I2C subsystem

Mg. Ing. Gonzalo E. Sanchez
MSE - 2022

Implementación de Manejadores de Dispositivos

I2C Subsystem

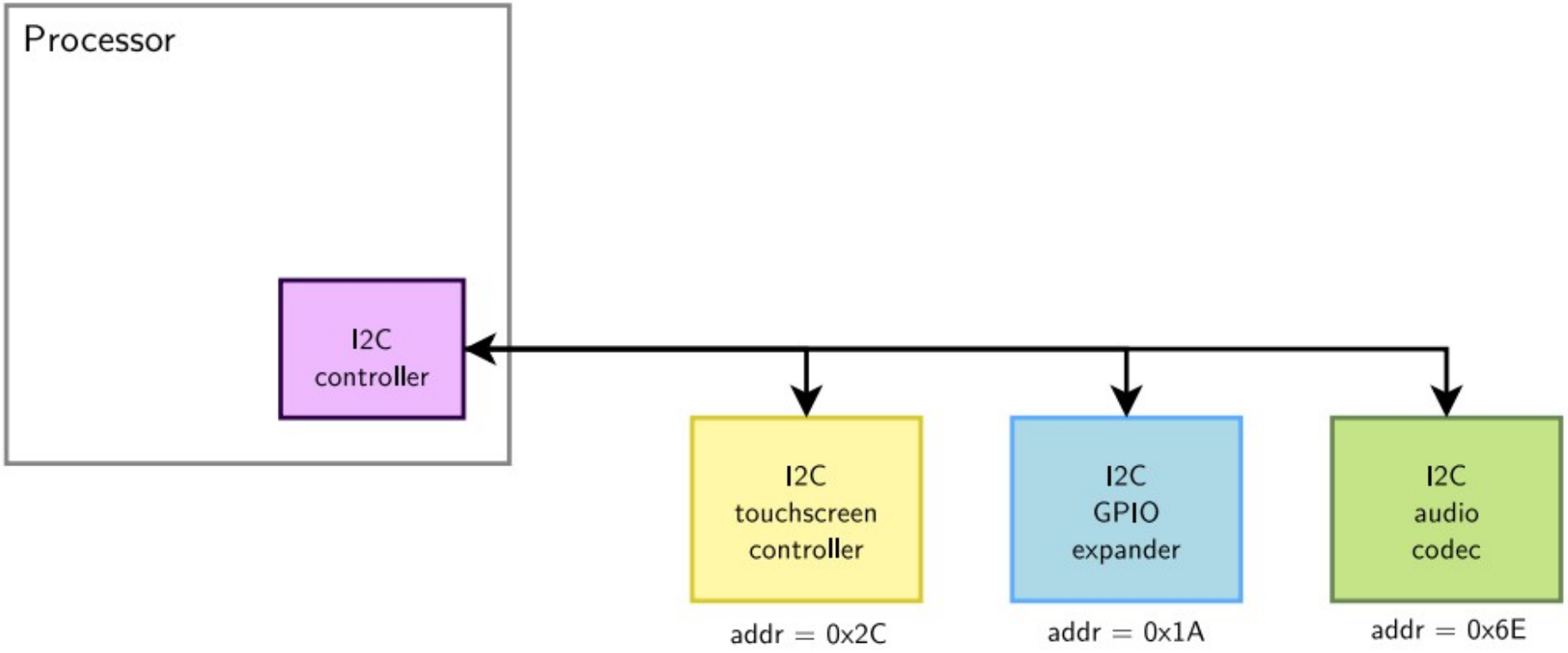
- Introducción
- I2C Clients drivers

Introducción

Introducción

- El bus I2C es muy utilizado para conectar dispositivos de baja velocidad.
- Utiliza solo dos líneas (cables): SDA y SCL.
- Es del tipo master/slave: el esclavo no puede comunicar si el master no inicia la transacción.
- En un sistema linux, el controlador embebido en el SoC es en general el master.
- Cada esclavo tiene su dirección, que es enviada por el master al inicio de cada transacción para identificar el dispositivo.

