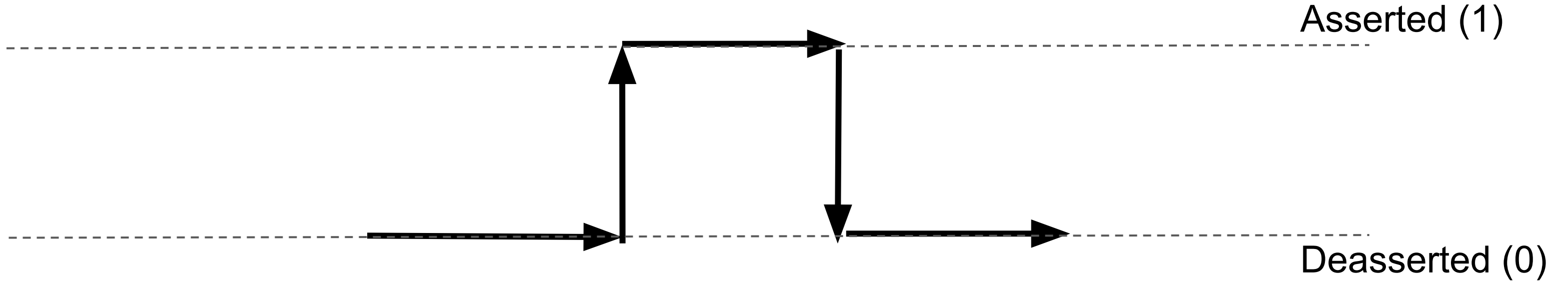
GPIO AND PIN CONTROL OVERVIEW

Linus Walleij

Linaro

WHAT IS GPIO?



APPLICATIONS

AUTOMATIC CONTROL

- POWER PLANTS
- FACTORY LINES
- ROBOTICS
- SDR
- HYDRATION
- ALARMS
- FARMING
- AUTOMOTIVE
- SPACECRAFTS
- VENTILATION

EMBEDDED HOUSEKEEPING

- LED INDICATORS
- REGULATOR
- SD CARD SLOT
- RANDOM KEYS
- TILT SWITCH
- RESET
- HEADSET PLUG
- POWER OFF

BIT-BANGING

- W1
- FLASH
- SPI (MMC and SD!)
- I2C
- MDIO
- PWM

GPIO IN THE LINUX KERNEL

GPIO drivers: follow the examples:

- ls drivers/gpio/gpio-*
- #include <linux/gpio/driver.h>

GPIO consumers:

- #include <linux/gpio/consumer.h>
- Any driver in the kernel that only includes this header is a good example
- **Good examples:** Documentation/driver-api/drivers-on-gpio-rst

GPIO configuration:

