Build NXP FreeRTOS

Rev 1.0 20170615

Contents

1. Supported hardware

These are the systems covered in this guide:
System-on-Modules:

- PICO-IMX7-SD

Carrier Boards:

- PICO-PI

FreeRTOS BSP is provided to the SOC with Cortex-M core. NXP i.MX7 has Cortex-A7 and Cortex-M4 cores. Cortex-A7 runs Linux, and Cortex-M4 runs FreeRTOS.

2. Get FreeRTOS BSP Source Code

There are two ways that you can get FreeRTOS BSP source code.

2.1 From TechNexion website

Download the source tarball “pico-imx7_pico-pi_FreeRTOS-1.0.1_source_20170615.zip”.

2.2 From TechNexion github

https://github.com/TechNexion/freertos-tn
To get the source code:

```
$ git clone https://github.com/TechNexion/freertos-tn.git
$ cd freertos-tn
$ git checkout freertos_1.0.1_imx7d
```

3. Development environment setup

3.1 Preparation

Download and install toolchain to compile FreeRTOS:

```
$ wget 'https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q3-update/+download/gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2'

$ tar jxvf  gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2
```

GNU make and cmake need to be available on host OS (e.g. Ubuntu):

```
$ sudo apt-get install make cmake
```

3.2 Set an environment variable

Export the environment variable to point to the toolchain location:

```
$ export ARMGCC_DIR=${absolute_path}/gcc-arm-none-eabi-4_9-2015q3/
```

replace '${absolute_path}' with corresponding path.

4. Build an example of FreeRTOS

Here, we take "hello_world" as example.

```
$ cd freertos-tn/examples/imx7d_pico_m4/demo_apps/hello_world/armgcc/
$ ./build_release.sh
```

Copy that "hello_world.bin" firmware to the root of an eMMC/SD to flash it. If you don't have m4 device tree file, you need to copy it. (e.g. imx7d-pico_pi-m4.dtb)

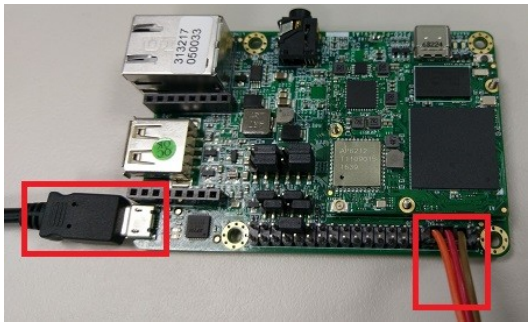
```
$ cp release/hello_world.bin /media/F332-F399/
```

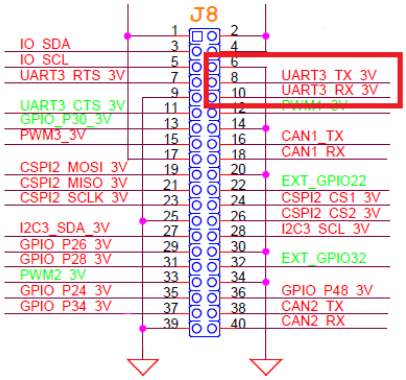
By default, Linux BSP does not come with Cortex-M4 enabled. The device tree needs to be changed in u-boot with commands as below. If there is no ${m4image} firmware, the device tree will not automatically change.

```
=> setenv loadm4image 'fatload mmc ${mmcdev}:${mmcpart} 0x7F8000 ${m4image}'
=> setenv m4boot 'if run loadm4image; then setenv mcu -m4; dcache flush; bootaux 0x7F8000; fi'
=> setenv setfdt 'setenv fdt_file ${som}_${baseboard}${mcu}.dtb'
=> setenv mmcboot 'echo Booting from mmc ...; run m4boot; run searchbootdev; run mmcargs;
echo baseboard is ${baseboard}; run setfdt; if test ${boot_fdt} = yes || test ${boot_fdt} = try; then
if run loadfdt; then bootz ${loadaddr} - ${fdt_addr}; else if test ${boot_fdt} = try; then echo
WARN: Cannot load the DT; echo fall back to load the default DT; setenv baseboard $
{default_baseboard}; run setfdt; run loadfdt; bootz ${loadaddr} - ${fdt_addr}; else echo WARN:
Cannot load the DT; fi; fi; else bootz; fi;'
```

5. Run the demo app

As the picture below, connect these two UARTs to your PC and set the baud rate as “115200 8n1”. Left UART is for linux, and right UART is for FreeRTOS.





In u-boot prompt:

```
=> fatload mmc 0:1 0x7F8000 hello_world.bin
=> dcache flush
=> bootaux 0x7F8000
```

On the Cortex-M4 debug console, you should see the following output:

```
Hello World!
```