Getting started with OpenRISC on the SocKit

Hans Baier*

hansfbaier@gmail.com

Abstract

Getting started with OpenRISC can be quite daunting for beginners. There is so much to learn. Fortunately the community is very helpful. In order to diminish duplication of effort, this tutorial was conceived out of the generous hours of support that Stefan Kristiansson provided over IRC. This tutorial also provides some of the supplemental files, for convenience.

I. Introduction

OpenRISC is an amazing learning plattform: You can run Open Source Software on Open Source Hardware and see down to the metal, how hardware is designed, drivers are written, and how software and hardware interact.

II. Prerequisites

This tutorial assumes you are using a GNU/Linux Distribution. This tutorial is based on Ubuntu 12.04 (precise).

- Experience with Linux systems
- Compiling software from source
- Basic understanding of digital logic, communication protocols and electronics.
- Familiarity with Verilog and C

You will need a Linux workstation with a 64 Bit operating system, 8GB of RAM and about 20-30GB free hard drive space. You will also need terasic's SocKit Development Board.

III. Step By Step

1. Install Altera Quartus II Web Edition at least version 13.0sp1. For this tutorial we assume you installed it in `/opt/altera/`

2. Download the resources for the development board. Find out what board revision you have (first PDF on the resources page), and download the SocKit System CD for the correct board revision. This will be most likely be Revision C at the time of this writing:

   ```
   $ wget -c http://www.terasic.com/downloads/cd-rom/sockit/SoCKit_V.1.0.0_System.zip
   ```

*Thanks to Stefan Kristiansson for his generous help and Kevin Mehall for his excellent tutorial on the de0_nano

3. Install development tools:

```
$ sudo apt-get install build-essential libmpc-dev libgmp3-dev libmpfr-dev  \
                       lzop libsdl1.2-dev xterm automake libtool \
                       git-core git subversion iverilog device-tree-compiler
```

4. Create the directory where we put openrisc related files:

```
$ mkdir ~/openrisc
```

5. Create a file with necessary environment variables:

```
$ cd ~/openrisc
$ editor altera_env.sh
```

```
export ALTERA_PATH=/opt/altera
export PATH=$PATH:$ALTERA_PATH/quartus/bin
```

```
$ source altera_env.sh
```

   You may want to source this file every time you want to use Altera's tools.

6. Unpack the SocKit CD:

```
$ mkdir CD
$ unzip ../SoCKit_V.1.0.0_System.zip -d CD
```

7. Get `orpsoc` which is the OpenRISC-SoC-Builder and `orpsoc-cores`, which contains the IP-cores and board configurations:

```
$ git clone https://github.com/openrisc/orpsoc.git
$ git clone https://github.com/skristiansson/orpsoc-cores.git
```

8. Build and install OrpSoC:

```
$ cd orpsoc
$ autoreconf -i && ./configure && make && sudo make install
```

9. Configure orpsoc-cores and test:

```
$ cd ../orpsoc-cores/
$ editor orpsoc.conf
```

```
[main]
cores_root   = ./cores
systems_root = ./systems
```

```
$ orpsoc sim wb_bfm
Preparing wb_bfm
Preparing vlog_tb_utils
All tests passed!
```

10. We need to start a first run on building the sockit. It will fail, because the Qsys infrastructure Verilog files don't exist yet. We need this run, though, since the target directory for the Qsys files does not exist yet:

    ```
    $ orpsoc build sockit
    ```

11. We now need to generate the infrastructure Verilog files for the HPS system. Fire up Quartus:

    ```
    $ quartus &
    ```

12. Close the splash screen and start Qsys with: Tools→Qsys
13. Open Qsys file with: File→Open
14. Choose File: `systems/sockit/data/sockit.qsys`
15. Klick on the 'Generation' Tab and choose for Output Directory (relative to orpsoc-cores): `build/sockit/src/qsys/`

    This takes a while. Get a glass of orange juice (healthier than coffee) or do a couple of pushups in the meantime.
16. When Generation is finished successfully, close the dialog, save the Qsys (then the output path will be right when you choose to alter the Qsys and need to rerun again).
17. Close Qsys and Quartus and build the FPGA bitstream:

    ```
    $ orpsoc build sockit
    ```

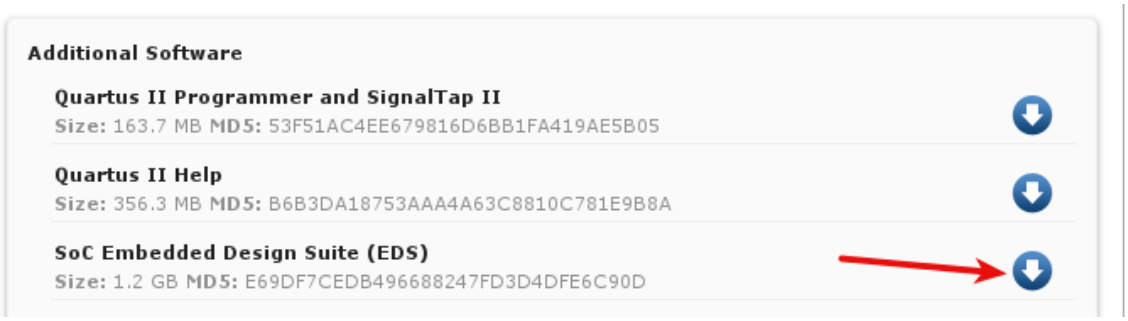
    Once more, this takes quite a while, depending on your machine. Time for another glass of orange juice and some pushups.

