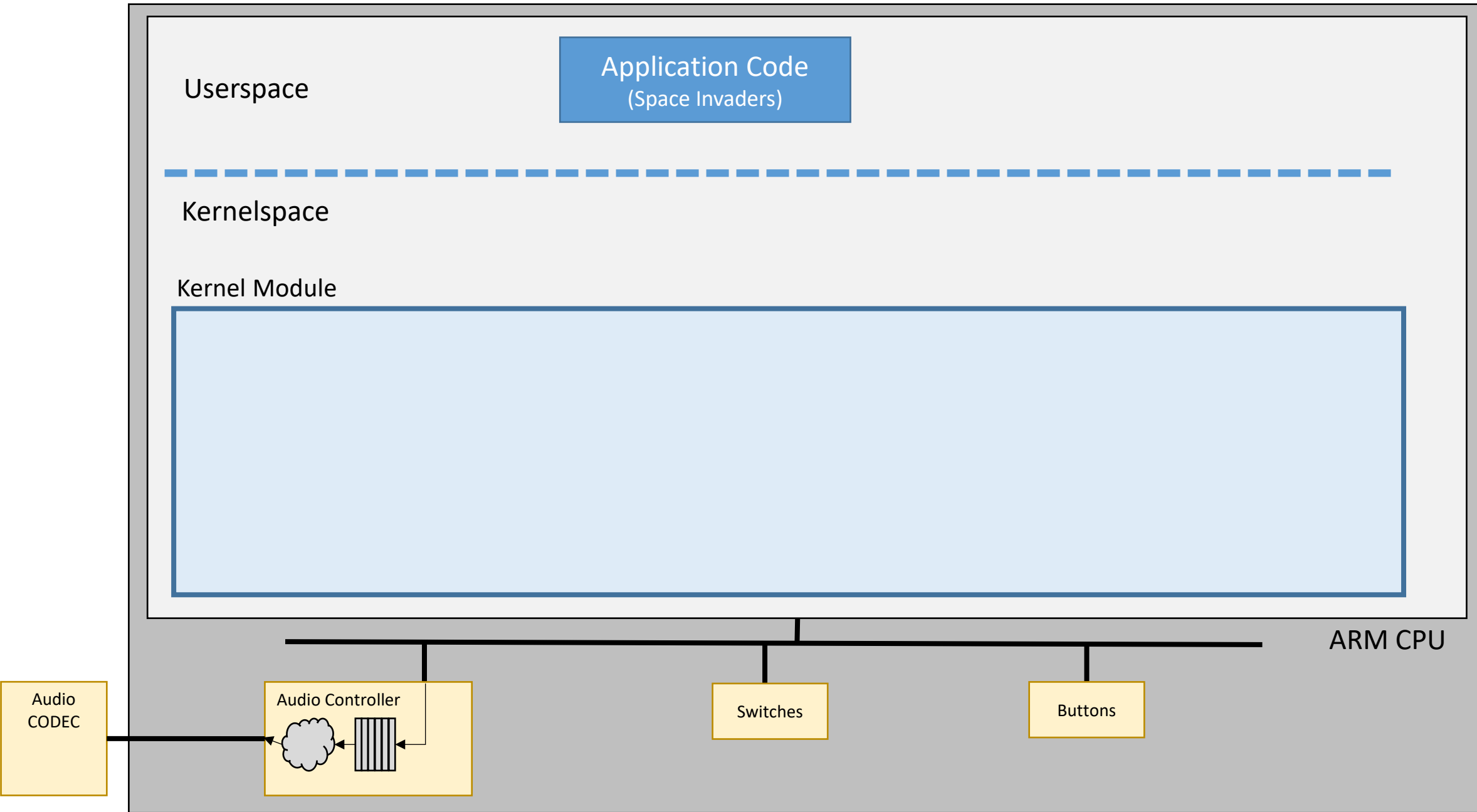


Linux Driver for Audio

Lab 5 Milestone 1 & 2

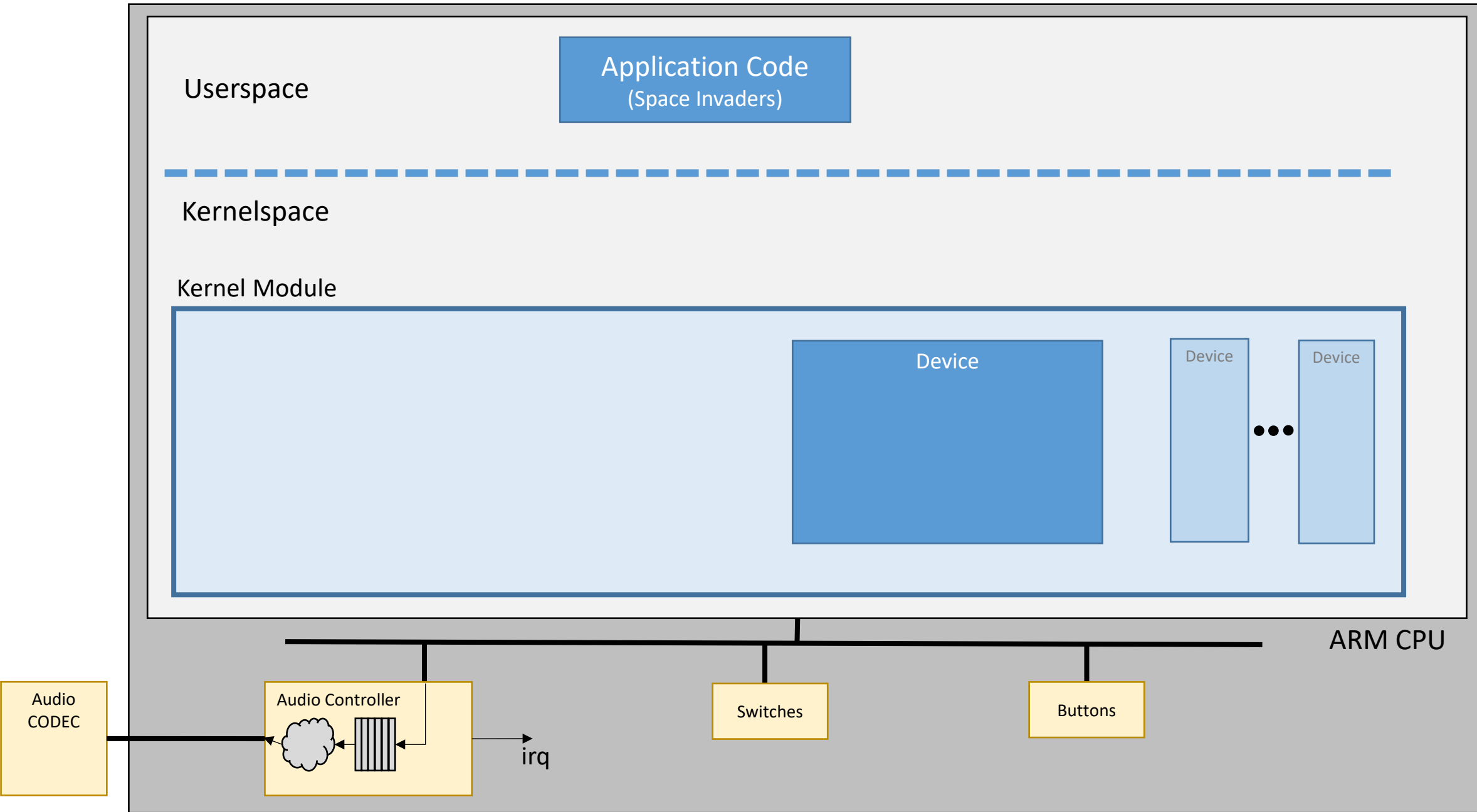
What our driver needs to do:

