Virtual Memory to Memory Scaler Hardware Device Using QEMU

Karthik Poduval

Abstract: ***This paper describes the methods to build a virtual memory to memory device attached to QEMU's ARM virt generic hardware device. This virtual hardware device could be used as a development platform to demonstrate Linux device driver development for Video4Linux2 M2M (Memory to Mem- ory) drivers.***

Keywords: QEMU, Scaler, Virtual, Hardware Emulation, M2M, V4L2 M2M

1. Introduction

Virtual hardware (that interface function like the real one) are extremely helpful as an educational tool while trying to learn how to develop device driver without having the need to use real hardware. It is also helpful tool for architecture exploration or exploring hardware software co-design without having to build the hardware. QEMU [1] is an open source machine emulator. QEMU is used widely as machine emulator and is the default emulator for Android build system and Yocto. This allows to run code for any machine/SoC archi- tecture on a given host machine that maybe of a different architecture. For example: qemuarm64 machine on yocto is a virtual QEMU ARM64 bit machine. Once can build image for this machine using Yocto and even boot up this machine on a X86 64 host computer using QEMU. The qemuarm64 machine of Yocto maps to QEMU's ARM System Emulator [2].

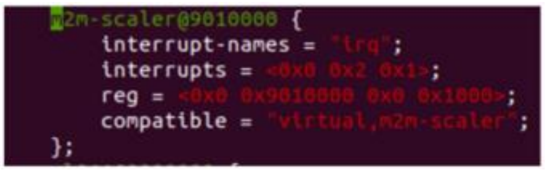
QEMU Device Model

QEMU Device Model is a method to add a device emulation to QEMU. An example of adding a QEMU Device Model can be seen in [3] and [4] example from Xilinx. Just like how a hardware device has an internal state, the QEMU Device Model also has a state object. The object may also include other resources like memory region (for MMIO access), in- terrupt lines and DMA capabilities. QEMU provides APIs to connect Device Model resources to the QEMU machine.

M2M Virtual Scaler

The M2M scaler virtual device has been added to QEMU virt ARM64 generic machine. The memory map of the virt device was modified to create a `0x1000` sized entry for the virtual M2M Scaler. A device tree entry was also created programaticly for the Virtual M2M Scaler for the device as shwon in Figure 1 The virt machine was also modified to add an IRQ line for the M2M Scaler Device. Next a device model was created for the M2M Scaler Device with following resources.

- MMIO Region



```
m2m-scaler@9010000 {
    interrupt-names = "irq";
    interrupts = <0x0 0x2 0x1>;
    reg = <0x0 0x9010000 0x0 0x1000>;
    compatible = "virtual,m2m-scaler";
};
```

Figure 1: Device Tree Entry

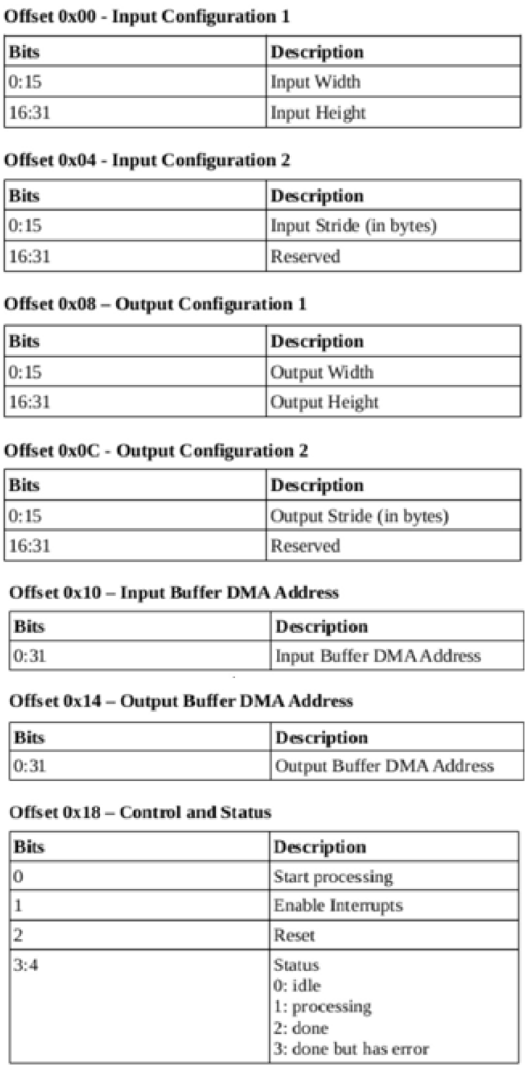
- IRQ Line
- DMA Read and Write Capability

The M2M Scaler device would be registered through the virt device using the allocated MMIO space and IRQ resource. Once registered the M2M Scaler device would then use the MMIO space tp install register read and write callbacks. These callbacks will be invoked in the M2M Scaler Device Model once a guest OS software starts communicating with the MMIO space. The read and write calls will cause the state of the device to be modified. For the actual scaler logic and open source project called STB [5] was used which is a header only implementation so its easy to incorporate into the M2M Scaler Device QEMU implementation. The register map of the device is shown in Figure 2.