    If all went well, then you should have the file `build/sockit/bld-quartus/sockit.sof`. This is the bitstream for the FPGA:

```
$ find . | grep sof$
./build/sockit/bld-quartus/sockit.sof
```

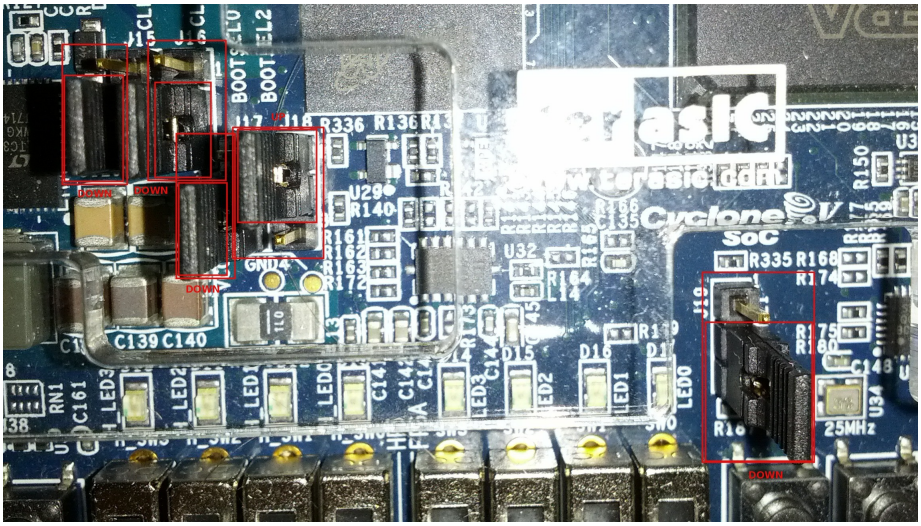
18. If you are done with the pushups, and it is still compiling, you can continue with the following steps until you need the Bitstream (`.sof`).

19. Download the Altera Embedded Design Suite (DS-5) Web Edition. Select the tab 'Individual Files' and klick the Download link for the Embedded Design Suite.

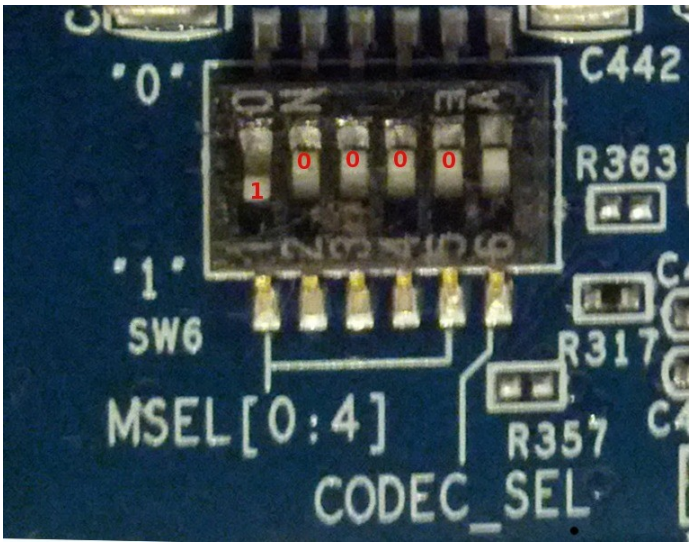


20. Install it to `/opt/altera/` as well. If everything went right, you should have a subdirectory `/opt/altera/embedded/`. This is the installation location of the embedded design suite.

21. Setup you SocKit Board. The Clock and Bootsel Jumpers need to look like this:

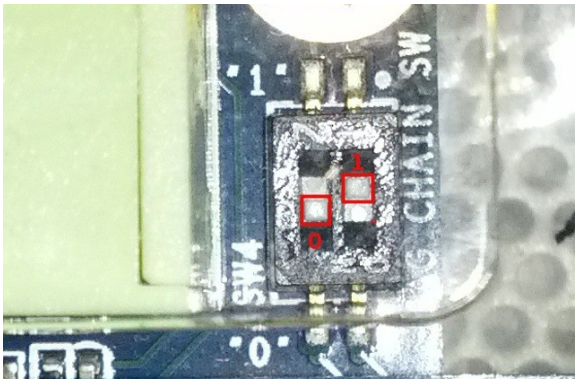


22. In order to boot Linux from the ARM-HPS, the MSEL dipswitch has to be set to passive serial (10000) like this:



Note that '0' on the board corresponds to 'ON' on the DIP-switch. Trust the board, not the switch. The last DIP-switch is probably irrelevant, I didn't bother to find out, you can let it how it came in the box. Don't blame me if your board catches fire or kills your cat.