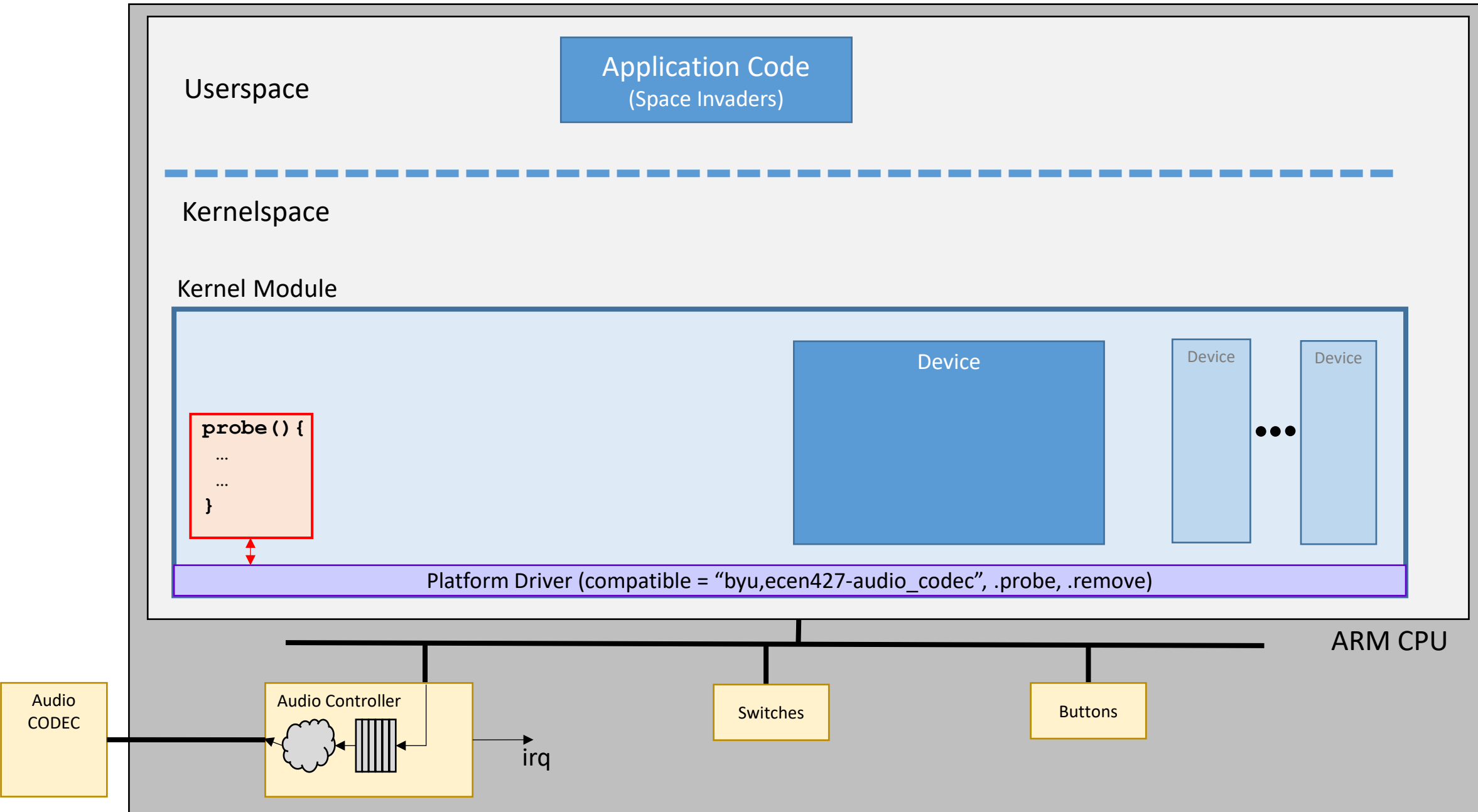
- Be notified of hardware in the system (Milestone 1)
- Allow user code to talk to the driver (Milestone 1)
- Talk to the hardware (Milestone 2)
- Handle interrupts from the hardware (Milestone 2)



What our driver needs to do:

- Be notified of hardware in the system (Milestone 1)
- Allow user code to talk to it (Milestone 1)
- Talk to the hardware (Milestone 2)
- Handle interrupts from the hardware (Milestone 2)





```
platform_driver_register(struct platform_driver *)
```

What our driver needs to do:

- Be notified of hardware in the system (Milestone 1)
- Allow user code to talk to the driver (Milestone 1)
- Talk to the hardware (Milestone 2)
- Handle interrupts from the hardware (Milestone 2)

User Code Needs to Talk to Driver

End Goal: Create a device file (/dev/xxx) that we can read() and write() to. *(Recall how you used /dev/uio)*

The device file (/dev/xxx) is an interface to a **character device**.

Steps:

1. Create a character device
2. Create a device file

Userspace

Application Code
(Space Invaders)

Kernelspace

Kernel Module

`int major_num;`

```
probe () {  
  ...  
  ...  
}
```

Device

`int minor_num;`

Device

Device

Platform Driver (compatible = "byu,ecen427-audio_codec", .probe, .remove)

ARM CPU

Audio
CODEC

Audio Controller

Switches

Buttons

irq

```
alloc_chrdev_region(dev_t * output, minor_start, count, MODULE_NAME)
```

Userspace

Application Code
(Space Invaders)

Kernelspace

Kernel Module

`int major_num;``struct file_operations fops;``probe() {
 ...
 ...
}``read() {
 ...
 ...
}``write() {
 ...
 ...
}`

Device

`int minor_num;``struct cdev cdev;`

Device

Device

Platform Driver (compatible = "byu,ecen427-audio_codec", .probe, .remove)

ARM CPU

Audio
CODEC

Audio Controller

Switches

Buttons

irq

`cdev_init (struct cdev*, fops* {.read, .write})`

Userspace

Application Code
(Space Invaders)

Kernelspace

Kernel Module

Character Device
(major, minor)Char.
DeviceChar.
Device

```
int major_num;  
struct file_operations fops;
```

```
probe() {  
  ...  
  ...  
}
```

```
read() {  
  ...  
  ...  
}
```

```
write() {  
  ...  
  ...  
}
```

Device

```
int minor_num;  
struct cdev cdev;
```

Device

Device

Platform Driver (compatible = "byu,ecen427-audio_codec", .probe, .remove)

ARM CPU

Audio
CODEC

Audio Controller

Switches

Buttons

irq

```
cdev_add (struct cdev*, dev_t, count)
```

User Code Needs to Talk to Driver

End Goal: Create a device file (/dev/xxx) that we can read() and write() to. *(Recall how you used /dev/uio)*

The device file (/dev/xxx) is an interface to a **character device**.