- Device Tree
- ACPI DSDT
- Board files using #include <linux/gpio/machine.h>

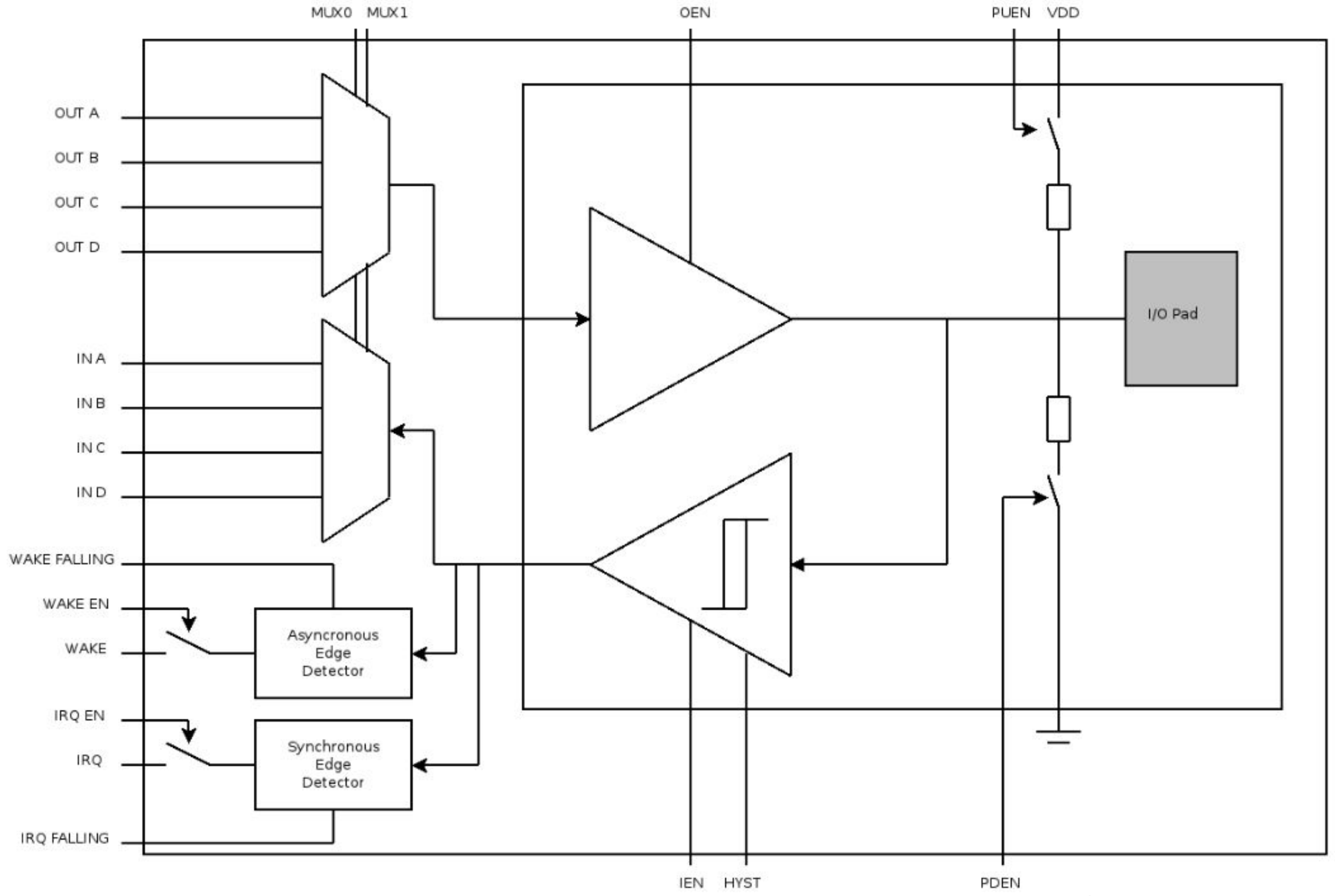
GPIO IN THE LINUX USERSPACE

SYSFS
/sys/class/gpio

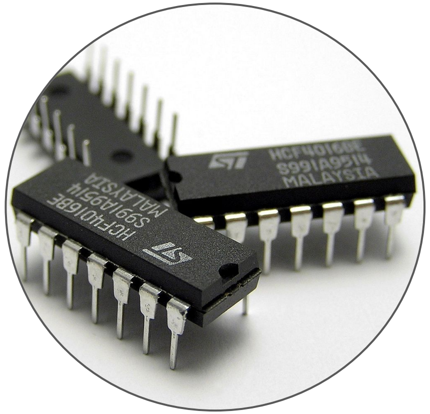
CHARDEV
/sys/bus/gpiochipN
/dev/gpiochipN

GOOD EXAMPLES:
tools/gpio/*
libgpiod ✓

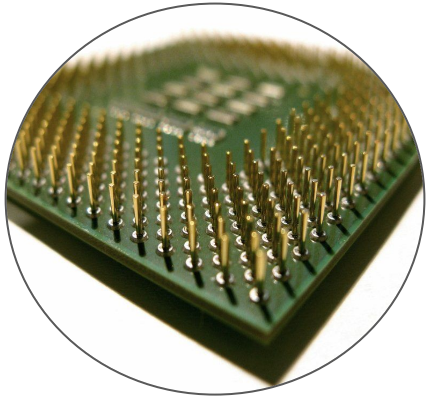
WHAT IS PIN CONTROL?



