Tutorial on

PORTING OF PYTHON AND MODULES ON SHAKTI

By:
KAPIL SHYAM. M

Linux Development Kit for Shakti SOC'S

The Development kit acts as a "One Stop" means to build and deploy linux on a specific Shakti capable SOC.

To Learn more about the Shakti and its available SOC , details can be found here at (http://shakti.org.in/)

Getting Started:

Setting up the Build Environment

1. Get the latest RISCV-Toolchain

This repository uses submodules. You need the --recursive option to fetch the submodules automatically

```
$ git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
```

Alternatively :

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
$ cd riscv-gnu-toolchain
$ git submodule update --init --recursive
```

Install the necessary packages

Several standard packages are needed to build the toolchain. On Ubuntu,
executing the following command should suffice:

```
$ sudo apt-get install autoconf automake autotools-dev curl libmpc-dev
libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool
patchutils bc zlib1g-dev libexpat-dev
```

Install the toolchain by doing

To build the Newlib cross-compiler, pick an install path. If you choose, say, ***/opt/riscv***, then add ***/opt/riscv/bin*** to your **PATH** now. Then, simply run the following commands,

```
$ cd riscv-gnu-toolchain
$ ./configure --prefix=/opt/riscv --with-arch=rv64imac --with-abi=lp64
--with-cmodel=medany
$ sudo make && sudo make linux
```

Once the above steps are completed you can find the cross-compile and Bare Metal binaries in the respective path '***/opt/riscv/bin***' also assuming you have added it to the **PATH** variable you can proceed to build linux.

Getting started on linux Devlopment:

The linux development repository uses submodules. You need the --recursive option to fetch the submodules automatically

```
 $ git clone --recursive
https://gitlab.com/shaktiproject/software/linux-devkit.git
 $ git checkout py-support (or) git checkout fpu-support
```

The development package Supports C-Class 64bit Core,which boots ***Linux*** on top of Proxy Kernel.

Rapid deployment using BBL as a bootloader

Linux can be built as a payload to BBL by doing the below command. For instance Shakti C-Class is based on RV64IMAC. So alter the config file.

Location : <Your linux-devkit dir>/buildroot/package/busybox/busybox.config

```
CONFIG_EXTRA_CFLAGS="-g -march=rv64imac -mabi=lp64"
CONFIG_EXTRA_LDFLAGS="-g -march=rv64imac -mabi=lp64"
```

Now, while building the linux-kernel, there is a possibility that you might get an error on *“multiple declarations of yylloc”* . For avoiding this error, in linux-devkit/linux-on-shakti/scripts/dtc/dtc-lexer.l , modify line number 26 as,

```
extern YYLTYPE yylloc;
```

Now, while building the bbl, there is a possibility that you might get an error like below:

```
../pk/pk.c:137:3: error: both arguments to '__builtin___clear_cache' must be pointers
```

For avoiding that, in linux-devkit/bootloaders/riscv-pk/pk/pk.c , modify line number 137 as,

```
asm volatile (“fence.i”);
```

Also once the above is done, open new terminal, and do the following commands and close the terminal:

```
$ cd linux-devkit/buildroot
$ git checkout 2021.05.x
```

Now, here, the buildroot_initramfs_config file is given. Copy this file and paste it in the linux-devkit/bsp/conf folder. Here, in this file, Python interpreter and pip will be set as default. So that, while building the bbl, the python and pip gets compiled by the buildroot by default, and will be able to use it on Shakti once the bbl gets generated.

Adding Python Packages to run on Shakti:

Now, to add python packages to run on Shakti, we must know from where the linux gets built from. The linux gets built using the ***BuildRoot***. And, for *Shakti* to support Python, we are using this specific version of buildroot, *Version 2021.05.x*.

That *buildroot_initramfs_config* file under *linux-devkit* supplies the instructions on what to be built and what should not. This file now contains the maximum number of python modules which can be built using BuildRoot. As of now, we have just set a few modules of python, which are considered as necessary to get loaded while the bbl is getting built. Now, if you think of having some other Python module will be useful for your work, then, it is easy for you to have one. For having the python module built, go to */linux-devkit/bsp/buildroot_initramfs_config* and go to the section named ***`Interpreter languages and scripting`***, and scroll till you see ***`External python modules`*** . Here, you will see that set of python modules which can be built. For having the python modules, listed here, you must search for that module and if it is present, just modify the line as given in the following example.

Example 1: Need of Python module named pyopenssl. For this, check for the line containing *BR2_PACKAGE_PYTHON_<PACKAGE_NAME>* For this example, search for line *BR2_PACKAGE_PYTHON_PYOPENSSL.* This line will be commented, and mentioned as *"is not set"* in the *buildroot_initramfs_config* file.

You need to change this from:

# BR2_PACKAGE_PYTHON_PYOPENSSL is not set to

BR2_PACKAGE_PYTHON_PYOPENSSL=y

That's all. Now, ***Save*** this file and start to build the bbl.

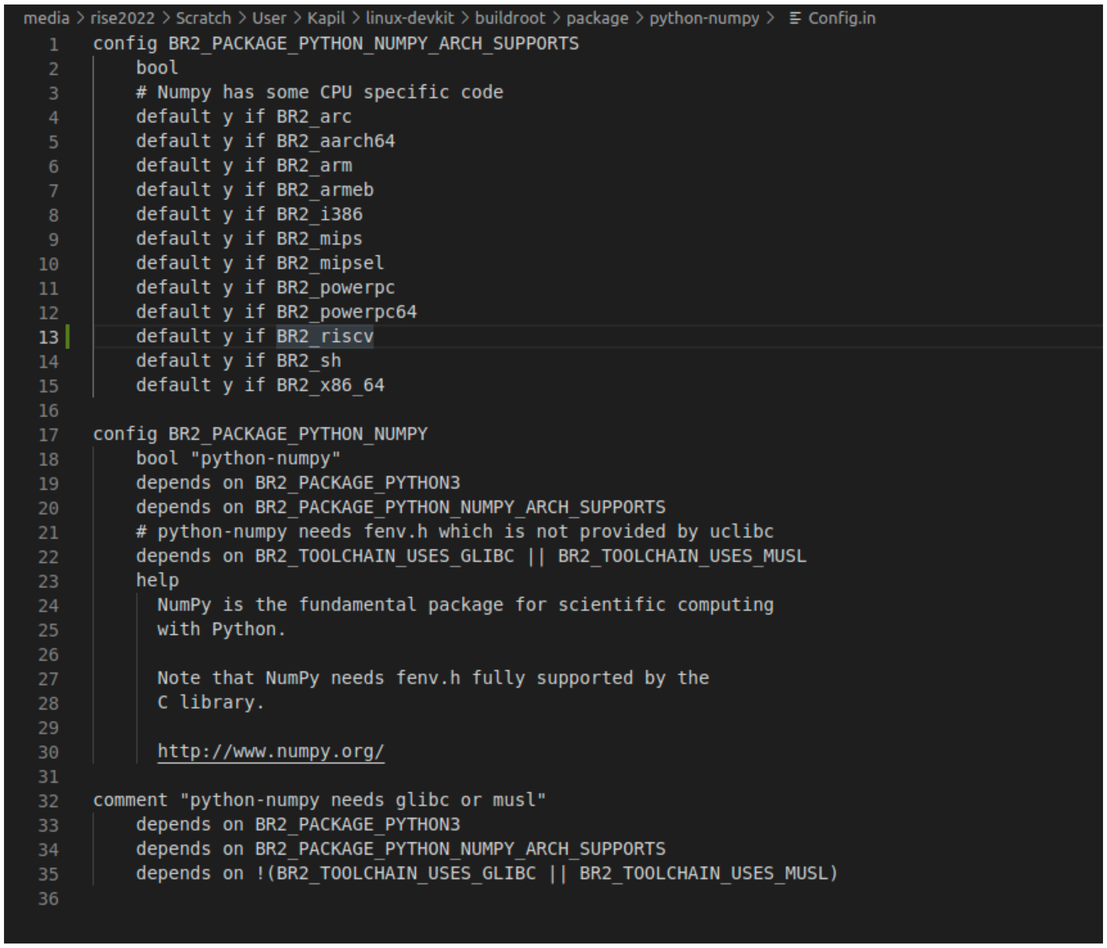
For building the bbl, do the following commands.