Steps:

1. Create a character device
2. Create a device file

Userspace

Application Code
(Space Invaders)

Kernelspace

Kernel Module

Character Device
(major, minor)Char.
DeviceChar.
Device

```
int major_num;  
struct file_operations fops;  
struct class * class
```

```
probe() {  
    ...  
    ...  
}  
read() {  
    ...  
    ...  
}  
write() {  
    ...  
    ...  
}
```

Device

```
int minor_num;  
struct cdev cdev;
```

Device

Device

Platform Driver (compatible = "byu,ecen427-audio_codec", .probe, .remove)

ARM CPU

Audio
CODEC

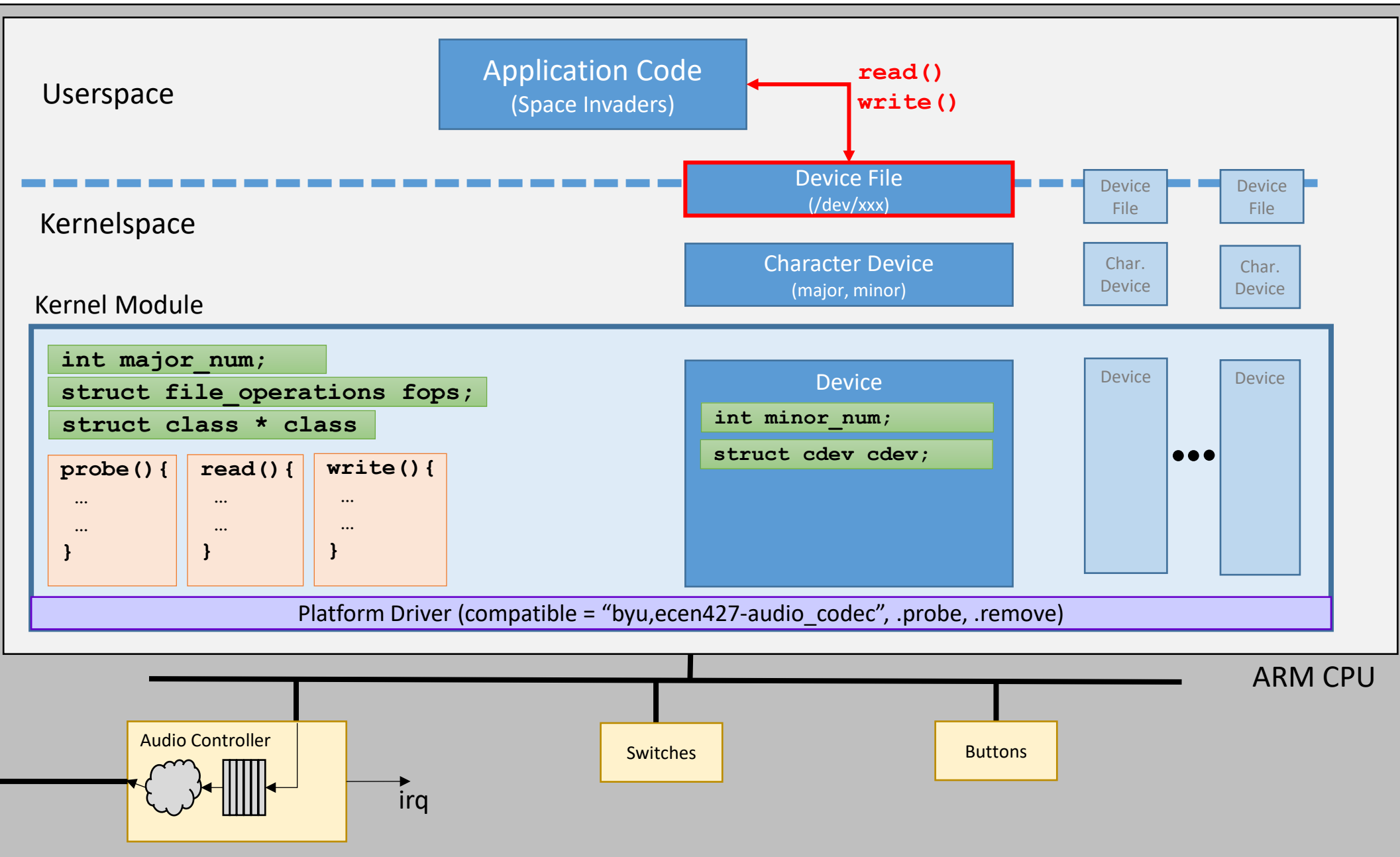
Audio Controller

Switches

Buttons

irq

```
class_create(owner = THIS_MODULE, "my class name")
```



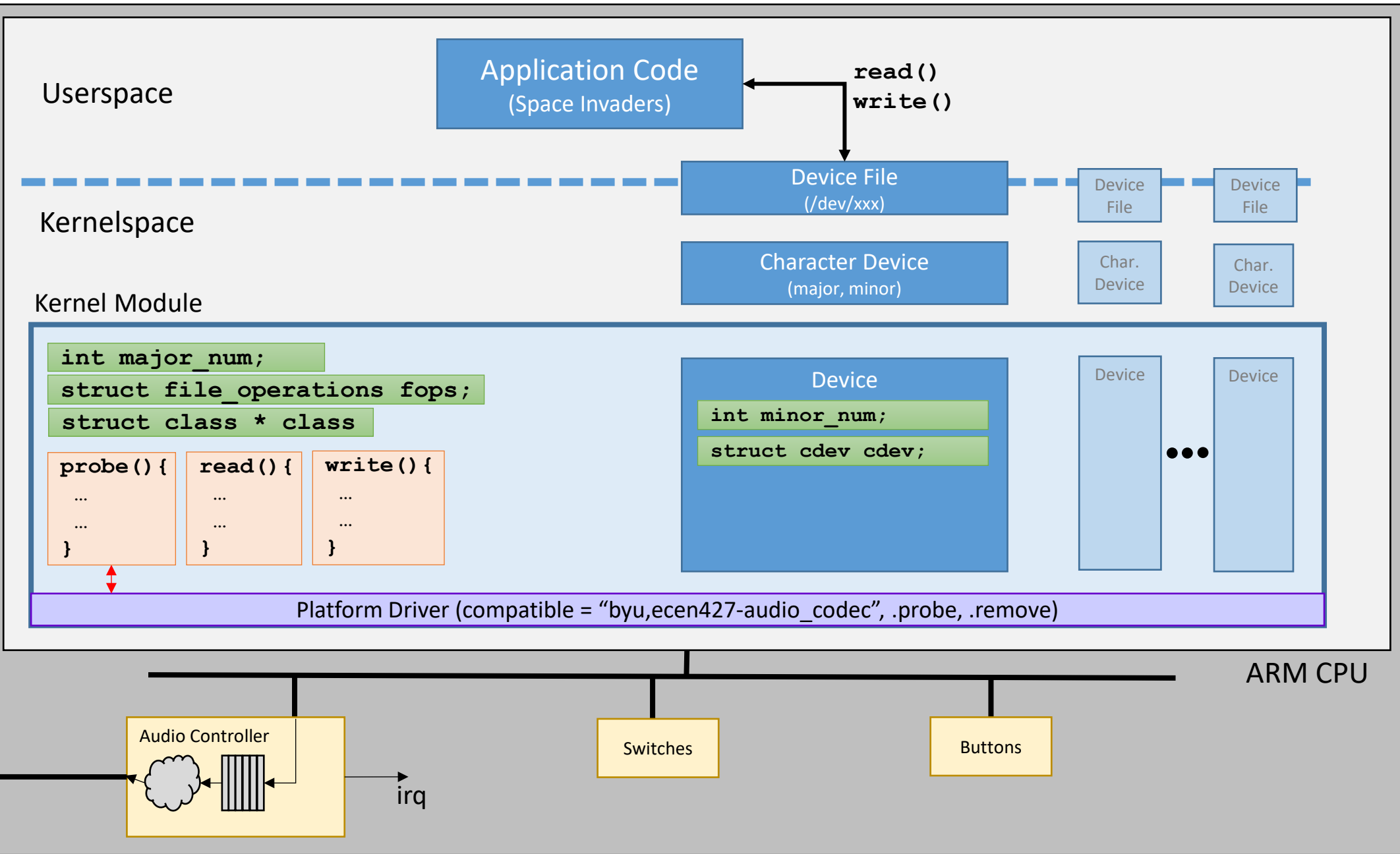
```
device_create (struct class*, parent = NULL, dev_t, "your device name")
```

