Multicore Communication

The i.MX7 processor comes equipped with 2x Arm Cortex-A7 cores as well as a Arm Cortex-M4 core. In this guide, we will modify the device tree to run in parallel with the FreeRTOS running on the M4 core. We will then demonstrate the A7 core's ability to communicate to the M4 core and cause it to effect a change on a GPIO signal. In order to follow this guide you will first need to download the demo tarball from Artifactory and extract the file rpmsg_gpio_toggle_freertos_example.bin.

Step-by-step guide

- Create an additional terminal for UART2 on your host machine.
  - Ensure this second Terminal is configured for 115200 baud, 8 bit data, no parity bit, 1 stop bit and no handshake. Note that UART1 should be connected as well and is the debug UART for Cortex-A7)
- Using your host machine, navigate to the location of your rpmsg_gpio_toggle_freertos_example.bin file and copy it to the Boot partition of your SD card.

Host (Ubuntu)

```
cd <location-of-rpmsg_gpio_toggle_freertos_example.bin>
sudo cp rpmsg_gpio_toggle_freertos_example.bin /media/Boot\ imx7d-/; sync
```

- Boot your i.MX7 from SD card and stop in U-boot. This is done by pressing any key within the first 3 seconds of the bootloader.
- We will first reset U-Boot to the default environment:

Target (U-Boot)

```
=> env default -f -a
```

- Edit the environment to use the dtb designed to run in parallel with FreeRTOS on the M4:

Target (U-Boot)

```
=> editenv fdt_file
```

  - **Once prompted, change "edit: oftree" to "edit: imx7d-phyboard-zeta-004-m4.dtb"**
- Save the environment and reset U-Boot:

Target (U-Boot)

```
=> saveenv
=> reset
```

- Stop in U-boot again by pressing any key within the first 3 seconds of the bootloader.

> If you miss the 3 second window let U-Boot finish loading the kernel, login as root, and enter the following to reboot:
>
> **Target (Linux)**
>
> ```
> reboot
> ```
>
> Make sure you stop in U-Boot this time.

- In U-Boot, type the following to load the application image from the SD card to TCM, flush any cached content, and start the M4 demo:

Target (U-Boot)

```
=> fatload mmc 0:1 0x7f8000 rpmsg_gpio_toggle_freertos_example.bin
=> dcache flush
=> bootaux 0x7f8000
```

- Now Boot into linux:

Target (U-Boot)

```
=> boot
```

- Once logged into Linux as root, load the kernel module to configure RPMSG on the Cortex-A7:

Target (Linux)

```
modprobe imx_rpmsg_tty
```

If you see the following output after loading the module, all is OK!

Expected Output

```
imx_rpmsg_tty virtio0.rpmsg-openamp-demo-channel.-1.0: new channel: 0x400 -> 0x0!
Install rpmsg tty driver!
```

- You can now echo content to /dev/ttyRPMSG0 and it will be received by the M4 and printed on the FreeRTOS serial console. To do this, enter the following:

Target (Linux)

```
echo "led1" > /dev/ttyRPMSG0
```

You will notice that there is a typo in the demo output to the M4 console specifying /dev/ttyRPMSG as the correct device for communicating with the RTOS. This is because there was a change in driver initialization for the imx_rpmsg_tty kernel module for PD19.1.0. This is a known issue and is planned to be resolved in a new demo binary in the future.

- Verify that the LED D1 on the PEB-D-RPI Expansion Board toggled and the following message was outputted to the FreeRTOS console:

Expected Output

```
Got Message From Master Side : "led1"
Toggling GPIO LED1
```