Introducción



Introducción

- Como cualquier subsistema, el I2C bus driver es responsable de:
  - Proveer una API para implementar drivers controladores I2C.
  - Proveer una API para implementar device drivers I2C en espacio kernel.
  - Proveer una API para implementar device drivers I2C en espacio usuario.
- El núcleo del bus driver I2C está en **drivers/i2c/**.
- Todos los drivers controladores de I2C están en **drivers/i2c/buses**.

Introducción

- Como cualquier subsistema, el I2C subsystem define una estructura **struct i2c_driver** que hereda de **struct device_driver**
- Esta estructura debe ser instanciada y registrada por cada device driver.
- Como deriva de la estructura **device_driver**, necesariamente debe implementar funciones **probe()** y **remove()**.
- Además contiene una lista de IDs (**id_table**) para el probing de dispositivos no basados en DT.

Introducción

- Se utilizan las funciones **i2c_add_driver()** y **i2c_del_driver()** para registrar y anular el registro (unregister).
- Si el driver no hace nada específico en sus funciones **init()** y **exit()** se recomienda utilizar la macro **module_i2c_driver().**
- Esta macro encapsula las definiciones de init y exit.
- Definición de la macro **module_i2c_driver()** (link).
- Observar que hace uso de **i2c_add_driver()** y **i2c_del_driver()**.
- Un ejemplo de driver sencillo (link).

Introducción

- **IMPORTANTE:** En el DT, el controlador de I2C se define en los archivos .dtsi
- En general, están definidos como **status=”disabled”**.
- A nivel de placa/plataforma se debe habilitar el controlador de I2C (**status=”okay”**).
- Se determina la frecuencia de utilización con la propiedad **clock-frequency**
- Los dispositivos se describen como hijos del nodo controlador, donde la propiedad **reg** es su dirección de slave.

Introducción

Definition of the I2C controller

```
i2c0: i2c@01c2ac00 {
        compatible = "allwinner,sun7i-a20-i2c",
                     "allwinner,sun4i-a10-i2c";
        reg = <0x01c2ac00 0x400>;
        interrupts = <GIC_SPI 7 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&apb1_gates 0>;
        status = "disabled";
        #address-cells = <1>;
        #size-cells = <0>;
};
```

Definition of the I2C device

```
&i2c0 {
        pinctrl-names = "default";
        pinctrl-0 = <&i2c0_pins_a>;
        status = "okay";

        axp209: pmic@34 {
                compatible = "x-powers,axp209";
                reg = <0x34>;
                interrupt-parent = <&nmi_intc>;
                interrupts = <0 IRQ_TYPE_LEVEL_LOW>;

                interrupt-controller;
                #interrupt-cells = <1>;
        };
};
```

Introducción

- La función **probe()** es responsable de iniciar el dispositivo y registrarlo en el framework apropiado.
- Como se hizo anteriormente, utilizaremos el framework misc.
- La función **probe()** recibe dos argumentos:
  - Un puntero a **struct i2c_client** que representa el dispositivo en sí.
  - Un puntero a **struct i2c_device_id** que contiene el ID con el que se hizo match del dispositivo al cual se le ejecuta **probe()**.

Introducción

- La función **remove()** es responsable de apagar el dispositivo y anular el registro en el framework apropiado.

- Recibe un solo argumento:
  - Un puntero a **struct i2c_client** que representa el dispositivo en sí. Es el mismo que se pasa a la función **probe()**.

- De esta manera, la inicialización del dispositivo se hace dentro de la función **probe()** y el apagado en la función **remove()**

Introducción

- La API más básica para establecer comunicación con un dispositivo I2C provee dos funciones:
  - **i2c_master_send()** la cual envía el contenido de un buffer.

```
int i2c_master_send(const struct i2c_client *client, const char *buf, int
count);
```

  - **i2c_master_recv()** la cual recibe una cantidad de bytes y la almacena en un buffer.

```
int i2c_master_recv(const struct i2c_client *client, const char *buf, int
count);
```

Introducción

- También hay disponible una API para describir transferencias de varios mensajes en una dirección:

```
int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msgs,
int num)
```

- El puntero struct **i2c_adapter** se obtiene de **client->adapter**.
- La estructura **struct i2c_msg** define la dirección, ubicación y longitud del mensaje.
- Es recomendable utilizar esta función porque permite conformar una transacción completa.

