Device Driver GPIO

Blinking LEDs from the Kernel

Blinking LEDs from the Kernel

- From: http://derekmolloy.ie/kernel-gpio-programming-buttons-and-leds/

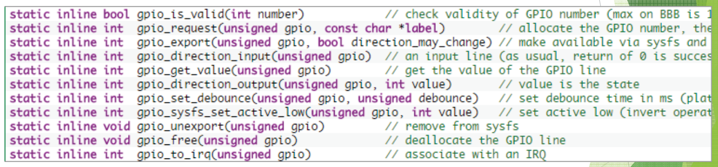
/sys GPIO

- We've seen this before:

```
bone$ cd /sys/class/gpio
bone$ echo 49 > export
export gpio49 gpiochip0   gpiochip32   gpiochip64   gpiochip96   unexport
bone$ cd gpio49
bone$ ls
active_low   device   direction   edge   power   subsystem   uevent   value
bone$ echo out > direction
bone$ echo 1 > value
bone$ echo 0 > value
```

Kernel GPIO calls

- This is much like the /sys interface



```
static inline bool gpio_is_valid(int number)             // check validity of GPIO number (max on BBB is 1
static inline int  gpio_request(unsigned gpio, const char *label)        // allocate the GPIO number, the
static inline int  gpio_export(unsigned gpio, bool direction_may_change) // make available via sysfs and
static inline int  gpio_direction_input(unsigned gpio)  // an input line (as usual, return of 0 is succes
static inline int  gpio_get_value(unsigned gpio)         // get the value of the GPIO line
static inline int  gpio_direction_output(unsigned gpio, int value)       // value is the state
static inline int  gpio_set_debounce(unsigned gpio, unsigned debounce)   // set debounce time in ms (plat
static inline int  gpio_sysfs_set_active_low(unsigned gpio, int value)   // set active low (invert operat
static inline void gpio_unexport(unsigned gpio)          // remove from sysfs
static inline void gpio_free(unsigned gpio)              // deallocate the GPIO line
static inline int  gpio_to_irq(unsigned gpio)            // associate with an IRQ
```

…/include/linux/gpio.h

Interrupts in the Kernel

- LKM driver must register a handler function for the interrupt
- It has the form:

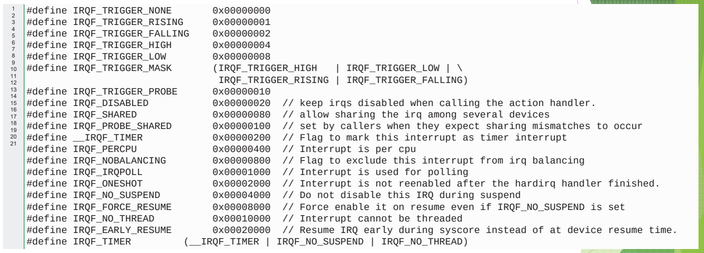
```
static irq_handler_t ebbgpio_irq_handler(unsigned int irq, void *dev_id,
                struct pt_regs *regs) {
   // the actions that the interrupt should perform
   … }
```

Determined automatically

- It is then registered with an interrupt request function:

```
result = request_irq(irqNumber,                // The interrupt number requested
         (irq_handler_t) ebbgpio_irq_handler, // The pointer to the handler function (above)
         IRQF_TRIGGER_RISING,                 // Interrupt is on rising edge (button press in Fig.1)
         "ebb_gpio_handler",                  // Used in /proc/interrupts to identify the owner
         NULL);                               // The *dev_id for shared interrupt lines, NULL here
```

…/include/linux/interrupt.h



```
#define IRQF_TRIGGER_NONE       0x00000000
#define IRQF_TRIGGER_RISING     0x00000001
#define IRQF_TRIGGER_FALLING    0x00000002
#define IRQF_TRIGGER_HIGH       0x00000004
#define IRQF_TRIGGER_LOW        0x00000008
#define IRQF_TRIGGER_MASK       (IRQF_TRIGGER_HIGH   | IRQF_TRIGGER_LOW | \
                                 IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING)
#define IRQF_TRIGGER_PROBE      0x00000010
#define IRQF_DISABLED           0x00000020  // keep irqs disabled when calling the action handler.
#define IRQF_SHARED             0x00000080  // allow sharing the irq among several devices
#define IRQF_PROBE_SHARED       0x00000100  // set by callers when they expect sharing mismatches to occur
#define __IRQF_TIMER            0x00000200  // Flag to mark this interrupt as timer interrupt
#define IRQF_PERCPU             0x00000400  // Interrupt is per cpu
#define IRQF_NOBALANCING        0x00000800  // Flag to exclude this interrupt from irq balancing
#define IRQF_IRQPOLL            0x00001000  // Interrupt is used for polling
#define IRQF_ONESHOT            0x00002000  // Interrupt is not reenabled after the hardirq handler finished.
#define IRQF_NO_SUSPEND         0x00004000  // Do not disable this IRQ during suspend
#define IRQF_FORCE_RESUME       0x00008000  // Force enable it on resume even if IRQF_NO_SUSPEND is set
#define IRQF_NO_THREAD          0x00010000  // Interrupt cannot be threaded
#define IRQF_EARLY_RESUME       0x00020000  // Resume IRQ early during syscore instead of at device resume time.
#define IRQF_TIMER         (__IRQF_TIMER | IRQF_NO_SUSPEND | IRQF_NO_THREAD)
```