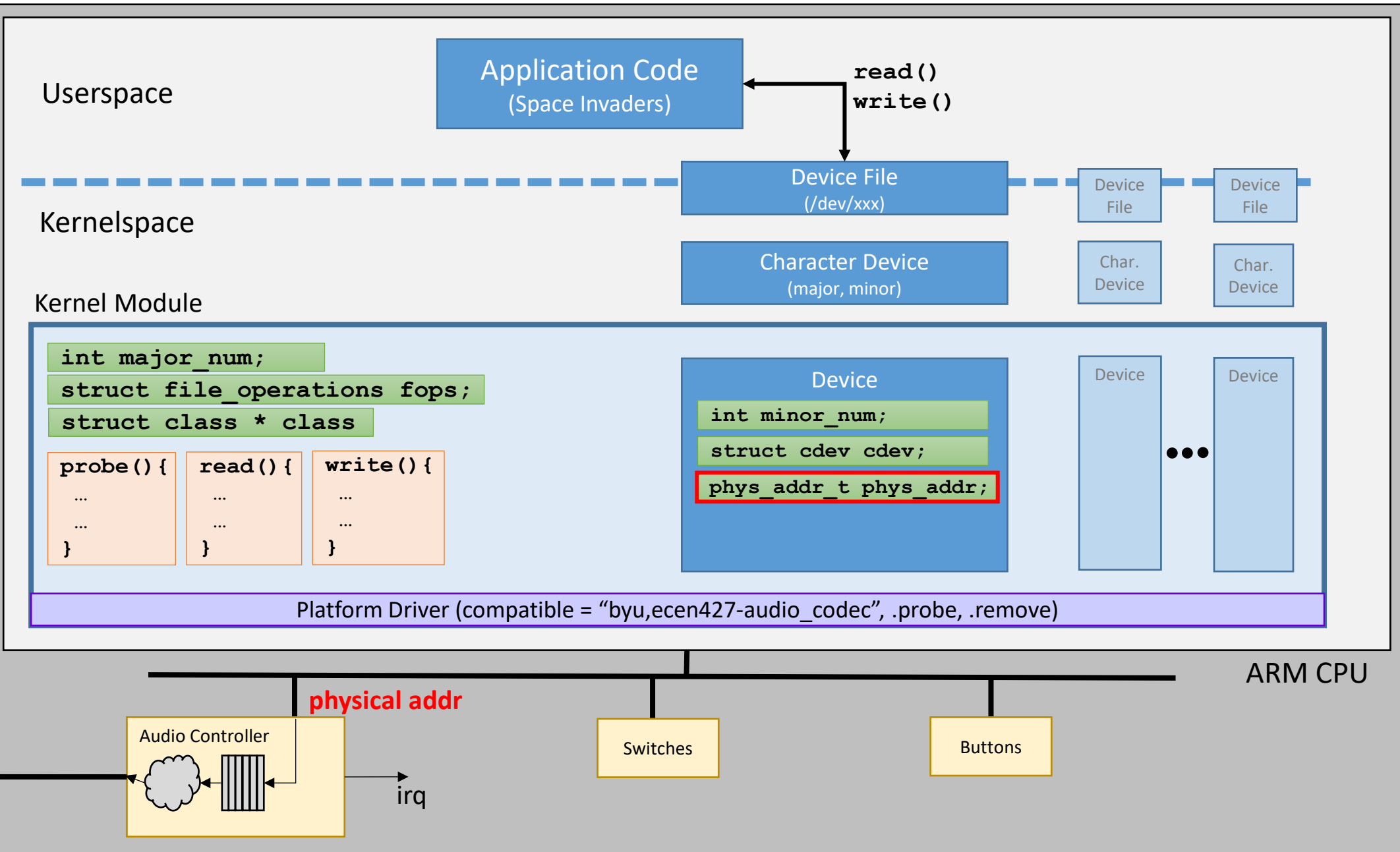
What our driver needs to do:

- Be notified of hardware in the system (Milestone 1)
- Allow user code to talk to it (Milestone 1)
- Talk to the hardware (Milestone 2)
- Handle interrupts from the hardware (Milestone 2)