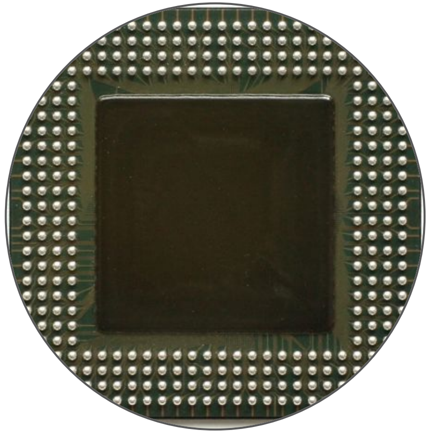
Dual In-Line (DIL) packages



Pin Grid Array (PGA) package



Ball Grid Array (BGA) package



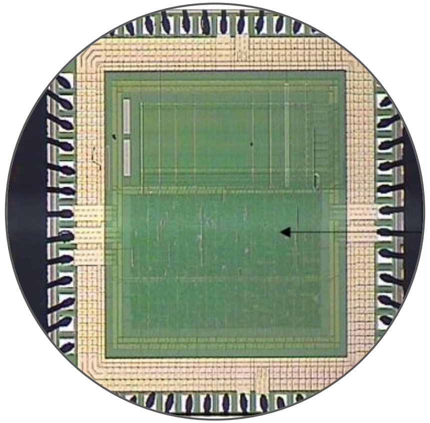
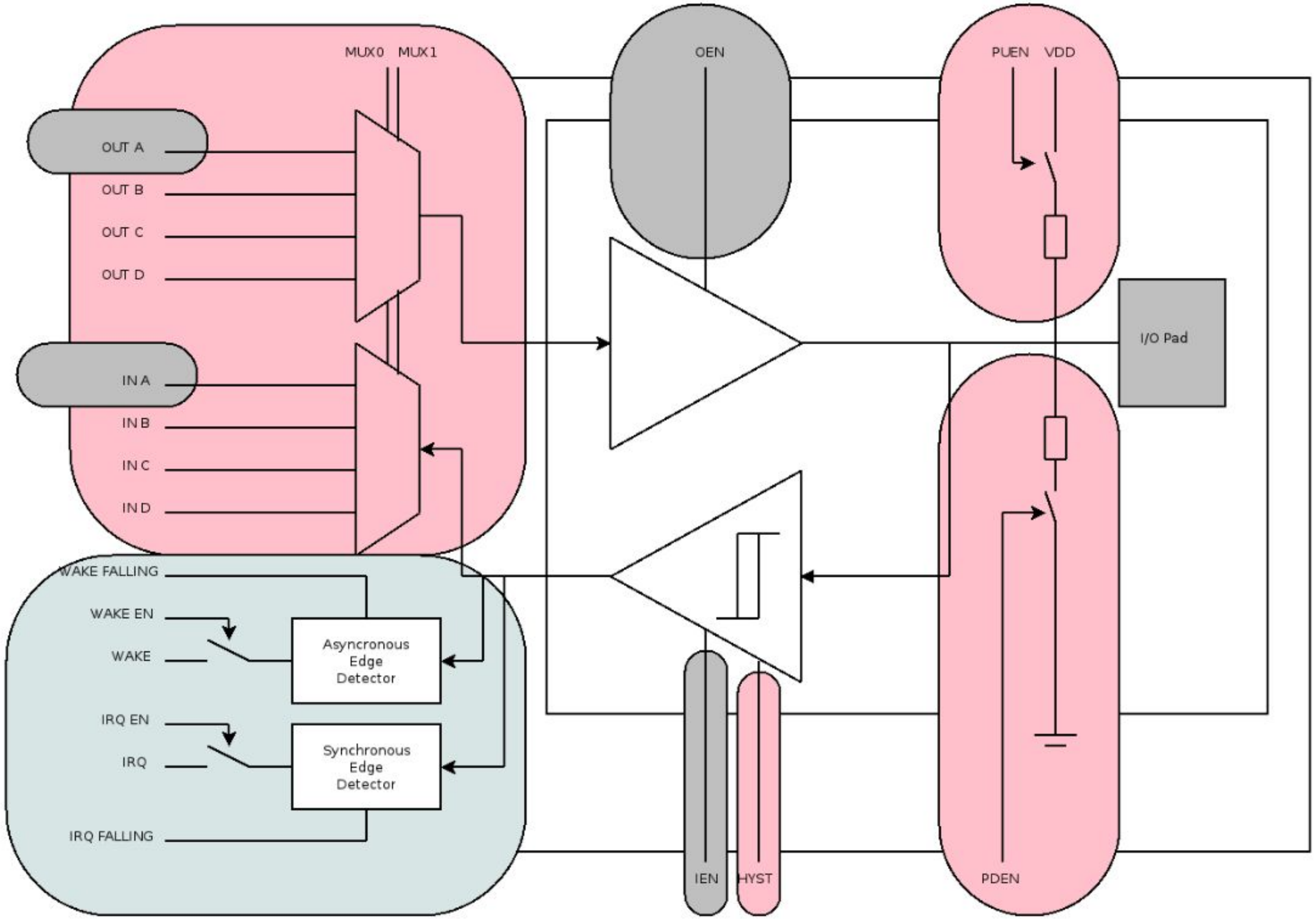
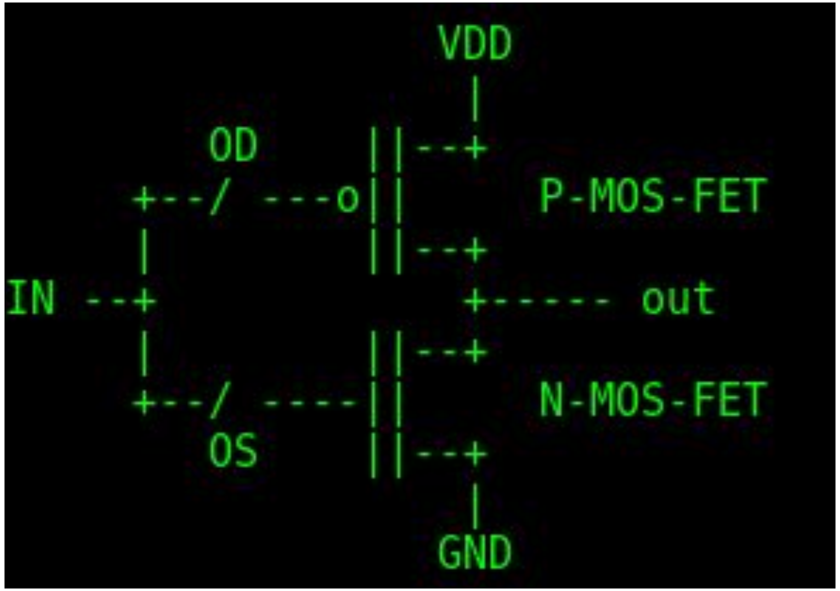


Image sources: the linked Wikipedia entries


MUX0
MUX1
OEN
PUEN
VDD
OUT A
OUT B
OUT C
OUT D
IN A
IN B
IN C
IN D
I/O Pad
WAKE FALLING
WAKE EN
WAKE
Asyncronous Edge Detector
IRQ EN
IRQ
Synchronous Edge Detector
IRQ FALLING
IEN
HYST
PDEN

