Juno Board Connection & Flashing Workflow

This Confluence page references this guide by the ARM community site

1 - Board to Host PC Connection Setup

My working setup of the Juno board involves connecting:

- Power Cable to the Board
- Serial Port (board) to USB-A (Host PC) for command line terminal
- USB-B (board) to USB-A (Host PC) for flashing kernels to filesystem

Putty was used to connect to the board through its serial port.

Connect it using the Serial Line. Use the following settings:

- 115200 baud
- 8 - bit word length
- No parity
- 1 stop bit
- No flow control

Windows or Linux can be used to connect. If on Windows, look at which COM port the Juno board is connected to.

Otherwise, on Linux, see which port corresponds to the device by issuing `lsblk` command on local terminal.

2 - Flashing to the Board (needs more details)

Be sure to connect to the board first before doing this task. Additionally, this section assumes that the **Building from Source** option is chosen with the BusyBox filesystem, as outlined in this confluence page, successfully completing the `apt` and `pip2` module installations, and building from source the intended images.

1. Mount the Juno Board through USB as a mass-storage device.
2. Copy the `recovery` folder contents contained within the `arm-platforms-deliverables` to the root directory of the Juno Board
3. Then copy the files from the `/output/` folder corresponding to the configuration chosen .(e,g. `/arm-reference-platforms/output/juno/juno-busybox/`) to the `/SOFTWARE` directory of the Juno Board.
4. Once finished copying, soft-reset the board and see the kernel in action. Check if it's properly booted by the `uname -a` terminal command

Only Currently Known Supported Kernel(s)

- The plain BusyBox Kernels (either prebuilt or built from source) currently work on the Juno. It's a Linux 5.3 Kernel
- Recently figured out how to flash a 5.4 Linux Kernel to the Juno R2 board; steps as follows:

Flashing a Working 5.4.207 LTS Linux Kernel to the Juno R2

1. Copy the contents of the `.config` file in this gist
2. Download a 5.4.207 kernel tree from kernel.org
3. Copy the `.config` file from the gist to the downloaded kernel tree
4. Ensure you downloaded a cross-compiler from this ARM GNU toolchain website, I used `aarch64-none-linux-gnu`
5. Set the appropriate environment variables:

   ```
   export ARCH=arm64
   export CROSS_COMPILE=/usr/local/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
   ```

   NB. the `CROSS_COMPILE` directory is specific to your installation folder.

6. Run the following commands to update the `.config` file to work with the newer kernel:

   ```
   make ARCH=$ARCH CROSS_COMPILE=$CROSS_COMPILE oldconfig
   ```

   (when asked if you'd like to add new features, keep saying **No**)

   ```
   make ARCH=$ARCH CROSS_COMPILE=$CROSS_COMPILE all
   ```

7. Flash as outlined in this page, i.e by copying `Image` from the `/arm64/boot/` directory, and the Juno `.dtb` files located in the `/arm64/boot/dts/arm/` (leaving the `.dts` files out)
8. Power cycle the Juno Board, let it boot, see the working Kernel by issuing the `uname -a` command

Known Building/Flashing Issues

Linux Script Building Issues *(Temporary Workaround Found)*

When running the `build-linux.sh` shell script like so:

```
build-scripts/build-linux.sh clean
build-scripts/build.linux.sh build
build-scripts/build.linux.sh package
```

You may potentially find errors on lines 79 and 129, requiring a numerical operator. A temporary workaround this by **adding an extra pair of brackets** (brackets are these []) to the if-statements. The script *should* now run successfully.

Lack of Support for Alternative Kernel Selections *(Only BusyBox and Linux Works)*

The only working kernel on the Juno Board is relatively simple BusyBox kernel, either built from source or using the pre-configured builds.

The overall workflow of attempting to flash the kernels to the board are the same, as outlined above.

Below is a table of kernels that I have failed to flash to the Juno Board:

Kernel Type	Kernel Building Method	Outcome	Additional Notes
OpenEmbedded	*bitbake* command: `bitbake core-image-minimal`	Failed - Kernel Panic	
	selecting pre-configured OE configuration during `python3 sync_workspace.py` from the `arm-reference-platforms`	Failed - Kernel Panic	
	Followed the directions up until the `pre-build-root-install-dependencies.sh` step. Failed building - bitbake command not found (presumably packages don't work anymore?)	Failed building	from this arm community webpage on building OE for Juno some packages don't work anymore and tool outlined that dependencies may be broken, even tried running this on Ubuntu 14.04 and 16.04
Linux LTS 5.4 Linux LTS 5.10	set environment variables for `ARCH=arm64` and `CROSS_COMPILE` to the cross compiler installation directory, ran `make menuconfig` , then ran `make`	Failed - bad kernel	from kernel.org website. disabled the PCIE, soundcard, and graphics card driver requirements from the setup. had to delete `CONFIG_SYSTEM_TRUSTED_KEY` , `CONFIG_MODULE_SIG_KEY` lines and set `CONFIG_DEBUG_INFO_BTF` line to `n` at the `.config` file to build the compiler
Android	selecting pre-configured android configuration during `python3 sync_workspace.py` from arm-reference-platforms	Failed - Stuck in a Boot Loop!	burnt image to a USB stick, followed the instructions on running the deliverables on the Juno platform