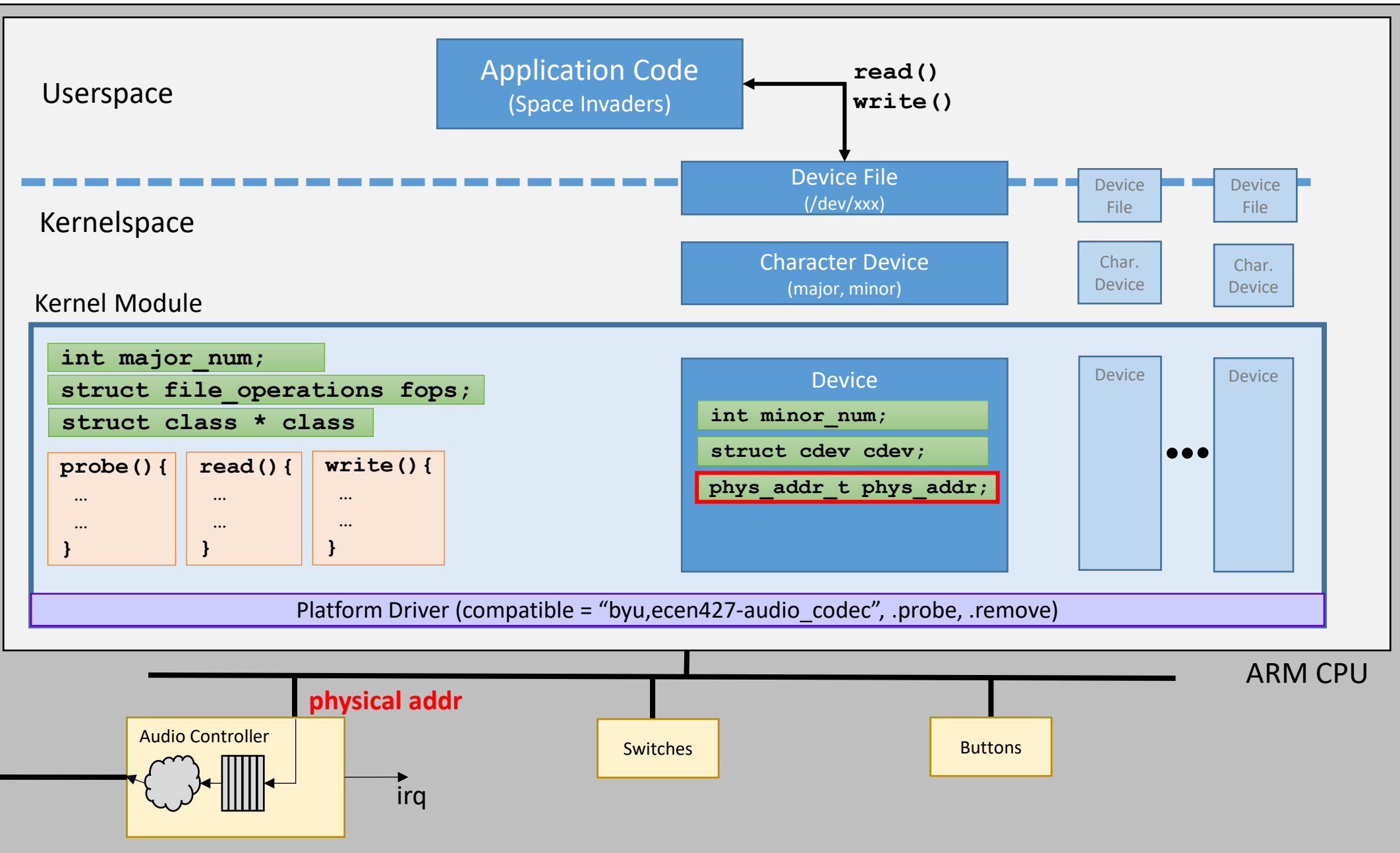
Driver needs to talk to the hardware

1. Need to figure out physical address
2. Need to reserve the physical address
3. Need to get a pointer (virtual address) to the physical address

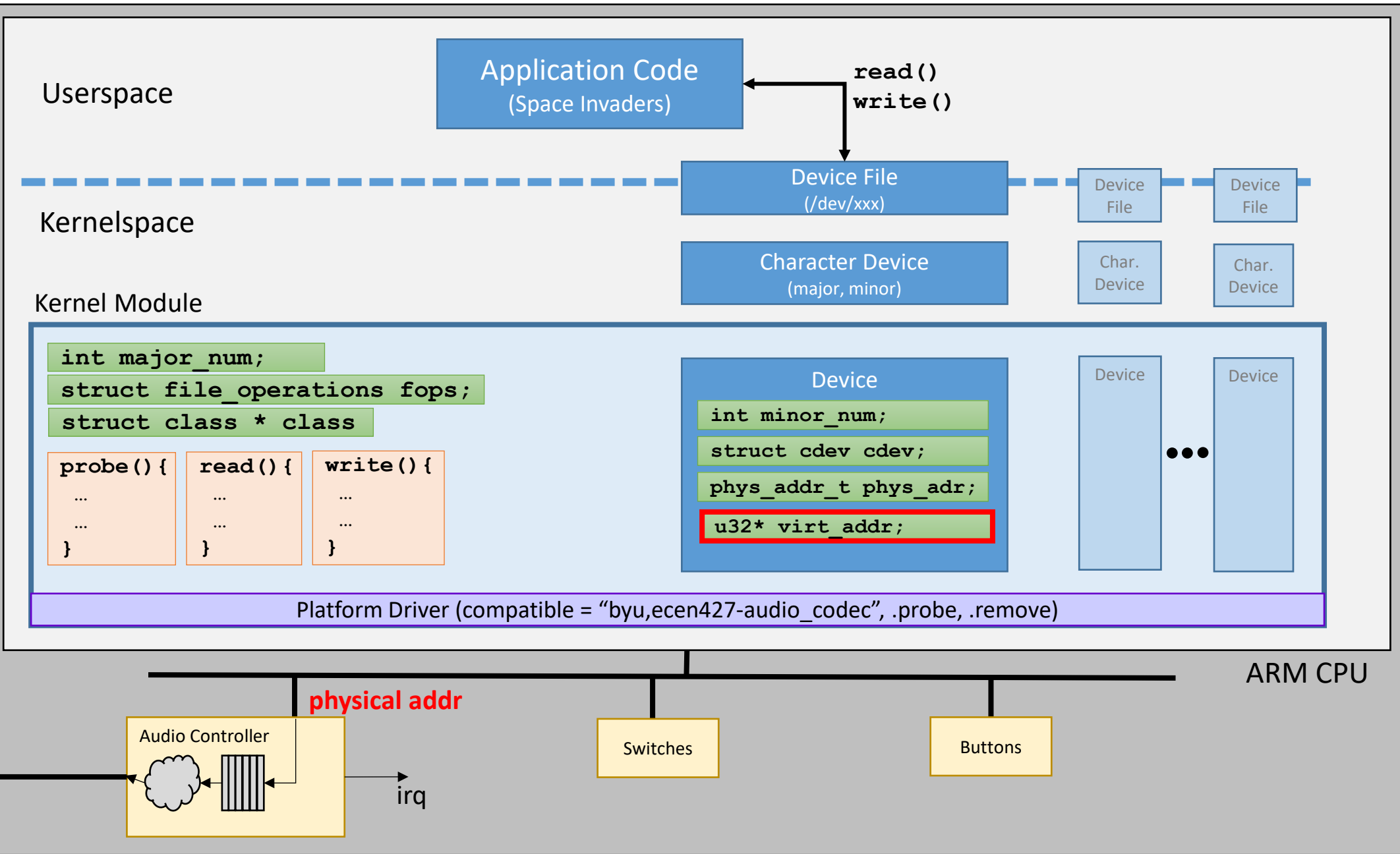




```
platform_get_resource(struct plaform_device * dev, IORESOURCE_MEM, 0);
```



```
request_mem_region(phys_addr, size, MODULE_NAME);
```

