6. Run the C++ server using `./ZmqProject` .
7. In a separate terminal window, run the python script using `python3 client_tk.py` .
8. You should now see a simple blue GUI pop up.
 - a. Type an initial value into the textbox and click Start. You should see the value be set in the C++ server terminal and a reply back to the python client.
 - b. You can click the ++ button to increment the counter by 10, - to decrement the counter by 10, or Stop to close out of the program. With each command, both the server and client should respond to each other (for example, the command "increment" should be sent to the server and the client should receive a reply back that "The counter is at <num>").

DUT Counter

++


--

Stop

Info

About

Initial value set
DUT reply: Start value is 10
DUT reply: The counter is at 11
DUT reply: The counter is at 12
DUT reply: The counter is at 13
DUT reply: The counter is at 14
DUT reply: The counter is at 13



cwu@bec207dl01:~/Documents/GUI

File Edit View Search Terminal Help

```
[cwu@bec207dl01 GUI]$ python3 client_tk.py
Initial value set
DUT reply: Start value is 10
DUT reply: The counter is at 11
DUT reply: The counter is at 12
DUT reply: The counter is at 13
DUT reply: The counter is at 14
DUT reply: The counter is at 13
```

cwu@bec207dl01:~/Documents/GUI/tmp

File Edit View Search Terminal Help

```
Received from client: increment
The counter is at 12
Received from client: increment
The counter is at 13
Received from client: increment
The counter is at 14
Received from client: decrement
The counter is at 13
```