Currently Working Setup - OpenEmbedded FS

OpenEmbedded can work on my particular copy of the Juno Board, including ethernet support and booting file system from USB.

Steps to Flash OE:

1. Install Open Embedded RootFS on a USB Stick

These instructions assume you are running on a Linux Machine

1. Download the VExpress OpenEmbedded Filesystem Image from the linaro releases portal
2. Extract the .gz to be an .img file
3. Format the USB Stick you'll be using to an `ext4` filesystem
4. *Zero* the USB stick using `dd`

   ```
   sudo dd if=/dev/zero of=/dev/sdX bs=1M status=progress
   ```
5. Then install using dd as well:

   ```
   sudo dd if=lt-vexpress64-openembedded ... .img  of /dev/sdX bs=4M status=progress
   ```

2. Build OpenEmbedded for UBOOT from Source using Arm Platforms Deliverables

NB: Can also use prebuilt deliverables from Linaro during the Arm Platforms Deliverables Setup

1. Follow the instructions from this confluence page on cloning & setting up the Arm Reference Platforms
2. Either choose the prebuilt configurations for OpenEmbedded Minimal Uboot, or opt to build from source
3. Copy the files from the output folder to the Juno mounted storage device

3. Insert USB Stick to the back of the Juno Board, Power the Board and Connect the Board through Serial

Running CSAL Repository Demos

These steps are based on this readme file from the original CSAL repository on running demos.

1. Clone the repository
2. Refer to the file `/demos/juno_demo_setup/no_cpu_idle.sh` or create a file with the following contents with a suitable name (e.g. no_cpu_idle.sh)

   ```
   #!/bin/sh
   for i in 0 1 2 3 4 5;
           do for j in 0 1 2 ;
                   do echo 1 > /sys/devices/system/cpu/cpu$i/cpuidle/state$j/disable;
           done;
   done
   ```
3. This script needs to be run on each boot. To automate the process, the first step is to copy the file into `/etc/init.d/`
4. Run the following command to create an execute on boot link

   ```
   ln -s /etc/init.d/YOUR_FILE_NAME /etc/rc5.d/S99_disable_cpuidle.sh
   ```
5. Reboot and confirm that idle is disabled by checking one of the cores (not core0)

   ```
   cat /sys/devices/system/cpu/cpu3/cpuidle/state1/disable
   ```

Sometimes steps 2-4 can't run successfully, so an alternative would be to run the above `.sh` file on system boot.

Then, on the CSAL repository root directory, run `make && make DEBUG=1`

Run the applications in the `/bin/arm64/rel/` directory, with the `-board-name Juno` option. e.g.:

```
./tracedemo -board-name Juno
```

Currently known Issues:

Everything can be built on the board. But running the programs after being Cross-Compiled doesn't work on the Juno Board

The GLIB6 Library for running cross-compiled software is Missing, and cannot be installed without apt

Apt itself is missing.

The Kernel needs to be rebuilt with `CONFIG_DEVMEM_STRICT=n` as the `./tracedemo` demonstration program cannot use mmap. This is a work in progress.

Cross-Compilation of CSAL from x86 to ARM64 Juno Board

(introduce cross-compilation concept here)

According to the `makefile` at the root directory of the CSAL repository and the docs, an environment variable of `CROSS_COMPILE` is required to get the library to work on the Juno board. That being, I believe it's a fair assumption that a cross-compiler is needed to make CSAL work on the board.

The cross-compiler used was this that is referenced in this arm GNU toolkit webpage.

1. Clone the CSAL repository.
   ```
   git clone https://github.com/ARM-software/CSAL.git
   ```
2. Download & Install the cross-compiler:
   ```
   wget https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
   tar -xJf gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz -C /usr/local/
   ```
3. Set the appropriate environment variables & run the makefile
   ```
   export CROSS_COMPILE=/usr/local/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
   export ARCH=arm64
   make
   ```