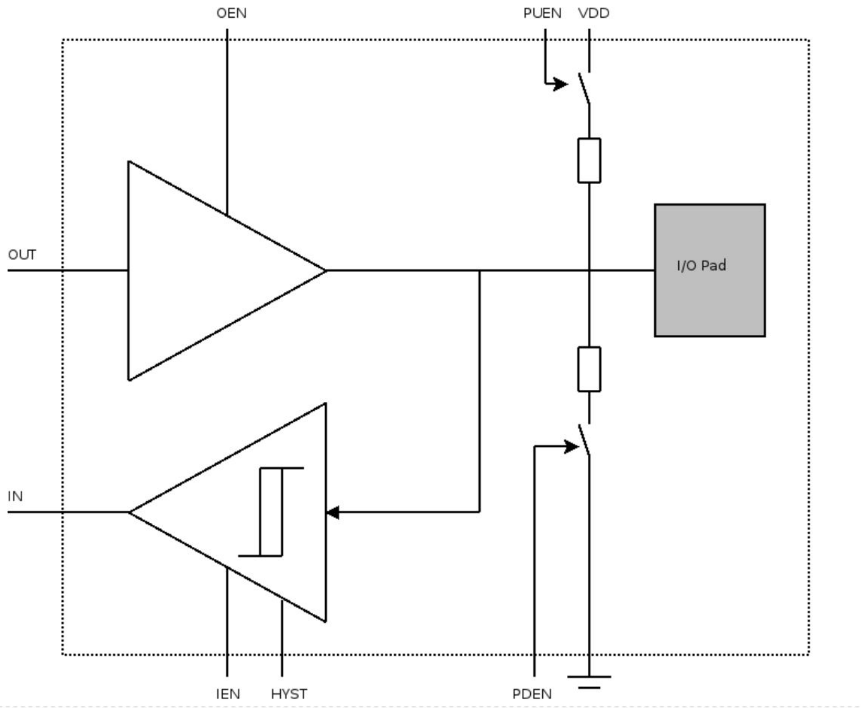
PIN CONFIG BUSINESS

- **Biasing**: pull up/down, bus hold (weak latch), high impedance (tristate)
- **Driving**: push-pull, open drain/source, drive strength (number of driver stages)
- **Debouncing**
- **Schmitt trigger**
- **Low power modes**
- **Power source selection**
- **Sleep state set-up**
- **Slew rate**
- **Signal skew or delay**
- **State persistance**



```
                VDD
                 |
       OD    ||--+
     +--/ ---o||     P-MOS-FET
     |       ||--+
IN --+           +------ out
     |       ||--+
     +--/ ----||     N-MOS-FET
       OS    ||--+
                 |
                GND
```

LET’S COOK SOMETHING FROM THIS

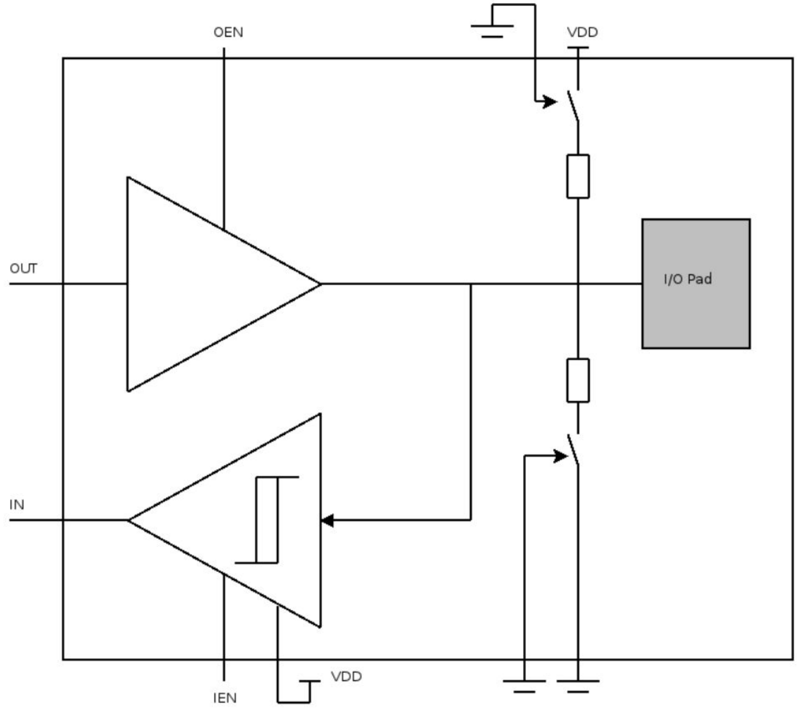


MAKING A GPIO BLOCK FROM A CELL

Simple GPIO block register map after Ctrl+C Ctrl+V x8 of this cell for 8 pins

0x00 8 bits MY_GPIO_OEN
0x01 8 bits MY_GPIO_OUT
0x02 8 bits MY_GPIO_IEN
0x03 8 bits MY_GPIO_INVAL

