Intelligent Toasters

General Intelligence on a chip is not far away

UrJTAG and Python – A lightweight HAL?

2014-07-282015-12-28 Intelligent Toasters FPGA Altera, FPGA

Following up on my last post on a Virtual UART for the DE0 Nano (https://intelligenttoasters.wordpress.com/2014/07/05/virtual-uart-for-the-terasic-de0-nano/), I've been exploring ways to make the logic available on any PC. The Nano is small and portable enough to carry around and use on any machine, but if you need to install Quartus on it before you can connect through to the logic, it sort of loses it's portability.

This article is another of my trips 'down the rabbit hole'. By that, I mean, when you start something, then find there's a problem or a dependency and you have to tackle *that* before you can tackle what you really wanted? Well, that was this story, but I won't bore you with the details. Needless to say, this article describes how to build the very latest version of UrJtag under Linux and generate the very accessible Python interface.

What's the point of that, you may ask? Well, UrJTAG (https://sourceforge.net/projects/urjtag/) is pretty lightweight, multi platform and provides a Hardware Abstraction Layer (HAL) that we can use for a future Python Virtual UART. You'll still need to install some files on the target machine, but nothing like the gigabytes of the Quartus installation.

Getting started – Build UrJTAG from source

Building from source seems to scare some people, but it's all about making sure you have the right selection of tools and the correct version of each. The pre-requisites for this installation are:

1. A linux installation (sorry Windows users, I'll come back to you another time)
2. Installed Bison 3.0.2, Flex 2.5.35, Python 2.7.6, autotools-dev, libftdi-dev plus the other dependencies in section 2.4 (http://urjtag.org/book/_compilation_and_installation.html) of the documentation (http://urjtag.org/book/_compilation_and_installation.html) (gettext, readline-dev)
3. If you need it, you can also install the FTDI D2xx library (http://www.ftdichip.com/Drivers/D2XX.htm)
4. You'll also need SVN installed to download the source files

First, get the source from sourceforge:

```
svn checkout -r 2041 svn://svn.code.sf.net/p/urjtag/svn/trunk urjtag-svn
```

You may find that a later version has been posted (check the web site (https://sourceforge.net/p/urjtag/svn/2041/log/?path=)), but these instructions are known to work on commit 2041 (revision 2041).

Next, go into the build directory and run autogen.sh to create the config.

```
cd urjtag-svn/urjtag
./autogen.sh
```

This locates all of the dependencies and settings needed to build the executable and creates 'Makefiles' and writes configuration files ready for the build itself.

Now, this is where I fell down the "Rabbit Hole". Commit 2041 didn't work on my set-up. After building, all seemed fine until I tried to read a BSDL file, or parse an SVF file. These would consistently fail with SEGFAULTs. With the help of another SourceForge user, Cioma, we managed to track the problem down to three lines of code that needed changing. To cut a long story short, you need to apply this patch (https://docs.google.com/uc?export=download&id=0B-itgUKokF0iYVUyUnlld2RFd0E) to make the release work.

To apply, download the patch into the same directory as the autogen.sh file and type:

```
patch -p1 < urpatch.txt
```

You should see four files are patched. Now we can follow the rest of the build instructions:

```
make
sudo make install
```

Unless you received any error messages, well done! UrJTAG is installed and ready.

Starting up UrJTAG for the first time

Once the build process is complete, you should be able to issue the command 'jtag' and be greeted with

```
UrJTAG 0.10 #2041-patched
Copyright (C) 2002, 2003 ETC s.r.o.
Copyright (C) 2007, 2008, 2009 Kolja Waschk and the respective authors

UrJTAG is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
There is absolutely no warranty for UrJTAG.

warning: UrJTAG may damage your hardware!
Type "quit" to exit, "help" for help.
```

Note that the version number #2041 indicates that it's been patched and is not the standard release.

Check that it's talking correctly to your DE0 Nano with the following:

```
cable usbblaster driver=ftdi
detect
```

If all has worked, you should see something like:

```
IR length: 10
Chain length: 1
Device Id: 00000010000011110011000011011101 (0x020F30DD)
```

You'll probably also get a message about an "unknown part". This is because UrJTAG doesn't ship with the descriptors for the EP4CE22 chip. There are two ways to describe the chip to UrJTAG:

1. Use a BSDL file – Your Quartus project can be configured to output these after compilation of your design.
2. Update the shared UrJTAG reference directories.

We're going with option 2 because the BSDL subsystem isn't available from the Python interface and the Python bindings are what we're really after. The bindings can be found in /usr/local/lib and /usr/local /lib/python2.7/dist-packages. Use the following command to see what's been installed on your system:

```
find /usr/local -name '*urjtag*' -type f
```

This will show you the all-important C and Python libraries, ready for action! The next step is to define the DE0 Nano to UrJTAG so that it knows how to communicate with the device.

