RD-1AE

This article explains how to upload firmware for the RD-1AE board in Arm Virtual Hardware.

Building

Please refer to our Building Stock RD-1AE Images article.

Assembly

The AVH "coreimg" is a simple ZIP archive that represents the firmware for a VM. The RD-1AE is a complete architecture model that begins execution in the ROM, and as such requires all firmware elements as produced by the Yocto build. The relevant components of the build can be found at:

`build/tmp_$type/deploy/images/fvp-rd-kronos/`

Where `$type` is one of `baremetal`, `virtualization`, or `systemready-glibc` depending on which of the firmware use-cases targets was selected at build time. Within this directory you will find a host of useful build products, but the relevant ones for assembling the firmware are:

```
rse-flash-image.img                                                            |
RSS boot flash
ap-flash-image.img                                                             |
AP Flash
efi-capsule-update-disk-image-fvp-rd-kronos.img                                |
SD card
rse-nvm-image.bin                                                              |
Lifecycle Manager OTP
$type-image-fvp-rd-kronos.wic / arm-systemready-ir-acs-fvp-rd-kronos.wic      |
System storage
encrypted_dm_provisioning_bundle.bin                                           |
DM provisioning firmware
encrypted_cm_provisioning_bundle_0.bin                                         |
CM provisioning firmware
rse-rom-image.img                                                              |
RSS ROM
```

Each of these components plays a critical role in the boot architecture of the RD-1AE. For more information about each element please see the official documentation linked above.

1. Download an official **`coreimg`** firmware container from AVH via the UI. This ZIP archive will be used as a starting point for updating with custom firmware. To do this, select the RD-1AE device in the AVH web interface, select the firmware base you would like to use, and then select **`Source Image`** to download the archive. For the purposes of this document the downloaded file will be referred to as **`baremetal-critical-application-monitoring-1.1.1.coreimg`**.

2. Collect and rename the Yocto build products according to the following mapping:

Original Name	**Updated Name**
rse-flash-image.img	boot_flash
ap-flash-image.img	ap_flash
efi-capsule-update-disk-image-fvp-rd-kronos.img	sdcard
rse-nvm-image.img	lcm_otp

Original Name	Updated Name
$type-image-fvp-rd-kronos.wic / arm-systemready-ir-acs-fvp-rd-kronos.wic	virtio_0
encrypted_dm_provisioning_bundle.bin	unchanged
encrypted_cm_provisioning_bundle_0.bin	unchanged
rss-rom-image.img	unchanged

3. First, we need to pack the ROM and the CM/DM provisioning firmware bundles into a nested archive. These firmware components of the RD-1AE are expected to be loaded into memory at specific addresses by a "debugger", or by the platform model.

The nested ZIP archive, named **`firmware`**, contains the multiple binary firmware components. When multiple binary files are used, AVH also requires a **`load.txt`** file, which serves as a scatter-gather instruction file, specifying where each component should be loaded in device memory.

Example **`load.txt`** from the RD-1AE Safety Island PSA Secure Storage APIs Architecture Test Suite (1.1.1) stock firmware:

```
$ cat load.txt
name:rse-rom-image.img                                  load:0x11000000
name:encrypted_cm_provisioning_bundle_0.bin             load:0x31000000
name:encrypted_dm_provisioning_bundle.bin               load:0x31080000
```

For more details about the ZIP archive and firmware file formats for microcontroller-based devices, refer to the firmware file formats documentation.

The resulting ZIP archive is named **`firmware`** and contains the **`load.txt`** file and the multiple binaries:

```
$ zip firmware encrypted_cm_provisioning_bundle_0.bin
encrypted_dm_provisioning_bundle.bin rse-rom-image.img
adding: load.txt (deflated 44%)
adding: rse-rom-image.img (deflated 67%)
adding: encrypted_dm_provisioning_bundle.bin (deflated 95%)
adding: encrypted_cm_provisioning_bundle_0.bin (deflated 79%)
```

Without including a **`load.txt`** file in the ZIP archive when uploading the custom firmware, the device may boot into an error state with the message:

```
There was a problem starting this device: Missing LOAD.TXT.
```