Use **GPIO_GENERIC**



GPIOLIB

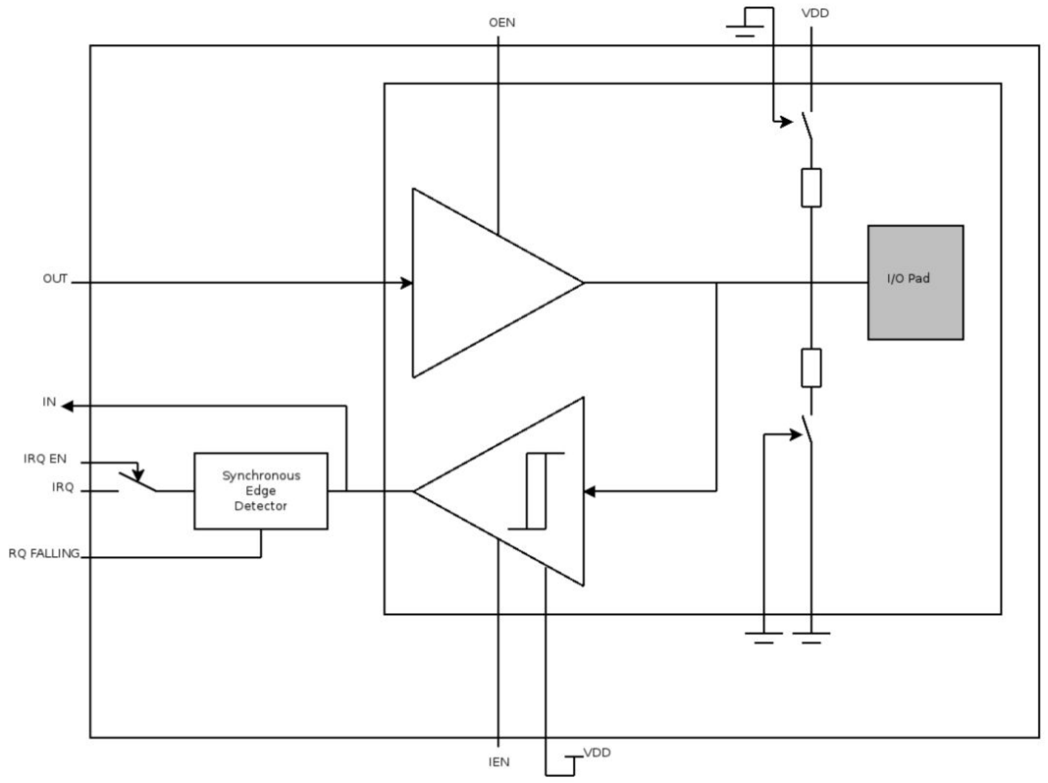
```
struct gpio_chip {
        const char              *label;
        struct gpio_device      *gpiodev;
        struct device           *parent;
        struct module           *owner;

        int                     (*request)(struct gpio_chip *chip,
                                                unsigned offset);
        void                    (*free)(struct gpio_chip *chip,
                                                unsigned offset);
        int                     (*get_direction)(struct gpio_chip *chip,
                                                unsigned offset);
        int                     (*direction_input)(struct gpio_chip *chip,
                                                unsigned offset);
        int                     (*direction_output)(struct gpio_chip *chip,
                                                unsigned offset, int
value);
        int                     (*get)(struct gpio_chip *chip,
                                                unsigned offset);
        int                     (*get_multiple)(struct gpio_chip *chip,
                                                unsigned long *mask,
                                                unsigned long *bits);
        void                    (*set)(struct gpio_chip *chip,
                                                unsigned offset, int
value);
        void                    (*set_multiple)(struct gpio_chip *chip,
                                                unsigned long *mask,
                                                unsigned long *bits);
        int                     (*set_config)(struct gpio_chip *chip,
                                              unsigned offset,
                                              unsigned long config);
        int                     (*to_irq)(struct gpio_chip *chip,
                                                unsigned offset);
(...)
```

```
#if IS_ENABLED(CONFIG_GPIO_GENERIC)
        unsigned long (*read_reg)(void __iomem *reg);
        void (*write_reg)(void __iomem *reg, unsigned long
              data);
        bool be_bits;
        void __iomem *reg_dat;
        void __iomem *reg_set;
        void __iomem *reg_clr;
        void __iomem *reg_dir;
        int bgpio_bits;
        spinlock_t bgpio_lock;
        unsigned long bgpio_data;
        unsigned long bgpio_dir;
#endif

(...)

};
```

MORE COMPLEX GPIO BLOCK WITH IRQ

0x00 8 bits MY_GPIO_OEN
0x01 8 bits MY_GPIO_OUT
0x02 8 bits MY_GPIO_IEN
0x03 8 bits MY_GPIO_INVAL
0x04 8 bits MY_GPIO_IRQ_EN
0x05 8 bits MY_GPIO_IRQ_FALLING

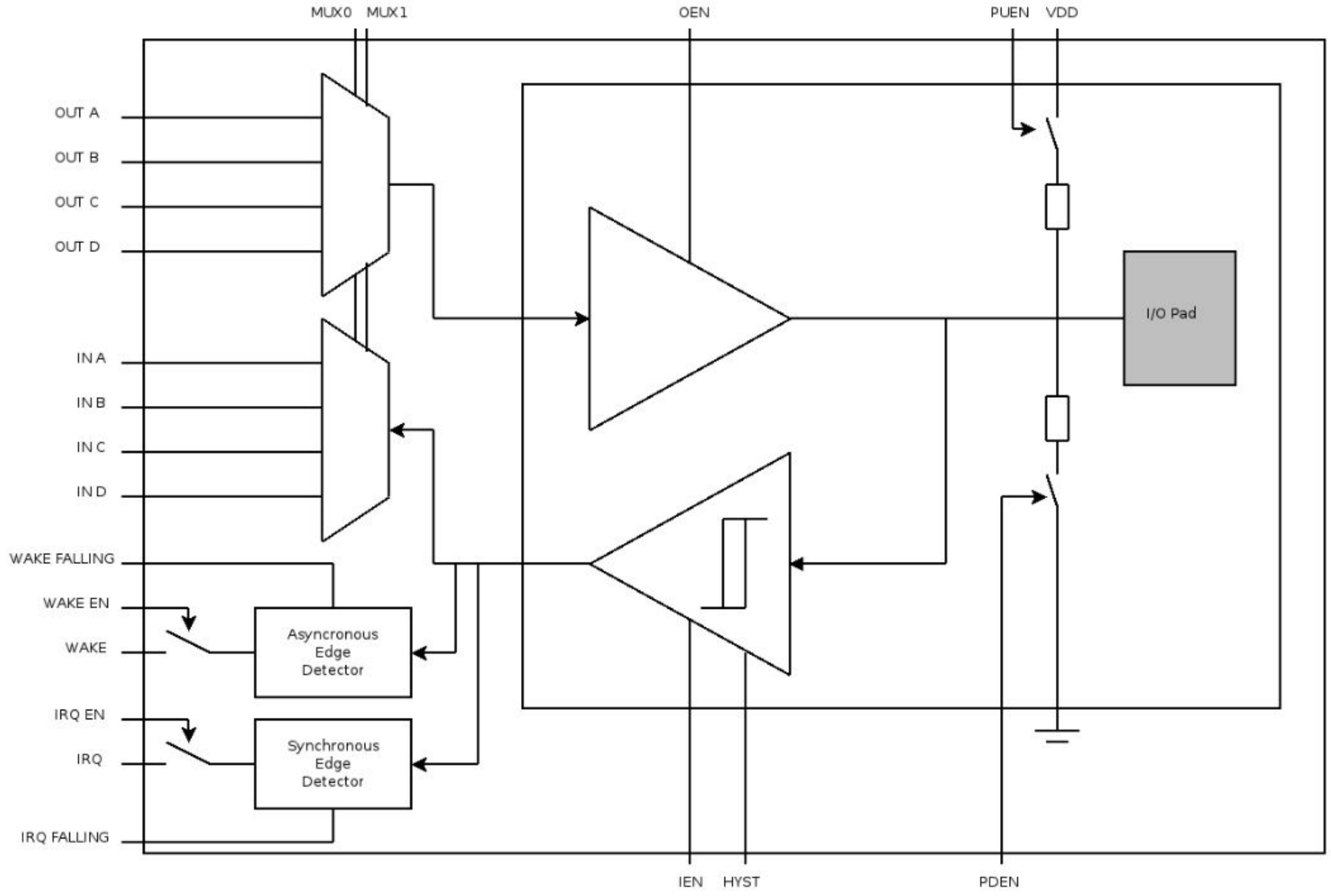
Use **GPIO_GENERIC**
Use **GPIOLIB_IRQCHIP**