23. The JTAG-DIP-switch should be set to bypass the HSMC connector and to enable the HPS. (That's '01' in the orientation of the silk screen print):



24. Make sure the USB Blaster II integrated on the board has proper permissions.

```
$ editor /etc/udev/rules.d/81-usbblaster.rules
```

```
# USB-Blaster II
SUBSYSTEM=="usb", ATTRSidVendor=="09fb", ATTRSidProduct=="6010", MODE="0666"
SUBSYSTEM=="usb", ATTRSidVendor=="09fb", ATTRSidProduct=="6810", MODE="0666"
```

I had to put two versions of the `idProduct` into the file, since the product id mutated since first plug in. If in doubt, check your Blaster with `lsusb` and alter `idProduct` accordingly.

25. Start the Altera embedded command shell:

```
$ /opt/altera/embedded/embedded_command_shell.sh
```

26. Start the BSP-Editor to create a board support package for the Qsys generated platform;

```
$ bsp-editor
```

27. Klick Menu: File→New BSP
28. For 'Preloader settings directory' choose
    `orpsoc-cores/build/sockit/bld-quartus/hps_isw_handoff/sockit_hps_0`
29. Press 'OK'
30. Press 'Generate'
31. Press 'Exit' if everything went well during generation.
32. Go to the generated sources and build them:

```
$ cd orpsoc-cores/build/sockit/bld-quartus/software/spl_bsp/
$ make
```

    This will extract the `uboot` source code into the subdirectory `uboot-socfpga`.

33. Now if everything went well, we need to modify uboot, so that the wishbone system on the FPGA is reset automatically on boot time:

```
$ editor uboot-socfpga/include/configs/socfpga_cyclone5.h
```

```
    "mmcboot=mw 0xffd0501c 4;setenv bootargs " CONFIG_BOOTARGS \
```

34. Rebuild:

```
$ make all
```

35. Then you are ready to prepare the SD card. You need an SD card large enough (2GB) to accomodate the Factory image. We extract the factory from on the SocKit CD and write it on SD card. Please note, that `/dev/sdX` denotes the device of your sd card.

```
$ cd ~/openrisc/CD/Tools/Factory_SD_image/
$ unrar x SoCKit_SD.rar
$ sudo dd if=SoCKit_SD.img of=/dev/sdX bs=512
$ sudo sync
$ sudo eject /dev/sdX
```

36. The layout should contain three partitions like this:

```
$ sudo fdisk -l /dev/sdX

   Device Boot      Start         End      Blocks   Id  System
/dev/sdX1         2121728     2162687       20480    b  W95 FAT32
/dev/sdX2           14336     2111487     1048576   83  Linux
/dev/sdX3            2048        4095        1024   a2  Unknown
```

    Note there is one FAT-Partition. This one contains the device tree and the uboot kernel image (`uImage`). The second partition (Linux) contains the Linux root filesystem. The third partition (type `a2`) contains the bootloader and uboot. It has no filesystem on it.

37. Before we modify the SD card, you might want to verify it boots (without our openrisc FPGA image flashed). To do this,

    (a) connect the USB serial cable to your Host PC

    (b) Insert the SD card into the SoCKit

    (c) Power the board on

    (d) Fire up a terminal program of your choice on the virtual USB terminal (we assume `/dev/ttyUSB0`):

    ```
    $ picocom /dev/ttyUSB0 -b 57600
    ```

    You might want to use `sudo` here if you don't have an appropriate `udev` entry or you are not a member of the `dialout` group. Do this quickly after the board is powered on to catch as much of the boot messages as possible.

    It should boot uboot and then Linux. If that is the case, you are fine for now. If not, you have a problem...

38. If Linux boots fine, switch off the device (we will replace the Linux system on it)

39. We will replace the Yocto Linux system with a Debian system (optional, but needed for convenient software install):

```
$ wget -c
http://www.rocketboards.org/pub/Projects/Debian/debian.img.gz
```

```
$ gunzip debian.img.gz
$ sudo dd if=debian.img of=/dev/sdX2 bs=512
```

You can also use an ubuntu based image, like Stefan:

```
$ wget -c \
https://releases.linaro.org/13.04/ubuntu/quantal-images/developer/linaro-quantal-developer-20130422-342.tar.gz
```

but this image is unsupported (by me) since I did not try it.

40. Next thing is to build the Linux kernel for the ARM based HPS system:

```
$ cd ~/openrisc
$ mkdir sockit
$ mv CD sockit
$ cd sockit
$ git clone --depth 1 git://git.rocketboards.org/linux-socfpga.git
$ cd linux-socfpga
$
```

References

[rocketboards] www.rocketboards.org Resource website for SoCKit and related boards. Lots of useful information and software.