ISA Drivers

The following text is adapted from the commit message of the initial commit of the ISA bus driver authored by Rene Herman.

During the recent "isa drivers using platform devices" discussion it was pointed out that (ALSA) ISA drivers ran into the problem of not having the option to fail driver load (device registration rather) upon not finding their hardware due to a probe() error not being passed up through the driver model. In the course of that, I suggested a separate ISA bus might be best; Russell King agreed and suggested this bus could use the .match() method for the actual device discovery.

The attached does this. For this old non (generically) discoverable ISA hardware only the driver itself can do discovery so as a difference with the platform_bus, this isa_bus also distributes match() up to the driver.

As another difference: these devices only exist in the driver model due to the driver creating them because it might want to drive them, meaning that all device creation has been made internal as well.

The usage model this provides is nice, and has been acked from the ALSA side by Takashi Iwai and Jaroslav Kysela. The ALSA driver module_init's now (for oldisa-only drivers) become:

```
static int __init alsa_card_foo_init(void)
{
        return isa_register_driver(&snd_foo_isa_driver, SNDRV_CARDS);
}

static void __exit alsa_card_foo_exit(void)
{
        isa_unregister_driver(&snd_foo_isa_driver);
}
```

Quite like the other bus models therefore. This removes a lot of duplicated init code from the ALSA ISA drivers.

The passed in isa_driver struct is the regular driver struct embedding a struct device_driver, the normal probe/remove/shutdown/suspend/resume callbacks, and as indicated that .match callback.

The "SNDRV_CARDS" you see being passed in is a "unsigned int ndev" parameter, indicating how many devices to create and call our methods with.

The platform_driver callbacks are called with a platform_device param; the isa_driver callbacks are being called with a `struct device *dev, unsigned int id` pair directly -- with the device creation completely internal to the bus it's much cleaner to not leak isa_dev's by passing them in at all. The id is the only thing we ever want other then the struct device anyways, and it makes for nicer code in the callbacks as well.

With this additional .match() callback ISA drivers have all options. If ALSA would want to keep the old non-load behaviour, it could stick all of the old .probe in .match, which would only keep them registered after everything was found to be present and accounted for. If it wanted the behaviour of always loading as it inadvertently did for a bit after the changeover to platform devices, it could just not provide a .match() and do everything in .probe() as before.

If it, as Takashi Iwai already suggested earlier as a way of following the model from saner buses more closely, wants to load when a later bind could conceivably succeed, it could use .match() for the prerequisites (such as checking the user wants the card enabled and that port/irq/dma values have been passed in) and .probe() for everything else. This is the nicest model.

To the code...

This exports only two functions; isa_{,un}register_driver().

isa_register_driver() register's the struct device_driver, and then loops over the passed in ndev creating devices and registering them. This causes the bus match method to be called for them, which is:

```
int isa_bus_match(struct device *dev, struct device_driver *driver)
{
        struct isa_driver *isa_driver = to_isa_driver(driver);

        if (dev->platform_data == isa_driver) {
                if (!isa_driver->match ||
                        isa_driver->match(dev, to_isa_dev(dev)->id))
                        return 1;
                dev->platform_data = NULL;
        }
        return 0;
}
```

The first thing this does is check if this device is in fact one of this driver's devices by seeing if the device's platform_data pointer is set to this driver. Platform devices compare strings, but we don't need to do that with everything being internal, so isa_register_driver() abuses dev->platform_data as a isa_driver pointer which we can then check here. I believe platform_data is available for this, but if rather not, moving the isa_driver pointer to the private struct isa_dev is ofcourse fine as well.

Then, if the the driver did not provide a .match, it matches. If it did, the driver match() method is called to determine a match.

If it did **not** match, dev->platform_data is reset to indicate this to isa_register_driver which can then unregister the device again.

If during all this, there's any error, or no devices matched at all everything is backed out again and the error, or -ENODEV, is returned.

isa_unregister_driver() just unregisters the matched devices and the driver itself.

module_isa_driver is a helper macro for ISA drivers which do not do anything special in module init/exit. This eliminates a lot of boilerplate code. Each module may only use this macro once, and calling it replaces module_init and module_exit.

max_num_isa_dev is a macro to determine the maximum possible number of ISA devices which may be registered in the I/O port address space given the address extent of the ISA devices.