GPIOLIB_IRQCHIP

```
struct gpio_irq_chip {
        struct irq_chip *chip;
        struct irq_domain *domain;
        const struct irq_domain_ops *domain_ops;
        irq_flow_handler_t handler;
        unsigned int default_type;
        struct lock_class_key *lock_key;
        struct lock_class_key *request_key;
        irq_flow_handler_t parent_handler;
        void *parent_handler_data;
        unsigned int num_parents;
        unsigned int *parents;
        unsigned int *map;
        bool threaded;
        bool need_valid_mask;
        unsigned long *valid_mask;
        unsigned int first;
};
```

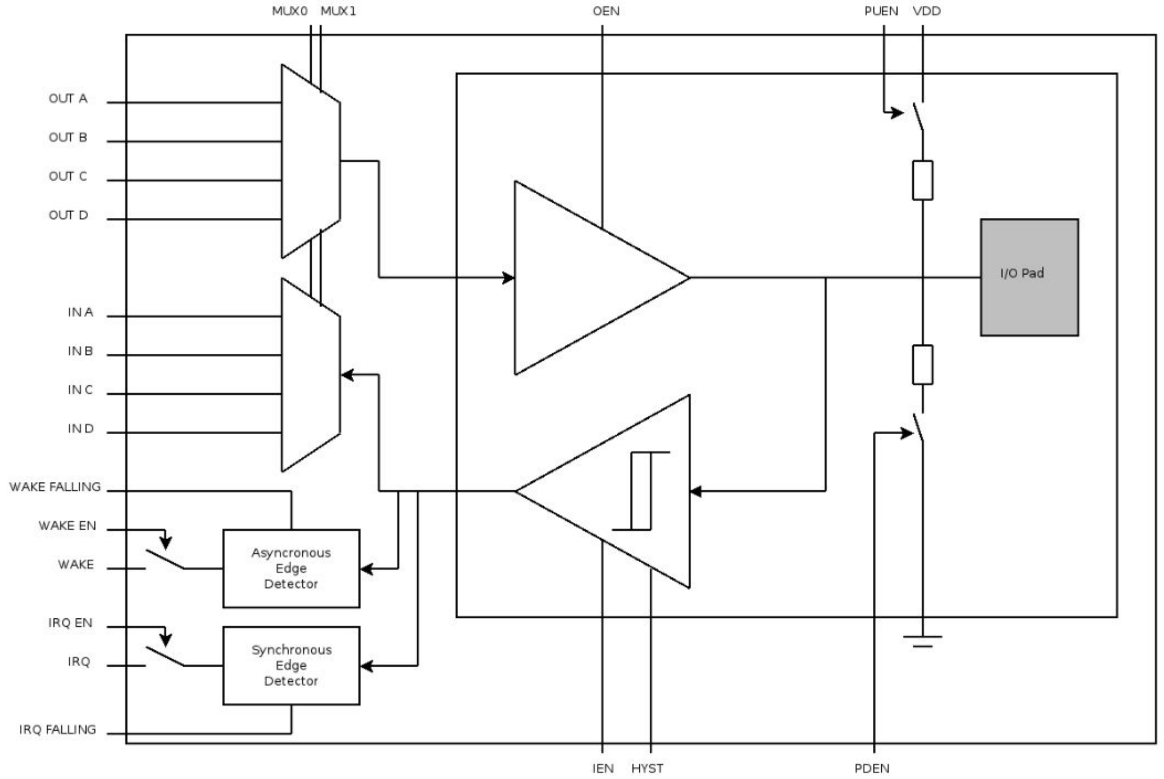
LET'S GET REALLY COMPLEX

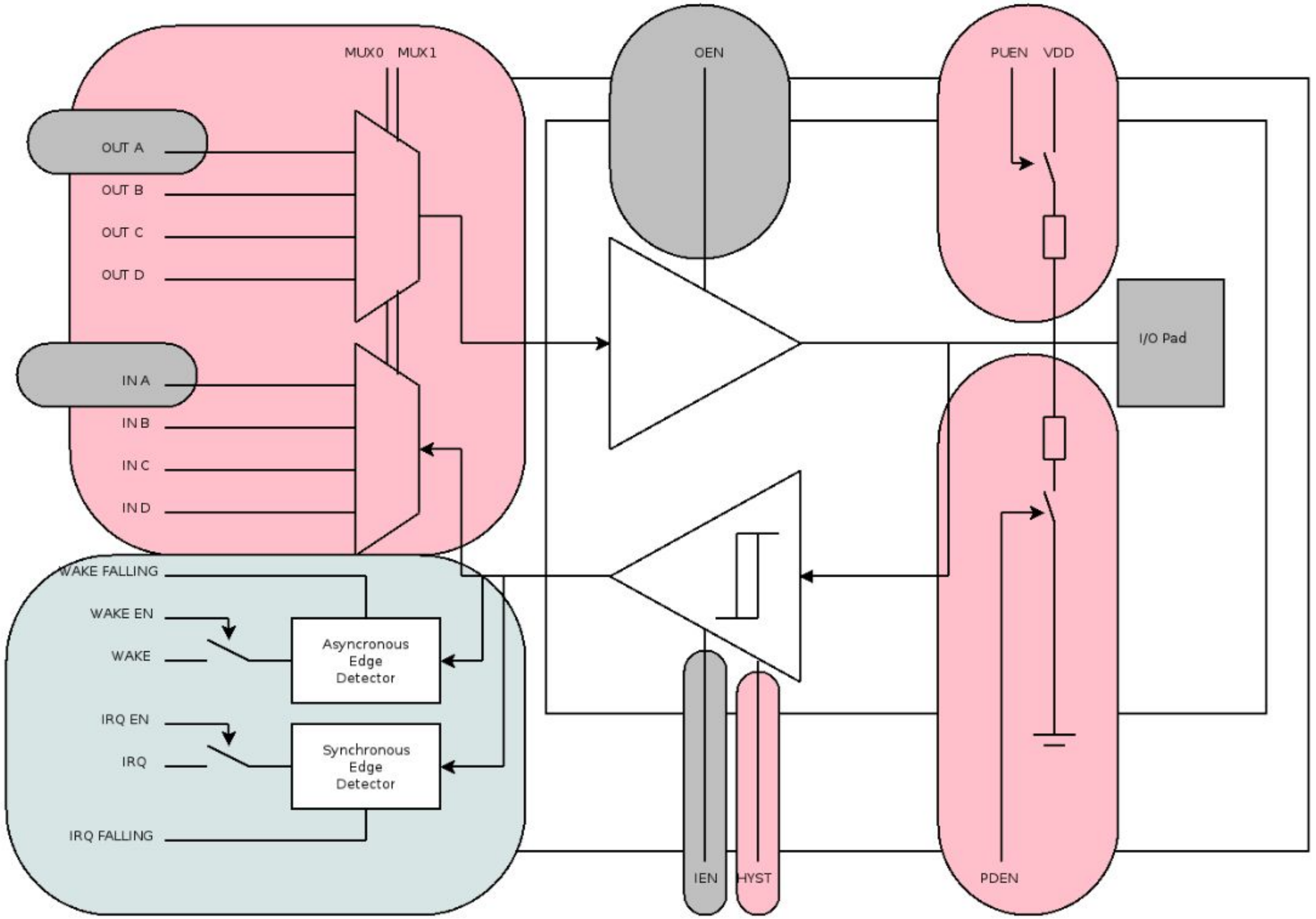


GPIO block register map after Ctrl+C Ctrl+V x8 of this cell for 8 pins

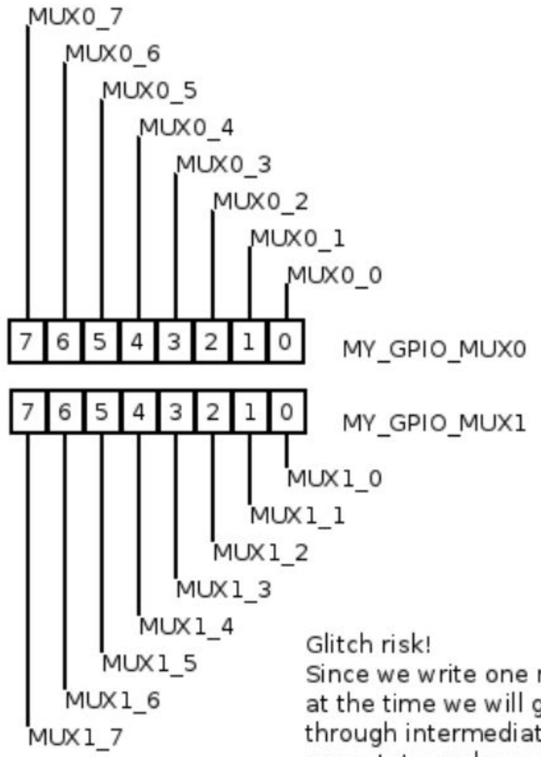
0x00 8 bits MY_GPIO_OEN
0x01 8 bits MY_GPIO_OUT
0x02 8 bits MY_GPIO_IEN
0x03 8 bits MY_GPIO_INVAL
0x04 8 bits MY_GPIO_IRQ_EN
0x05 8 bits MY_GPIO_IRQ_FALLING
0x06 8 bits MY_GPIO_PUEN
0x07 8 bits MY_GPIO_PDEN
0x08 8 bits MY_GPIO_WAKE_EN
0x09 8 bits MY_GPIO_WAKE_FALLING
0x0A 8 bits MY_GPIO_HYST
0x0B 8 bits MY_GPIO_MUX0
0x0C 8 bits MY_GPIO_MUX1