/extras/kernel/gpio_test/gpio_test.c

```c
static unsigned int gpioLED = 49;    ///< hard coding the LED gpio for this example to P9_23 (GPIO49)
static unsigned int gpioButton = 115;///< hard coding the button gpio for this example to P9_27
(GPIO115)
static unsigned int irqNumber;       ///< Used to share the IRQ number within this file
static unsigned int numberPresses = 0; ///< For information, store the number of button presses
static bool     ledOn = 0;       ///< Is the LED on or off? Used to invert its state (off by default)

/// Function prototype for the custom IRQ handler function -- see below for the implementation
static irq_handler_t  ebbgpio_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs);
```

/extras/kernel/gpio_test/gpio_test.c

```c
 // Going to set up the LED. It is a GPIO in output mode and will be on by default
  ledOn = true;
  gpio_request(gpioLED, "sysfs");          // gpioLED is hardcoded to 49, request it
  gpio_direction_output(gpioLED, ledOn);   // Set the gpio to be in output mode and on
// gpio_set_value(gpioLED, ledOn);          // Not required as set by line above (here for reference)
  gpio_export(gpioLED, false);             // Causes gpio49 to appear in /sys/class/gpio
                  // the bool argument prevents the direction from being changed
  gpio_request(gpioButton, "sysfs");       // Set up the gpioButton
  gpio_direction_input(gpioButton);        // Set the button GPIO to be an input
  gpio_set_debounce(gpioButton, 200);      // Debounce the button with a delay of 200ms
  gpio_export(gpioButton, false);          // Causes gpio115 to appear in /sys/class/gpio
                  // the bool argument prevents the direction from being changed
  // Perform a quick test to see that the button is working as expected on LKM load
  printk(KERN_INFO "GPIO_TEST: The button state is currently: %d\n",
         gpio_get_value(gpioButton));
```

/extras/kernel/gpio_test/gpio_test.c

```c
   // GPIO numbers and IRQ numbers are not the same! This function performs the mapping for us
   irqNumber = gpio_to_irq(gpioButton);
   printk(KERN_INFO "GPIO_TEST: The button is mapped to IRQ: %d\n", irqNumber);

   // This next call requests an interrupt line
   result = request_irq(irqNumber,             // The interrupt number requested
                        (irq_handler_t) ebbgpio_irq_handler, // The pointer to the handler function below
                        IRQF_TRIGGER_RISING,   // Interrupt on rising edge (button press, not release)
                        "ebb_gpio_handler",    // Used in /proc/interrupts to identify the owner
                        NULL);                 // The *dev_id for shared interrupt lines, NULL is okay

   printk(KERN_INFO "GPIO_TEST: The interrupt request result is: %d\n", result);
   return result;
}
module_init(ebbgpio_init);
module_exit(ebbgpio_exit);
```

Run the module

```
bone$ make
bone$ insmod gpio_test.ko
bone$ dmesg –H | tail -6
[Oct13 12:52] GPIO_TEST: Initializing the GPIO_TEST LKM
[  +0.000116] GPIO_TEST: The button state is currently: 0
[  +0.000027] GPIO_TEST: The button is mapped to IRQ: 145
[  +0.000179] GPIO_TEST: The interrupt request result is: 0
[  +3.702854] GPIO_TEST: Interrupt! (button state is 1)
[  +1.339237] GPIO_TEST: Interrupt! (button state is 1)
```

Interupts

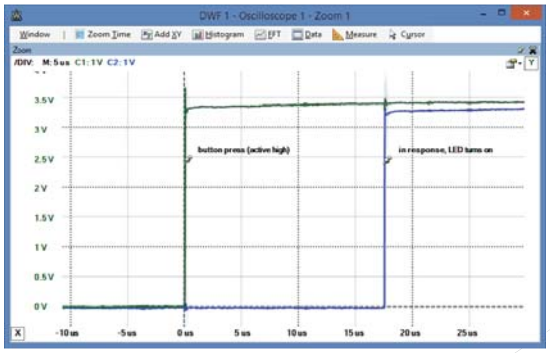
```
bone$ cat /proc/interupts
 CPU0
 16:    4669125      INTC  68 Level     gp_timer
 19:          1      INTC  78 Level     wkup_m3_txev
 20:      12031      INTC  12 Level     49000000.edma_ccint
 22:         76      INTC  14 Level     49000000.edma_ccerrint
 26:          0      INTC  96 Level     44e07000.gpio
 32:          0  44e07000.gpio   5 Edge      gpiolib
                   0.g                                   c cd
                     C              480
 92:                 INTC  3   evel     481a   0.gpio
 25:          2      INTC  62 Level     481ae000.gpio
145:          2  481ae000.gpio  19 Edge      ebb_gpio_handler
158:         19      INTC  72 Level     44e09000.serial
159:      43039      INTC  70 Level     44e0b000.i2c
160:     591097      INTC  30 Level     4819c000.i2c
```

Interrupt Number

Number of Interrupts

Gpio port

Gpio bit



- 15 to 25 µS