```
$ cd ~/linux-devkit/
$ make bbl
```

Adding NUMPY Package to run on Shakti:

Till now you would have got an idea on how to build python to run on Shakti. Now, for having Numpy, we need to configure the config file of the Numpy package on BuildRoot. For this goto *~/linux-devkit/buildroot/package/python-numpy/Config.in* and add the below line to line number 13 of that file as shown in the figure below:

default y if BR2_riscv



media > rise2022 > Scratch > User > Kapil > linux-devkit > buildroot > package > python-numpy > Config.in

```
1   config BR2_PACKAGE_PYTHON_NUMPY_ARCH_SUPPORTS
2       bool
3       # Numpy has some CPU specific code
4       default y if BR2_arc
5       default y if BR2_aarch64
6       default y if BR2_arm
7       default y if BR2_armeb
8       default y if BR2_i386
9       default y if BR2_mips
10      default y if BR2_mipsel
11      default y if BR2_powerpc
12      default y if BR2_powerpc64
13      default y if BR2_riscv
14      default y if BR2_sh
15      default y if BR2_x86_64
16
17  config BR2_PACKAGE_PYTHON_NUMPY
18      bool "python-numpy"
19      depends on BR2_PACKAGE_PYTHON3
20      depends on BR2_PACKAGE_PYTHON_NUMPY_ARCH_SUPPORTS
21      # python-numpy needs fenv.h which is not provided by uclibc
22      depends on BR2_TOOLCHAIN_USES_GLIBC || BR2_TOOLCHAIN_USES_MUSL
23      help
24        NumPy is the fundamental package for scientific computing
25        with Python.
26
27        Note that NumPy needs fenv.h fully supported by the
28        C library.
29
30        http://www.numpy.org/
31
32  comment "python-numpy needs glibc or musl"
33      depends on BR2_PACKAGE_PYTHON3
34      depends on BR2_PACKAGE_PYTHON_NUMPY_ARCH_SUPPORTS
35      depends on !(BR2_TOOLCHAIN_USES_GLIBC || BR2_TOOLCHAIN_USES_MUSL)
36
```

Now, follow the same steps as mentioned in the above example ***Example 1***. Search for *BR2_PACKAGE_PYTHON_NUMPY* ,uncomment it, and set it *'=y'* . Now Save the file, and make bbl.

Using SOC to Boot Linux

Currently the linux kernel boots on ARTY A7 100t with C-Class.

Assuming you have programmed the board and ready to deploy the bbl follow the below steps.

Open Three terminals,

1. Miniterm
2. OpenOcd
3. RISC-V GDB

->Connect to the board using openocd with shakti-sdk

```
$ cd ~/shakti-sdk/bsp/third_party/vajra
$ sudo $(which openocd) -f ftdi.cfg
```

->Connect to gtkterm or minicom or miniterm with a baudrate of 19200 and port as /dev/ttyUSB1

```
$ sudo miniterm.py /dev/ttyUSB1 19200
```

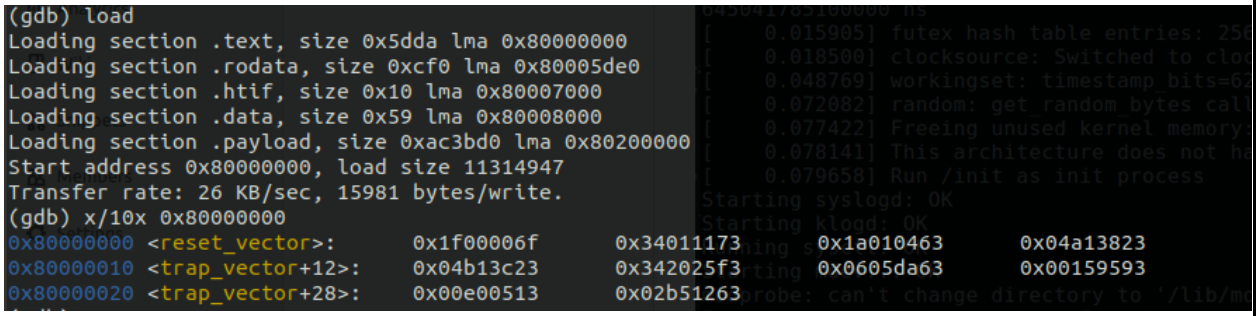
->Using gdb(riscv64-unknown-elf-gdb) load the bbl as shown