The programming model is shown below:

- Program input width and height register 0x0
- Program input stride register 0x4
- Program output width and height register 0x8
- Program output stride register 0xC
- Program input DMA Address register 0x10
- Program output DMA Address register 0x14
- Write the start bit (bit 0) register 0x18
- Either poll or wait for interrupt (if interrupt enabled register 0x14 bit 1)

After all parameters for scaler are programmed and guest OS driver intends to begin the scaling operation using the start bit, the M2M Scaler device code in QEMU will signal an internal thread using a condition variable. The thread is used to emulate the asynchronous behavior of the virtual hardware. The thread will consume the programmed parameters through the device state object and start doing the scaling operation using the STB scaling function. For simplicity the M2M Scaler device only supports a single RGB888 color format. The DMA read is used to transfer the contents from Guest OS DDR address for input scaling image into an internal device state memory. The STB scaling function is invoked with input and output memories from device state object. If scaling was successful, the contents of scaling output are transferred to



Offset 0x00 - Input Configuration 1

Bits	Description
0:15	Input Width
16:31	Input Height

Offset 0x04 - Input Configuration 2

Bits	Description
0:15	Input Stride (in bytes)
16:31	Reserved

Offset 0x08 – Output Configuration 1

Bits	Description
0:15	Output Width
16:31	Output Height

Offset 0x0C - Output Configuration 2

Bits	Description
0:15	Output Stride (in bytes)
16:31	Reserved

Offset 0x10 – Input Buffer DMA Address

Bits	Description
0:31	Input Buffer DMA Address

Offset 0x14 – Output Buffer DMA Address

Bits	Description
0:31	Output Buffer DMA Address

Offset 0x18 – Control and Status

Bits	Description
0	Start processing
1	Enable Interrupts
2	Reset
3:4	Status 0: idle 1: processing 2: done 3: done but has error

Figure 2: Virtual M2M Scaler Register Map

the Guest OS DDR memory using a DMA operation API.

After this the interrupt is raised using QEMU's API which will cause the Guest OS's interrupt service routing to get invoked. If scaling wasn't sucessful, then approprate error status bits and updated in the device state before invoking an IRQ. The overall system design can be seen in Figure 3. This work was inspired by [3] PCI device but implemented it as an MMIO device. The full source code of the implementation can be found in [6] and [7].

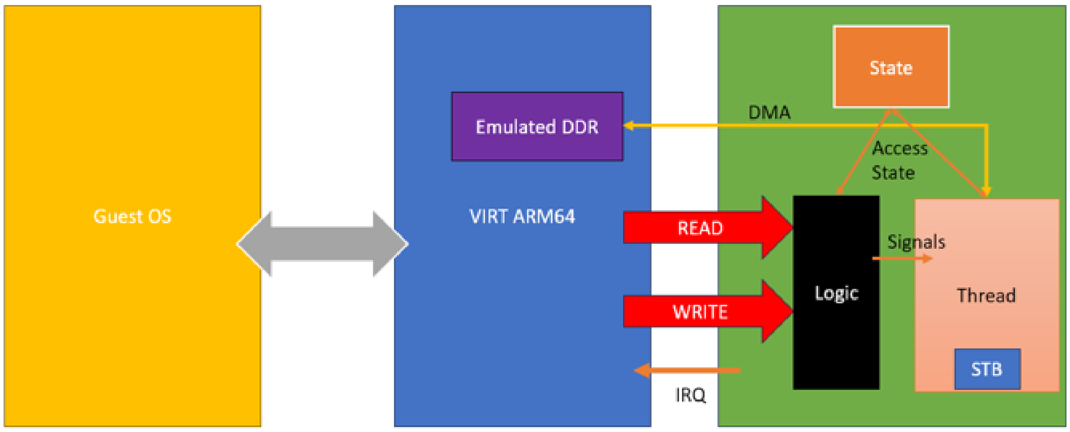


Figure 3: M2M Scaler Design

2.

3. Conclusions

This work done as a part of this paper was used to develop a virtual M2M scaler device on QEMU that was used to implement a V4L2 M2M Guest OS device driver. Such techniques to build virtual MMIO devices could be used to develop virtual hardware devices for use cases such as.

- Easy to understand devices for device driver development as an educational tool
- A platform to prototype new hardware ideas for architec- tural exploration

V4L2 M2M [8] is a very good Linux kernel framework that can be used to develop device drivers for M2M devices and such a virtual hardware forms a great example of usable virtual hardware for software design.

References

[1] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41. Califor-nia, USA, 2005, p. 46.
[2] "Qemu system arm." [Online]. Available: https://www.qemu.org/docs/ master/system/target-arm.html
[3] "Educational pci device." [Online]. Available: https://github.com/ karthikpoduval/qemu/blob/elc-2022/hw/misc/edu.c
[4] "Qemu device model." [Online]. Available: https: //xilinx-wiki.atlassian.net/wiki/spaces/A/pages/861569267/QEMU+ Device+Model+Development
[5] "Stb." [Online]. Available: https://github.com/nothings/stb
[6] "virt." [Online]. Available: https://github.com/karthikpoduval/qemu/blob/ elc-2022/hw/arm/virt.c
[7] "M2m scaler device." [Online]. Available: https://github.com/ karthikpoduval/qemu/blob/elc-2022/hw/misc/m2m scaler.c
[8] "Linux kernel v4l2 documentaion." [Online]. Available: https://www. kernel.org/doc/html/v4.19/media/uapi/v4l/v4l2.html