# This document will be Chapter 7 of QuRT RTOS Application System Build Guide (80-VB419-79 Rev. D).

# 7 Multi-process System Build Procedure

## 7.1 Overview

This chapter introduces procedure on building a Multi-Process QuRT (MP QuRT) application system.

MP QuRT application system consists of a guest OS image and optionally application images. Each image is a separate ELF file and needs to be built separately. Application images have different build procedure than guest OS image. The details will be presented in different sections below.

Users needs to select appropriate target to enable multi-process system support. Please contact RTOS team for details.

An example is given at the end to demonstrate the build difference and extra requirements for multi-process application system compared with single process application system.

## Multi-process application build

This section explains how to build

* Guest OS image
* Application images
* Boot image consists of muliple images

Boot image is used by simulator or boot loader to load and run applications on Hexagon processor.

### 7.2.1 Guest OS image build

In MP QuRT, the guest OS is an image that contains kernel, shared device drivers and common system services such as security, timer and etc. The guest OS image runs in a priviledged mode and be protected from user processes.

The guest OS image build follows same procedure as the single-process application system build and configuration introduced in previous chapters.

In summary, it links with

* Kernel library
* Crt0.o and other QuRT libraries
* System configuration object
* Other device driver and system libraries

System configuration is only applicable to guest OS image build. The starting address of guest OS image has to be equal to to greater than 0xf0000000. The example uses 0xfe000000.

### 7.2.2 Application image build

Application image has simplified build procedure. It is built by linking the following files:

* Application object files
* QuRT **crt1.o** and libraries
* C libraries
* Other system libraries except libqurtkernel.a

Application images shall not be overlapped in virtual address space. It will be safe each application takes unique 256MB range. For example, if application A’s starting address is 0x10000000. Application B could start from 0x20000000. Please note above addresses and range are in virtual memory space. The image builder (introduced in next section) will ensure the efficient physical memory allocation for each image.

### 7.2.3 Boot image build

Neither guest OS image nor application image is bootable in multi-process environment. A new image build python tool, “qurt\_image\_build.py”, is created to merge images and generate boot image. The usage of the tool is like:

**Python $(Q6\_RTOS\_ROOT)/scripts/qurt\_image\_build.py os\_image.elf app1.elf app2.elf**

**–o bootimg.pbn –p phys\_addr**

* The –o option specifies the boot image name
* The –p option specifies the physical starting or loading address of the boot image.

The image build tool will automatically update the loading addresses (LMA) of each input image. The LMA of image will be used by debugger to display symbols using virtual address.

### 7.2.4 Multi-process QuRT application system build example

Environment variables used in following example are same as those defined in previous chapters.

Use the following command to link and relocate the guest OS image and libraries:

**qdsp6-ld -m$(Q6VERSION) -nostdlib -section-start \**

**.start=0xfe000000 \**

**$( LIB\_DINKUMWARE\_ROOT)/init.o \**

**$( Q6\_RTOS\_ROOT)/lib/crt0.o \**

**--start-group \**

**$( LIB\_DINKUMWARE\_ROOT)/libc.a \**

**$( LIB\_DINKUMWARE\_ROOT)/libqcc.a \**

**$(Q6\_RTOS\_ROOT)/lib/libqurt.a \**

**$(Q6\_RTOS\_ROOT)/lib/libqurtkernel.a \**

**$(Q6\_RTOS\_ROOT)/lib/libqube\_compat.a \**

**$(Q6\_RTOS\_ROOT)/lib/libtimer.a \**

**$(LIB\_DSP\_ROOT)/libqdsp.a <application object files> --end-group \**

**$( LIB\_DINKUMWARE\_ROOT)/fini.o \**

**-o build/qurt.elf**

Environment variables used in following example are same as those defined in previous chapters.

Use the following command to link application images and libraries:

**qdsp6-ld -m$(Q6VERSION) -nostdlib -section-start \**

**.start=0x1e000000 \**

**$( LIB\_DINKUMWARE\_ROOT)/init.o \**

**$( Q6\_RTOS\_ROOT)/lib/crt1.o \**

**--start-group \**

**$( LIB\_DINKUMWARE\_ROOT)/libc.a \**

**$( LIB\_DINKUMWARE\_ROOT)/libqcc.a \**

**$(Q6\_RTOS\_ROOT)/lib/libqurt.a \**

**$(Q6\_RTOS\_ROOT)/lib/libqube\_compat.a \**

**$(Q6\_RTOS\_ROOT)/lib/libtimer.a \**

**$(LIB\_DSP\_ROOT)/libqdsp.a <application object files> --end-group \**

**$( LIB\_DINKUMWARE\_ROOT)/fini.o \**

**-o build/app1.elf**

**qdsp6-ld -m$(Q6VERSION) -nostdlib -section-start \**

**.start=0x2e000000 \**

**$( LIB\_DINKUMWARE\_ROOT)/init.o \**

**$( Q6\_RTOS\_ROOT)/lib/crt1.o \**

**--start-group \**

**$( LIB\_DINKUMWARE\_ROOT)/libc.a \**

**$( LIB\_DINKUMWARE\_ROOT)/libqcc.a \**

**$(Q6\_RTOS\_ROOT)/lib/libqurt.a \**

**$(Q6\_RTOS\_ROOT)/lib/libqube\_compat.a \**

**$(Q6\_RTOS\_ROOT)/lib/libtimer.a \**

**$(LIB\_DSP\_ROOT)/libqdsp.a <application object files> --end-group \**

**$( LIB\_DINKUMWARE\_ROOT)/fini.o \**

**-o build/app2.elf**

Use the following command to build boot image consists of guest OS image and application images:

**Python $(Q6\_RTOS\_ROOT)/scripts/qurt\_image\_build.py \**

**build/qurt.elf \**

**build/app1.elf \**

**build/app2.elf \**

**-o build/bootimg.pbn \**

**-p 0x10000000**