```
(gdb) set remotetimeout unlimited
(gdb) target remote localhost:3333
(gdb) file path/to/linux-devkit/bootloaders/riscv-pk/build/bbl
(gdb) load
```

Once done inspect the memory at 0x80000000 to check if the image is loaded properly.

(gdb) x/10x 0x80000000

GDB after load,



```
(gdb) load
Loading section .text, size 0x5dda lma 0x80000000
Loading section .rodata, size 0xcf0 lma 0x80005de0
Loading section .htif, size 0x10 lma 0x80007000
Loading section .data, size 0x59 lma 0x80008000
Loading section .payload, size 0xac3bd0 lma 0x80200000
Start address 0x80000000, load size 11314947
Transfer rate: 26 KB/sec, 15981 bytes/write.
(gdb) x/10x 0x80000000
0x80000000 <reset_vector>:      0x1f00006f      0x34011173      0x1a010463      0x04a13823
0x80000010 <trap_vector+12>:    0x04b13c23      0x342025f3      0x0605da63      0x00159593
0x80000020 <trap_vector+28>:    0x00e00513      0x02b51263
```

->Hit Continue to get the output. Output is displayed in Serial Monitor (Miniterm).

(gdb) c

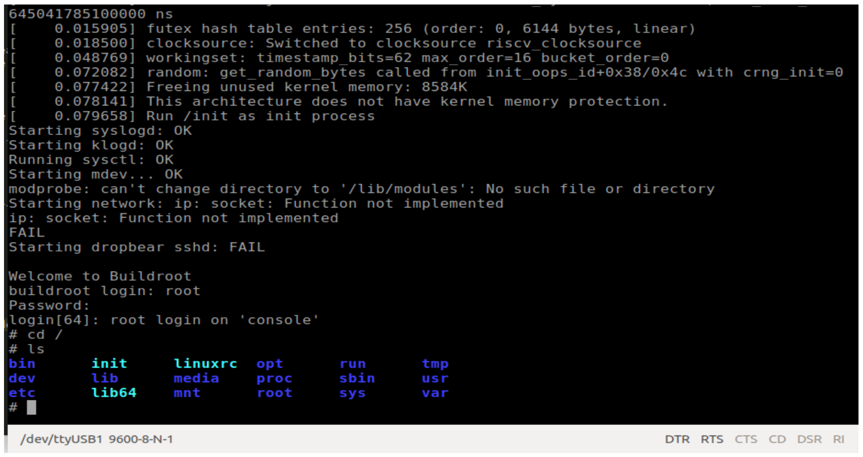
Login details are:

Login ID : root

Password : shakti

One can use "adduser" to add new users .

Linux with minimal filesystem (miniterm)



```
645041785100000 ns
[    0.015905] futex hash table entries: 256 (order: 0, 6144 bytes, linear)
[    0.018500] clocksource: Switched to clocksource riscv_clocksource
[    0.048769] workingset: timestamp_bits=62 max_order=16 bucket_order=0
[    0.072082] random: get_random_bytes called from init_oops_id+0x38/0x4c with crng_init=0
[    0.077422] Freeing unused kernel memory: 8584K
[    0.078141] This architecture does not have kernel memory protection.
[    0.079658] Run /init as init process
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Starting mdev... OK
modprobe: can't change directory to '/lib/modules': No such file or directory
Starting network: ip: socket: Function not implemented
ip: socket: Function not implemented
FAIL
Starting dropbear sshd: FAIL

Welcome to Buildroot
buildroot login: root
Password:
login[64]: root login on 'console'
# cd /
# ls
bin       init      linuxrc   opt       run       tmp
dev       lib       media     proc      sbin      usr
etc       lib64     mnt       root      sys       var
# 
```