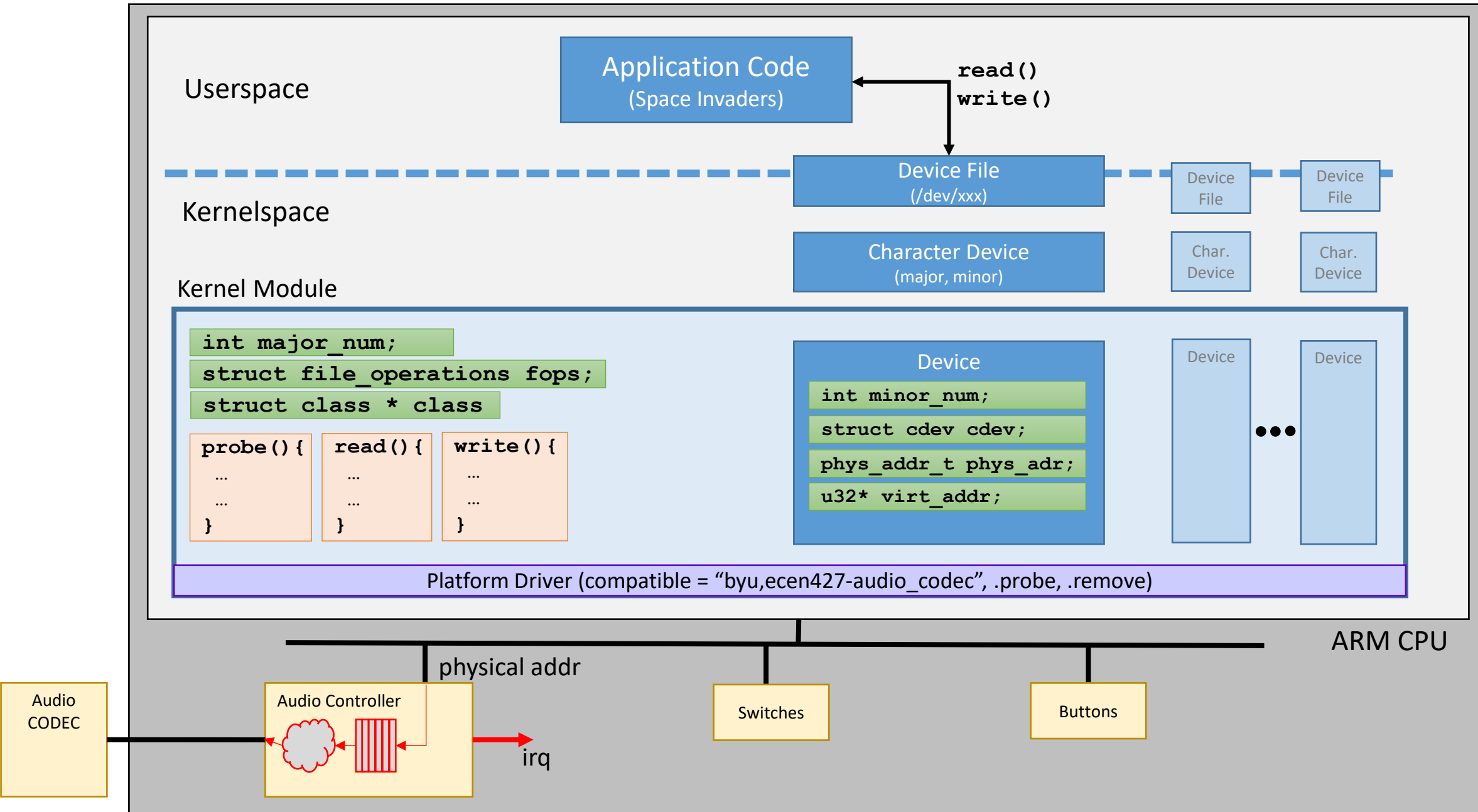


```
virt_addr = ioremap(phys_addr, size);
```

Driver needs to talk to the hardware

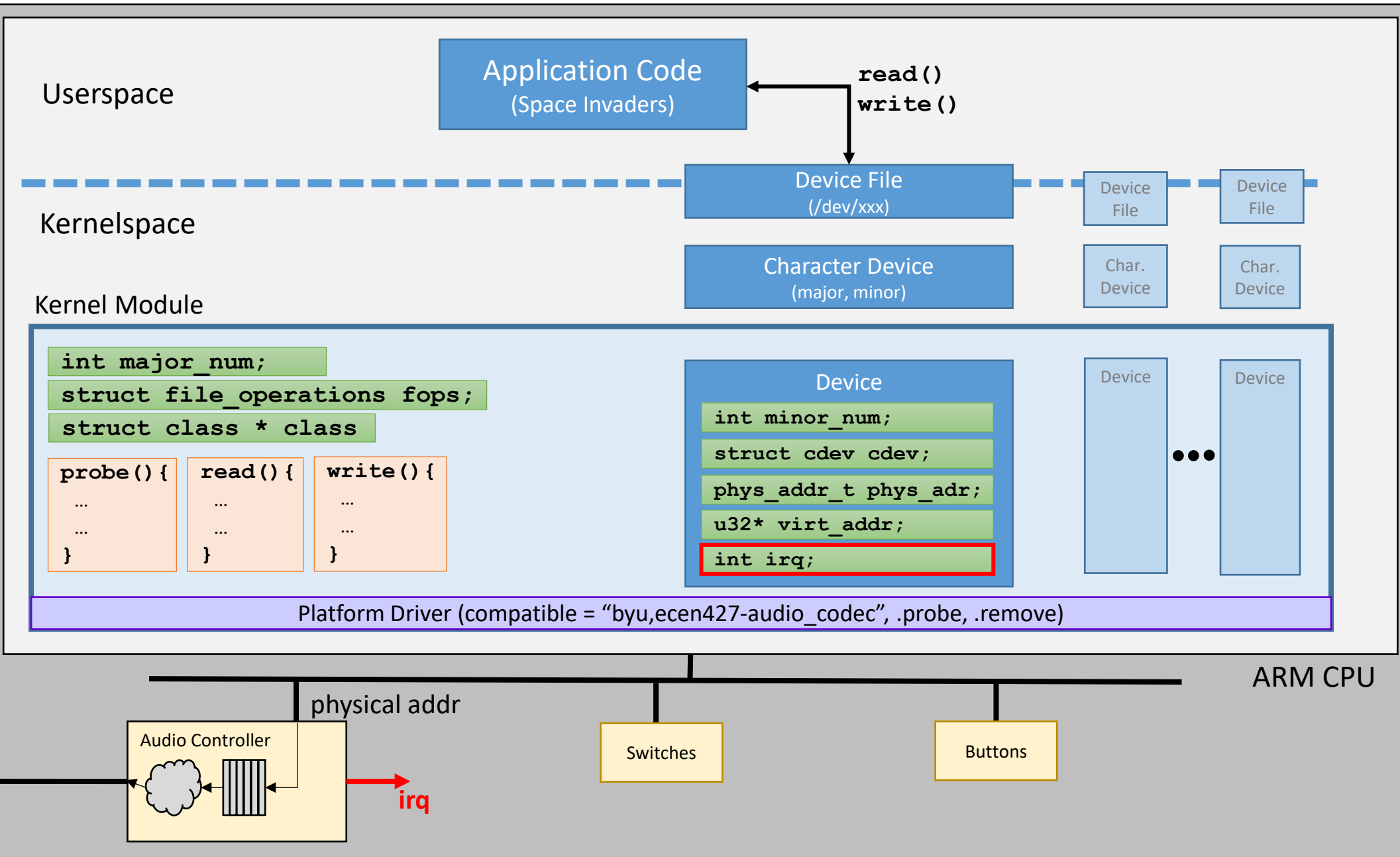
1. Need to figure out physical address
2. Need to reserve the physical address
3. Need to get a pointer (virtual address) to the physical address
4. Talk to the hardware with:
 - `iowrite32 (value, virt_addr + offset)`
 - `ioread32(virt_addr + offset)`