Creating the chip definition files

Let's start with the quick and easy way to set these files up. Download this file (https://docs.google.com /uc?export=download&id=0B-itgUKokF0iTnM0SWx3TzhQMTA) into /tmp and patch the UrJTAG shared files:

```
cd /usr/local/share/urjtag
sudo patch -p1 < /tmp/urjtag-descriptors.txt
```

Assuming there are no errors, reissue the 'jtag' command;

```
UrJTAG 0.10 #2041-patched
Copyright (C) 2002, 2003 ETC s.r.o.
Copyright (C) 2007, 2008, 2009 Kolja Waschk and the respective authors

UrJTAG is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
There is absolutely no warranty for UrJTAG.

warning: UrJTAG may damage your hardware!
Type "quit" to exit, "help" for help.

jtag> cable usbblaster driver=ftdi
Connected to libftdi driver.
jtag> detect
IR length: 10
Chain length: 1
Device Id: 00000010000011110011000011011101 (0x020F30DD)
 Manufacturer: Altera (0x0DD)
 Part(0): EP4CE22 (0x20F3)
 Stepping: 0
 Filename: /usr/local/share/urjtag/altera/ep4ce22/ep4ce22
jtag>
```

OK, now UrJTAG knows how to issue commands to the DE0 Nano!

A Python Stub

The next step is to create our Python script to connect to the Nano and issue the correct JTAG commands to pass data in and out of the device.

For now, I have a proof of concept that shows how this would work. Go back to the directory where you downloaded and built UrJTAG. Download this file (https://docs.google.com/uc?export=download&id=0B-itgUKokF0iQXlJVUdLWUl4MVk) and this file (https://docs.google.com/uc?export=download&id=0B-itgUKokF0iWDBOUVhVOXlTcFE) into urjtag-svn/urjtag/bindings/python and issue the following:

```
python my.py
```

command. The rather flaky programming will download the example application to the DE0 Nano and shift the data bits out of the device and display them as printable characters. The sample script results in:

```
python my.py
<urjtag.chain object at 0x7f148b5bf120>
urc.cable done <urjtag.chain object at 0x7f148b5bf120>
urc.test_cable done
frequency was 12000000 Hz
urc.detect done
chainlength=1
id[0]=020f30dd
warning: USB-Blaster frequency is fixed to 12000000 Hz
warning: unimplemented mode 'ABSENT' for TRST
BSR dr_in result: 0
BSR dr_out result:
00100100100100100011111100011100001111111111111111000111011010011011011011111111001
00100100100111111111111100000100111111111111100100111111111111111111111001001001111
11111111111111100100111111100100111111111111111111111001111001111111001001111010011
01101100100100111100100111111111111111111110010010011111111111110010011111110010011111
11111100111111100100100100100111111111111100100111111111100111111111100111111111111111
11100111100101101101101101101101001101100100111111111100011111100100111100111111001
00111111111111111111110000001111111111110011111110001111111010010010010010011111111111
11111100100100111100100100101101101101100100111111110011111111111111001001111111111111
00100100111111111111100111111111111100111111100100111111111001
IDREG dr result: 020F30DD
USERCODEREG dr result: DEADBEEF
Status: 00000110
WWelcome to the DE0-Nano - the simplest Virtual JTAG UART in the
Done!
A
B
Status: 00000101
```

(Yes, I know there's a bug that prints the 'W' twice in Welcome).

Have a look at the code to see how it all works! If you're familiar with the Altera Virtual JTAG SLD commands, USER0 and USER1, then you should recognize line 69 onwards.

That's as far as I'll be going with the Python libraries, my primary objective was to get the C libraries so that I can build straight C code to interface to the Nano. (I couldn't face learning yet another interpreted language!)

If you have any success expanding the Python script, please be sure to let me know!

2 thoughts on "**UrJTAG and Python – A lightweight HAL?**"

1. **Retro CPC464 Dongle | Intelligent Toasters** says:
   2015-12-28 at 19:15

   […] two channel devices can be configured for JTAG and UART simultaneously. urJTAG, which I've written about before will be used as the programming device. In the first prototype, I'll likely bring out the […]

   Reply (https://intelligenttoasters.blog/2014/07/28/urjtag-and-python-a-lightweight-hal/?replytocom=14#respond)
2. **Retro CPC Dongle – Part 10 | Intelligent Toasters** says:
   2016-10-16 at 14:25

   […] this version won't be able to program the FPGA out of the box with standard utilities such as URJTAG, so on the bottom layer the JTAG connections are very important to test the FPGA configuration […]

Reply (https://intelligenttoasters.blog/2014/07/28/urjtag-and-python-a-lightweight-hal/?replytocom=40#respond)