Use **GPIO_GENERIC**
Use **GPIOLIB_IRQCHIP**
Use **PINCTRL**
Use **PINMUX**
Use **GENERIC_PINCONF**

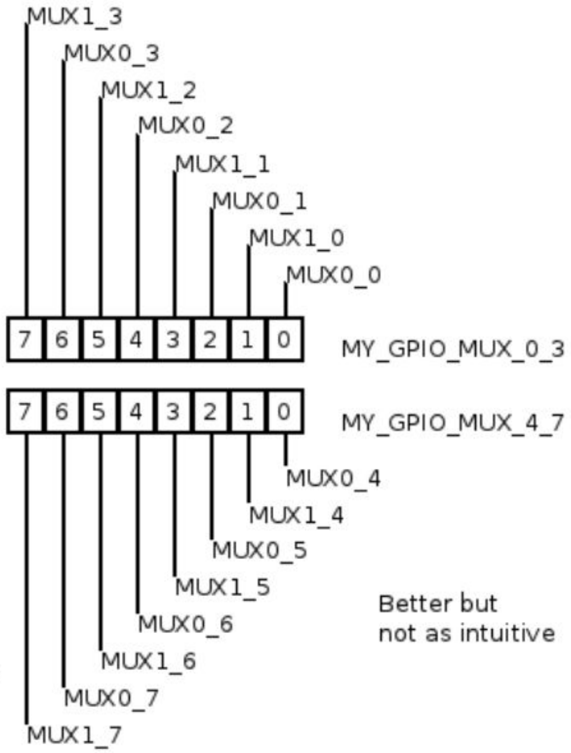



MUX0
MUX1
OEN
PUEN
VDD
OUT A
OUT B
OUT C
OUT D
I/O Pad
IN A
IN B
IN C
IN D
WAKE FALLING
WAKE EN
WAKE
Asyncronous Edge Detector
IRQ EN
IRQ
Synchronous Edge Detector
IRQ FALLING
IEN
HYST
PDEN

MUX0 AND MUX1 REGISTER LAYOUT

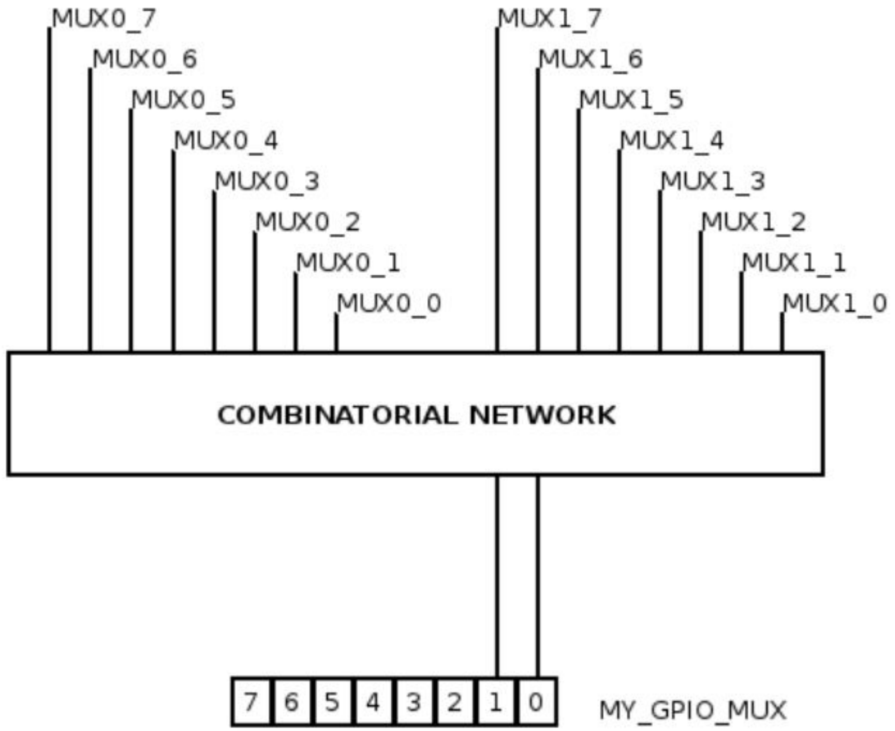


Glitch risk!
Since we write one register at the time we will go through intermediate mux states unless some write trigger logic is used



Better but
not as intuitive

ANOTHER ALTERNATIVE



Bit pattern	Mux setting
00	Device A
01	Device B
10	Device C
11	Device D

HOW DO THEY MATCH UP?

FUTURE IDEAS

- Cleanse the kernel from the global GPIO numberspace and use only descriptors.
- Generic double edge detector simulation (Uwe)
- Generic level from edge simulation (Uwe)
- More pin config properties available to userspace, is this a good or bad idea?

Thank you