I2C Client drivers

I2C Client drivers

- Desde el punto de vista del mecanismo de match, se debe exponer un array de **device_id**
- Esto hace que los dispositivos sean expuestos a los drivers presentes para luego vincularlos al correspondiente.
- Para poder exponer los dispositivos se instancia **struct i2c_device_id** y se expone mediante **MODULE_DEVICE_TABLE**.
- **ATENCION:** Para esta materia se utiliza solamente la exposición mediante DT.

I2C Client drivers

- Para poder configurar un dispositivo i2c existen esencialmente dos pasos:
  - Definir y registrar el driver I2C.
  - Definir y registrar los dispositivos I2C.
- Definir el driver se resume a instanciar una estructura **struct i2c_driver** y en ella:
  - Asignar los punteros a funciones **probe()** y **remove().**
  - Proveer un nombre de driver.
  - Proveer información sobre el ID de los dispositivos (**of_device_id**).

I2C Client drivers

- Registrar el driver se resume a utilizar la macro **module_i2c_driver**
- Se pasa como argumento la estructura **i2c_driver** instanciada en el paso anterior.
- La definición de los dispositivos se resume a:
  - Declarar una instancia de un dispositivo correspondiente al framework a utilizar (para esta materia **struct miscdevice**).
  - Instanciar una estructura **struct of_device_id** (DT compatible).
  - Instanciar un puntero **struct i2c_client** para tener acceso al dispositivo.

I2C Client drivers

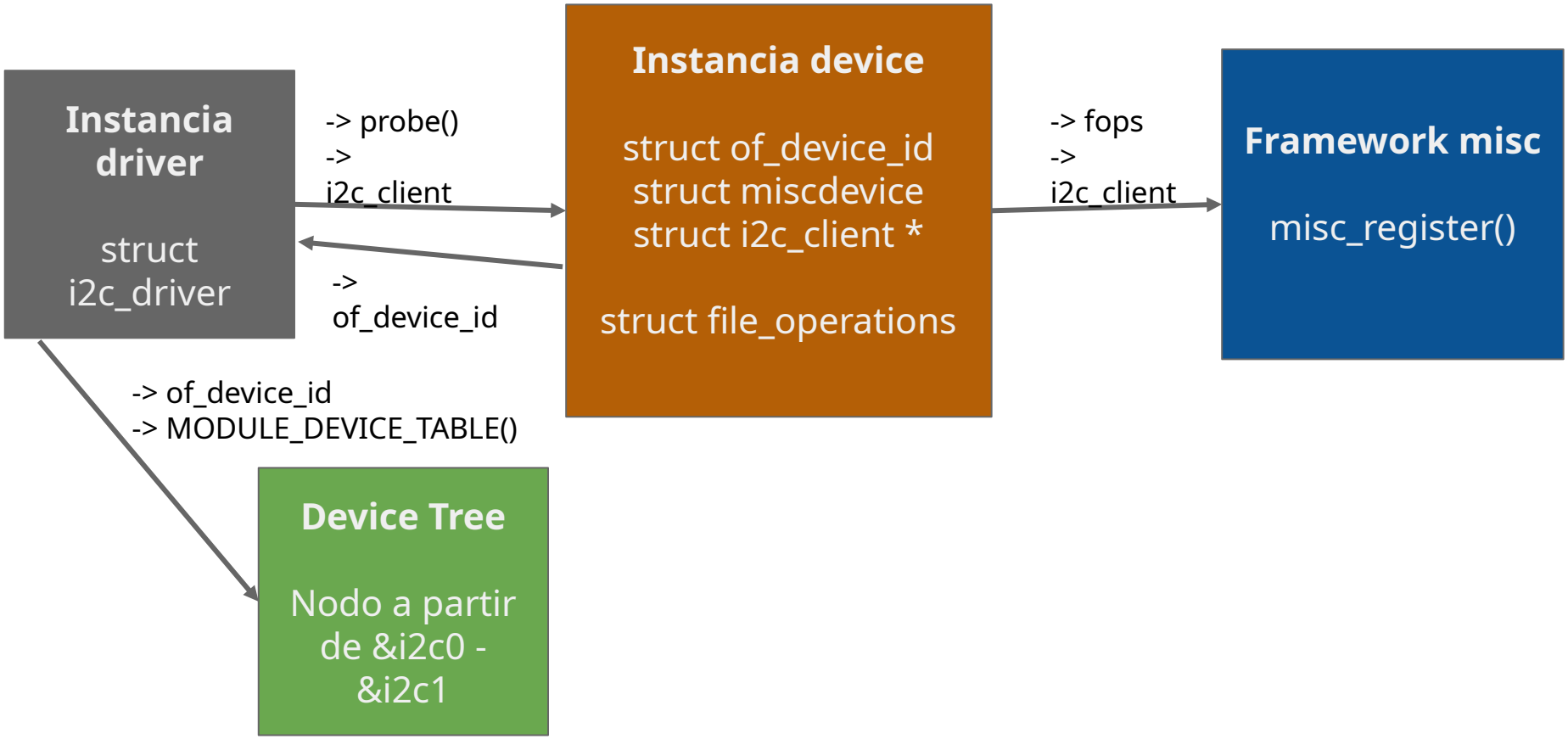
- Registrar el dispositivo con el framework misc es utilizar la función **misc_register()** con el puntero **struct miscdevice**.