Driver Needs to Handle Interrupts

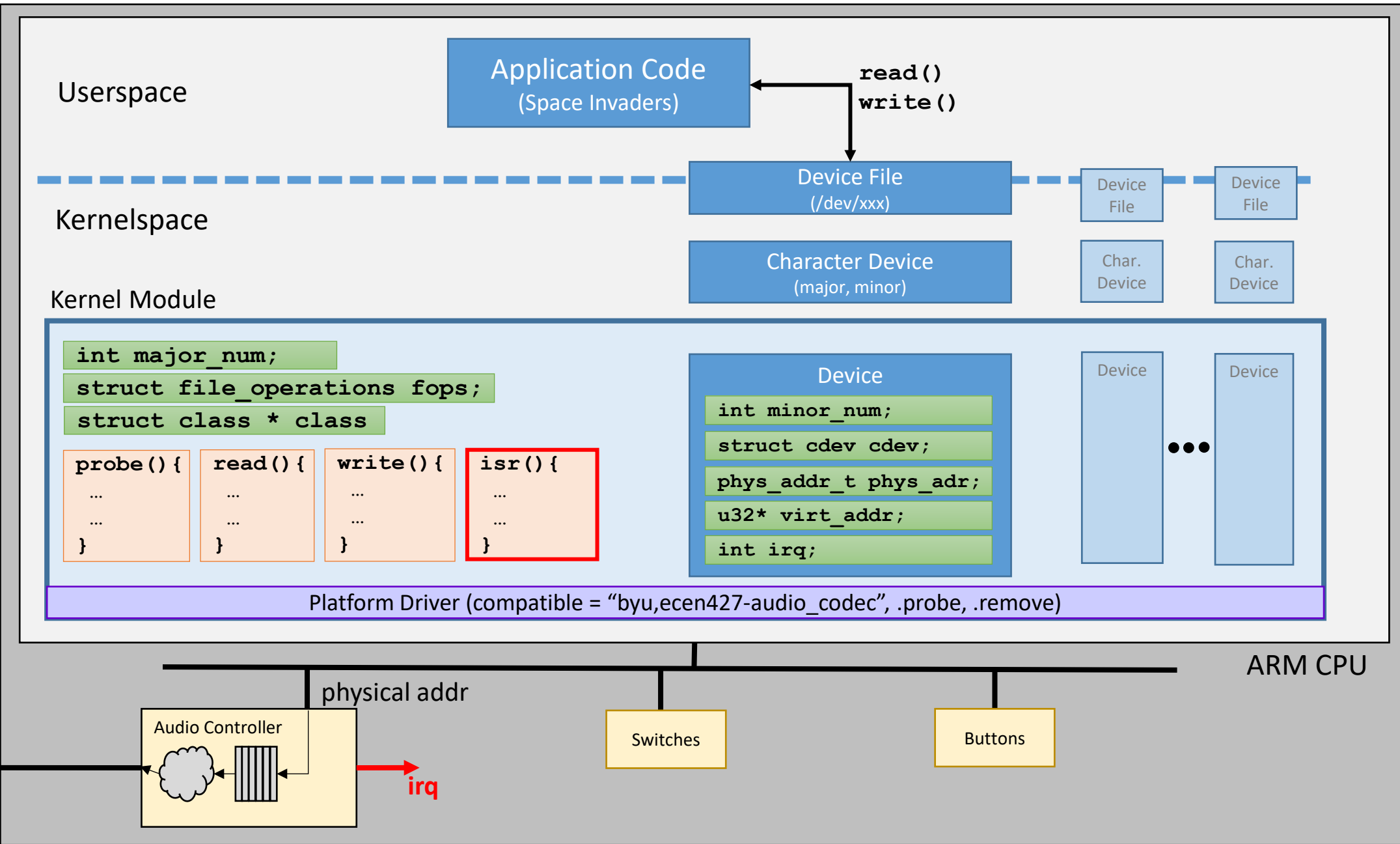


Driver Needs to Handle Interrupts

1. Get IRQ Number
2. Register Interrupt Handler with Linux



```
irq = platform_get_resource(struct platform_device * dev, IORESOURCE_IRQ, 0);
```



The diagram illustrates the Linux kernel architecture for a character device driver, showing the interaction between Userspace, Kernel Filesystem, and Kernel Modules.

Userspace: Application Code (Space Invaders) interacts with the Kernel Filesystem via `read()` and `write()` system calls.

Kernel Filesystem: Contains the `Device File (/dev/xxx)`, which is linked to the `Character Device (major, minor)`.

Kernel Module: Contains the driver code, including the `Platform Driver` and the `Device` structure.

Platform Driver: Contains the following code snippets:

```
int major_num;
struct file_operations fops;
struct class * class
```

Function definitions (highlighted in red in the original image):

```
probe() { ... }
read() { ... }
write() { ... }
isr() { ... }
```

