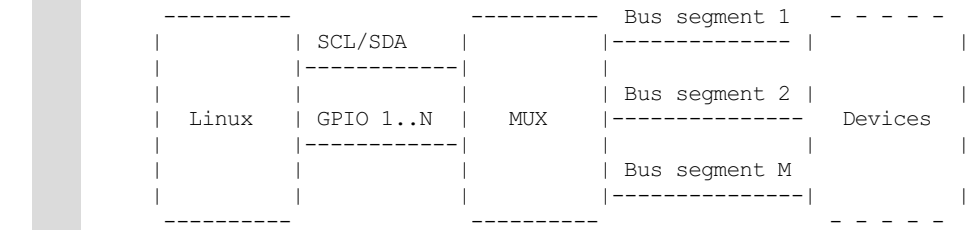
Kernel driver i2c-mux-gpio

Author: Peter Korsgaard <peter.korsgaard@barco.com>

Description

i2c-mux-gpio is an i2c mux driver providing access to I2C bus segments from a master I2C bus and a hardware MUX controlled through GPIO pins.

E.G.:



SCL/SDA of the master I2C bus is multiplexed to bus segment 1..M according to the settings of the GPIO pins 1..N.

Usage

i2c-mux-gpio uses the platform bus, so you need to provide a struct platform_device with the platform_data pointing to a struct i2c_mux_gpio_platform_data with the I2C adapter number of the master bus, the number of bus segments to create and the GPIO pins used to control it. See include/linux/platform_data/i2c-mux-gpio.h for details.

E.G. something like this for a MUX providing 4 bus segments controlled through 3 GPIO pins:

```
#include <linux/platform_data/i2c-mux-gpio.h>
#include <linux/platform_device.h>

static const unsigned myboard_gpiomux_gpios[] = {
      AT91_PIN_PC26, AT91_PIN_PC25, AT91_PIN_PC24
};

static const unsigned myboard_gpiomux_values[] = {
      0, 1, 2, 3
};

static struct i2c_mux_gpio_platform_data myboard_i2cmux_data = {
      .parent         = 1,
      .base_nr        = 2, /* optional */
      .values         = myboard_gpiomux_values,
      .n_values       = ARRAY_SIZE(myboard_gpiomux_values),
      .gpios          = myboard_gpiomux_gpios,
      .n_gpios        = ARRAY_SIZE(myboard_gpiomux_gpios),
      .idle           = 4, /* optional */
};

static struct platform_device myboard_i2cmux = {
      .name           = "i2c-mux-gpio",
      .id             = 0,
      .dev            = {
              .platform_data  = &myboard_i2cmux_data,
      },
};
```

If you don't know the absolute GPIO pin numbers at registration time, you can instead provide a chip name (.chip_name) and relative GPIO pin numbers, and the i2c-mux-gpio driver will do the work for you, including deferred probing if the GPIO chip isn't immediately available.

Device Registration

When registering your i2c-mux-gpio device, you should pass the number of any GPIO pin it uses as the device ID. This guarantees that every instance has a different ID.

Alternatively, if you don't need a stable device name, you can simply pass PLATFORM_DEVID_AUTO as the device ID, and the platform core will assign a dynamic ID to your device. If you do not know the absolute GPIO pin numbers at registration time, this is even the only option.