- Ahora la pregunta del millón: Cómo se vincula el dispositivo registrado mediante misc framework con el driver registrado?

I2C Client drivers

- Respuesta: Mediante el puntero struct i2c_client devuelto por **probe()**
- Es este puntero de cliente que se utiliza para las funciones de lectura/escritura **i2c_master_recv()** e **i2c_master_send()**
- A fin de cuentas, las *fops* write/read/ioctl terminan utilizando el puntero **struct i2c_client** mediante:
  - **i2c_master_recv()**
  - **i2c_master_send()**
  - **i2c_transfer()**

I2C Client drivers



I2C Client drivers

- **ATENCIÓN!!!!**
- Para versiones de kernel anteriores a 4.10 es necesario tener definido **id_table** (que es la forma non-DT de proveer IDs).
- Esto quiere decir que aunque solo se utilice el estilo de match OF, se debe definir de todas maneras **id_table**.
- Esto fue actualizado en versiones posteriores, por eso en el ejemplo solo se define la estructura **of_device_id.**
- Ver página 175 Linux Device Drivers Development.

I2C Client drivers

- Haciendo un resumen de los pasos a seguir:
- **PASO 1:**
  - Declarar IDs soportados por el driver (mediante **of_device_id**).
- **PASO 2:**
  - Invocar a MODULE_DEVICE_TABLE(of, my_of_match_table) para exponer el dispositivo.

I2C Client drivers

- **PASO 3:**
  - Escribir las funciones **probe()** y **remove()**, teniendo en cuenta que el comportamiento del driver depende mucho de **probe()**.
  - **remove()** debe deshacer todo lo que hizo **probe()**.
  - Se utiliza **probe()** para registrar el device mediante el framework misc.
  - Mediante el framework misc se instancia el dispositivo y se le asignan las file operations correspondientes.
  - Cada file operation hará uso de las API del I2C Subsystem para leer/escribir archivos.

I2C Client drivers

- **PASO 4:**
  - Declarar y llenar la estructura **i2c_driver**. Los campos **.probe** y **.remove** deben ser asignados con las correspondientes funciones.
  - En la subestructura **.driver**, el campo **.owner** debe ser seteado a THIS_MODULE, también llenar el campo **.name**
  - En la subestructura **.driver** el campo **.of_match_table** se setea con el array **of_device_id**.

I2C Client drivers

- **PASO 5:**
  - Utilizar la macro **module_i2c_driver()** para exponer el driver al kernel, pasando como argumento la estructura **i2c_driver** declarada antes.

Introducción al Device Tree

HANDS ON

1. Comenzar a escribir el modulo driver para I2C
2. Luego de comprobar funcionamiento de probe() y remove() escribir fops
3. Escribir un programa en espacio usuario para interactuar con el dispositivo

Gracias.