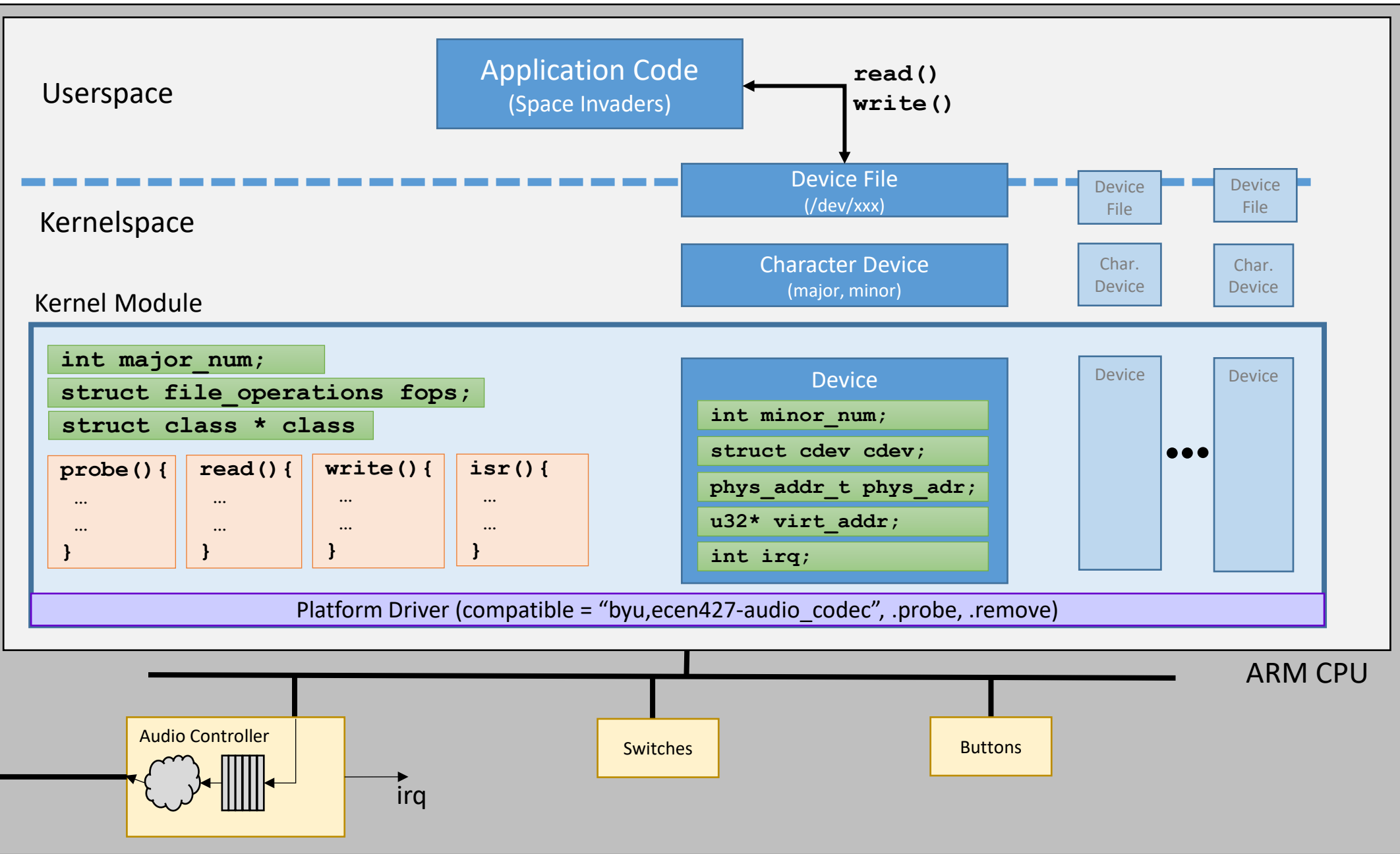
Device Structure: Contains the following fields:

```
int minor_num;
struct cdev cdev;
phys_addr_t phys_addr;
u32* virt_addr;
int irq;
```

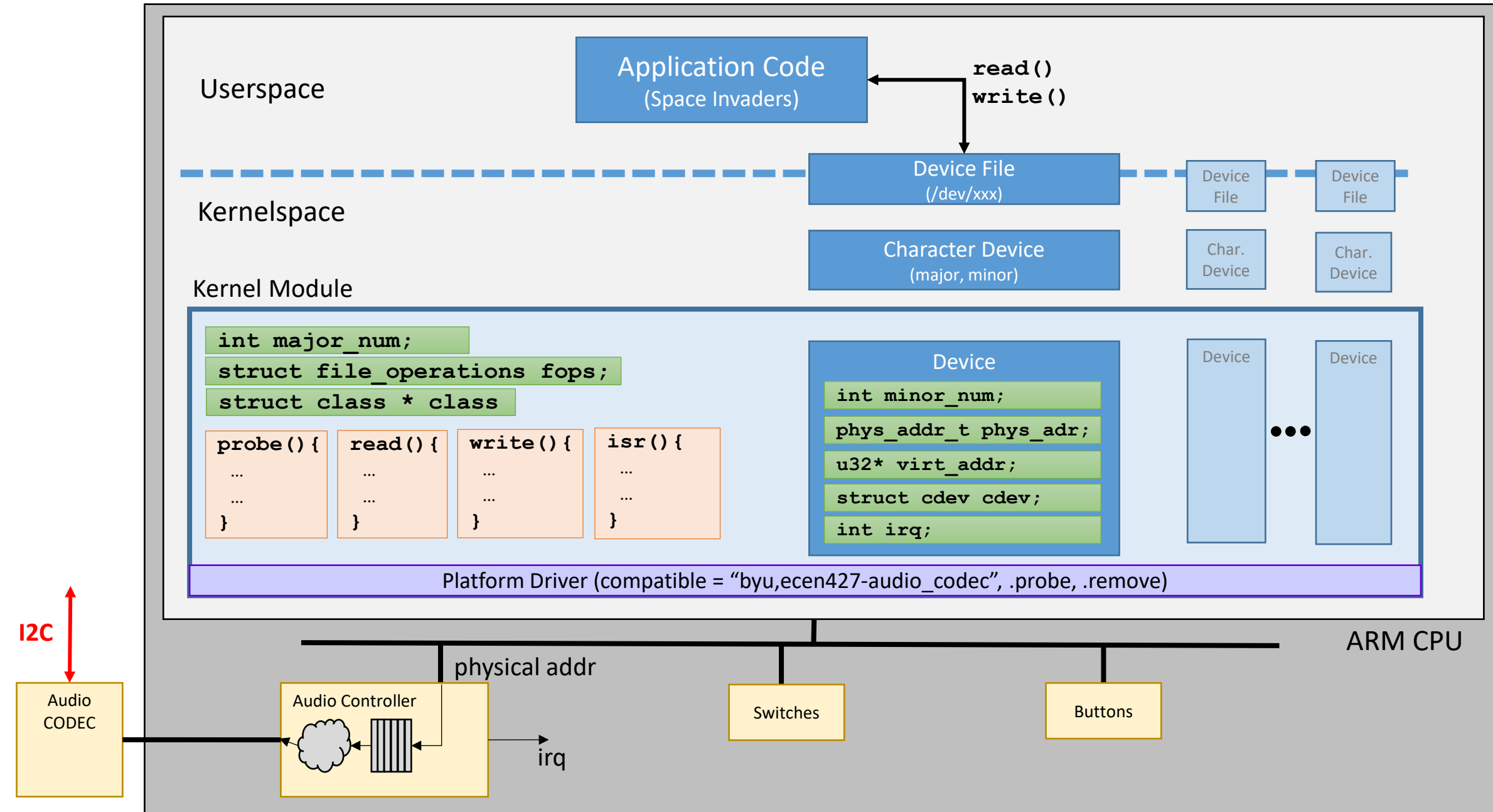
Hardware Components: The `Platform Driver` is compatible with `"byu,ecen427-audio_codec"`. The hardware components include the `Audio Controller`, `Switches`, and `Buttons`, which are connected to the `ARM CPU` via `physical addr` and `irq` (interrupt request) lines.

Driver Needs to Handle Interrupts

1. Get IRQ Number
2. Register Interrupt Handler with Linux



One last thing...



Before you actually send data to the CODEC chip, you need to configure it via I2C. I have provided you with a userspace library to do this. Run it before loading your driver.