1. Finally, use **`zip`** to update the contents of the downloaded **`.coreimg`** ZIP archive by adding the **`firmware`** ZIP archive and the renamed build products. This will replace the existing files with the updated versions.

```
$ zip baremetal-critical-application-monitoring-1.1.1.coreimg firmware
ap_flash boot_flash virtio_0 sdcard lcm_otp
updating: firmware (deflated 2%)
updating: sdcard (deflated 98%)
updating: virtio_0 (deflated 85%)
updating: boot_flash (deflated 98%)
updating: ap_flash (deflated 98%)
updating: lcm_otp (deflated 100%)
```

By updating the contents of the originally downloaded coreimg, an **`Info.json`** file will be present within the **`coreimg`**, containing device metadata. If needed, you can modify the metadata in **`Info.json`** (list the files in the ZIP archive with **`unzip -l <*.coreimg>`**).

Most elements in this file should remain unchanged, but you may update the **`Build`** and/or **`Version`** fields as needed. These changes will be reflected in the UI dashboard of your VM instance.

```
diff --git a/yocto/meta-arm-bsp-extras/recipes-bsp/trusted-firmware-
a/files/fvp-rd-kronos/rdkronos.dts b/yocto/meta-arm-bsp-extras/recipes-
bsp/trusted-firmware-a/files/fvp-rd-kronos/rdkronos.dts
index 65ff67b..67dd3b2 100644
--- a/yocto/meta-arm-bsp-extras/recipes-bsp/trusted-firmware-a/files/fvp-rd-
kronos/rdkronos.dts
+++ b/yocto/meta-arm-bsp-extras/recipes-bsp/trusted-firmware-a/files/fvp-rd-
kronos/rdkronos.dts
@@ -604,7 +604,7 @@
                 * 0x7fbf 0000 - 0x7fbf ffff : FFA_SHARED_MM_BUF
                 */
                reg = <0x00000000 0x80000000 0 0x7fbf0000>,
-                         <0x00000080 0x80000000 0 0x80000000>;
+              <0x00000080 0x80000000 0x00000003 0x80000000>;
        };

        reserved-memory {
@@ -721,6 +721,21 @@
                #size-cells = <2>;
                ranges;

+        xhci@10000000 {
+            compatible = "generic-xhci";
+            reg = <0 0x10000000 0 0x10000>;
+            interrupts = <0x00 0x33 0x04>;
+        };
+
+        framebuffer@b000000000 {
+            compatible = "simple-framebuffer";
+            reg = <0xb0 0x0 0x0 (1920*1080*4)>;
+            width = <1920>;
+            height = <1080>;
+            stride = <(1920*4)>;
+            format = "a8b8g8r8";
+        };
+
                timer@2a810000 {
                        compatible = "arm,armv7-timer-mem";
                        reg = <0x0 0x2a810000 0 0x10000>;
```

In addition, apply the following patch to disable PCI passthrough of an unmodeled AHCI PCI device. This is relevant only for the *virtualization* firmware targets.

```
diff --git a/yocto/meta-arm-auto-solutions/recipes-core/domu-package/domu-
envs.inc b/yocto/meta-arm-auto-solutions/recipes-core/domu-package/domu-
envs.inc
index 56ec5a5..b4ebbf9 100644
--- a/yocto/meta-arm-auto-solutions/recipes-core/domu-package/domu-envs.inc
+++ b/yocto/meta-arm-auto-solutions/recipes-core/domu-package/domu-envs.inc
@@ -35,7 +35,7 @@ DOMU1_NUMBER_OF_CPUS ?= "2"
 DOMU1_VCPU_PIN ?= "cpus = [\"1\", \"2\"]"
 DOMU1_MPAM ?= "mpam = [\"slc=0xc0\"]"
 DOMU1_SVE ?= "sve = \"${DOMU_SVE_SETTING}\""
-DOMU1_PCI_PASSTHROUGH ?= "pci = [\"${DOMU1_PCI_ID}\"]"
+DOMU1_PCI_PASSTHROUGH ?= ""
 DOMU1_EXTRA ?=
"${DOMU1_BRIDGES}\\n${DOMU1_VCPU_PIN}\\n${DOMU1_MPAM}\\n${DOMU1_SVE}\\n${DOMU
1_PCI_PASSTHROUGH}\\n"

 DOMU2_BRIDGES ?= "vif = ['script=vif-openvswitch,bridge=ovsbr0,vifname
```