

GPIO

Objective

To be able to General Purpose Input Output (GPIO), to generate digital output signals, and to read input signals. Digital outputs can be used as control signals to other hardware, to transmit information, to signal another computer/controller, to activate a switch or, with sufficient current, to turn on or off LEDs, or to make a buzzer sound.

Below will be a discussion on using GPIO to drive an LED.

Although the interface may seem simple, you do need to consider hardware design and know some of the fundamentals of electricity. There are a couple of goals for us:

- No hardware damage if faulty firmware is written
 - The circuit should prevent an excess amount of current to avoid processor damage.
-

Required Background

You should know the following:

- [bit-masking in C](#)
- Wire-wrapping or use of a breadboard
- Fundamentals of electricity, such as Ohm's law ($V = IR$) and how diodes work.

GPIO

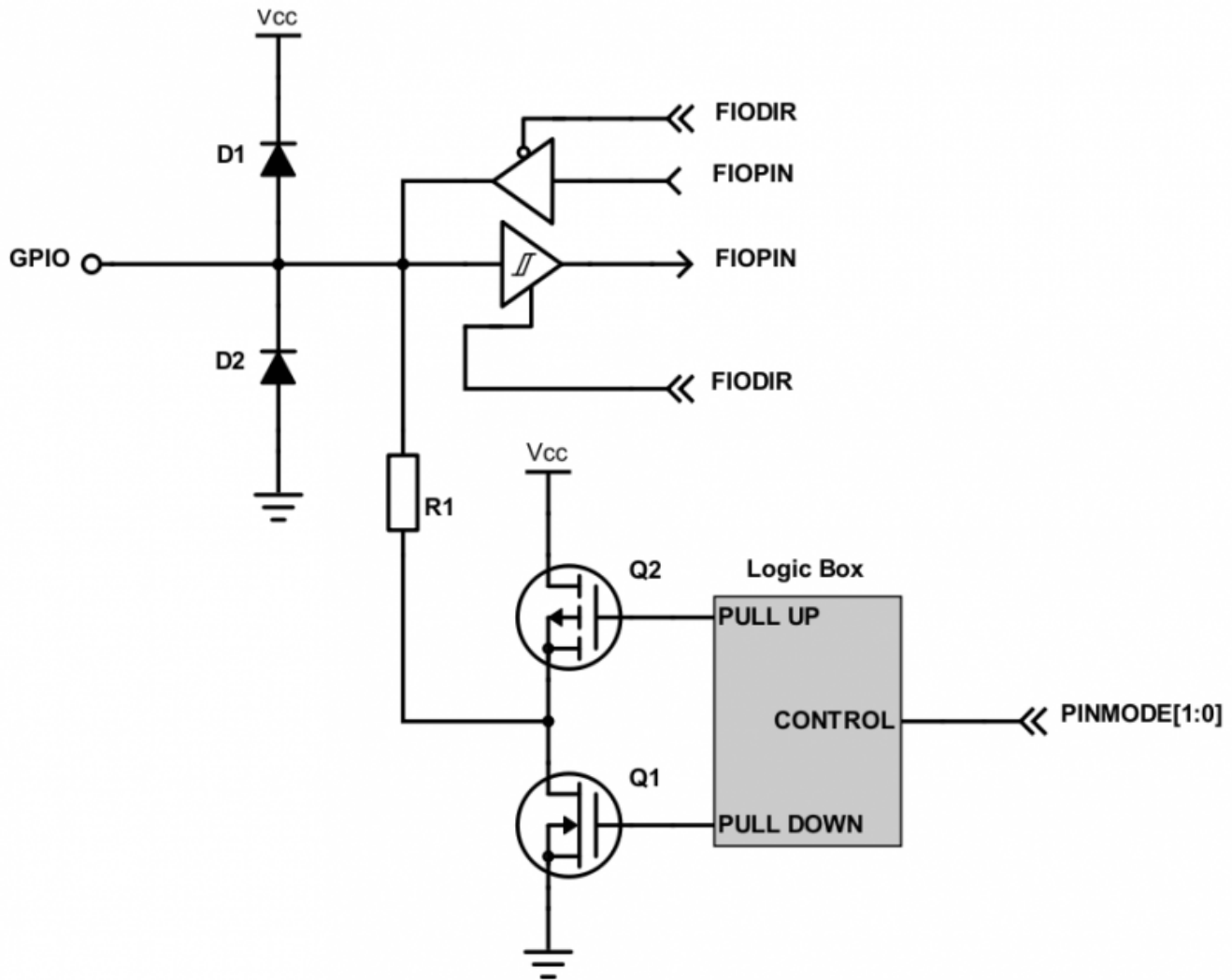


Figure 1. Internal Design of a GPIO

GPIO stands for "General Purpose Input Output". Each pin can at least be used as an output or input. In an output configuration, the pin voltage is either 0v or 3.3v. In input mode, we can read whether the voltage is 0v or 3.3v.

You can locate a GPIO that you wish to use for a switch or an LED by first starting with the schematic of the board. The schematic will show which pins are "available" because some of the microcontroller pins may be used internally by your development board. After you locate a free pin, such as P2.0, then you can look-up the microcontroller user manual to locate the memory that you can manipulate.

Hardware Registers Coding

The hardware registers map to physical pins. If we want to attach our switch and the LED to our microcontroller's PORT0, then reference the relevant registers and their functionality.

8.5 Register description

The registers represent the enhanced I/O registers. These registers are located on an AHB bus and can be accessed in byte, half-word, and word. Each register controls bits in a single GPIO port independently.

Table 94. Register overview: GPIO (base address 0x2009 8100)

| Name | Access | Address offset | Description |
|-------|--------|----------------|-------------------------------|
| DIR0 | R/W | 0x000 | GPIO Port0 Direction Register |
| MASK0 | R/W | 0x010 | Mask register |
| PIN0 | R/W | 0x014 | Port0 Pin Value Register |
| SET0 | R/W | 0x018 | Port0 Output Set Register |
| CLR0 | WO | 0x01C | Port0 Output Clear Register |
| DIR1 | R/W | 0x020 | GPIO Port1 Direction Register |
| MASK1 | R/W | 0x030 | Mask register |
| PIN1 | R/W | 0x034 | Port1 Pin Value Register |
| SET1 | R/W | 0x038 | Port1 Output Set Register |
| CLR1 | WO | 0x03C | Port1 Output Clear Register |
| DIR2 | R/W | 0x040 | GPIO Port2 Direction Register |

Note that in the LPC17xx, the registers had the words `FI0` preceding the `LPC_GPIO` data structure members. In the LPC40xx, the word `FI0` has been dropped. `FI0` was a bit historic and it stood for "Fast Input Output", but in the LPC40xx, this historic term was deprecated.

| LPC40xx Port0 Registers | |
|--------------------------------|---|
| <code>LPC_GPIO0->DIR</code> | The direction of the port pins, 1 = output |
| <code>LPC_GPIO0->PIN</code> | <i>Read:</i> Sensed inputs of the port pins, 1 = HIGH <i>Write:</i> Control voltage level of the pin, 1 = 3.3v |
| <code>LPC_GPIO0->SET</code> | <i>Write only:</i> Any bits written 1 are OR'd with PIN |
| <code>LPC_GPIO0->CLR</code> | <i>Write only:</i> Any bits written 1 are AND'd with PIN |

Switch

We will interface our switch to PORT0.2, or port zero's 3rd pin (counting from 0).

Note that the "inline" resistor is used such that if your GPIO is misconfigured as an OUTPUT pin, hardware damage will not occur from badly written software.

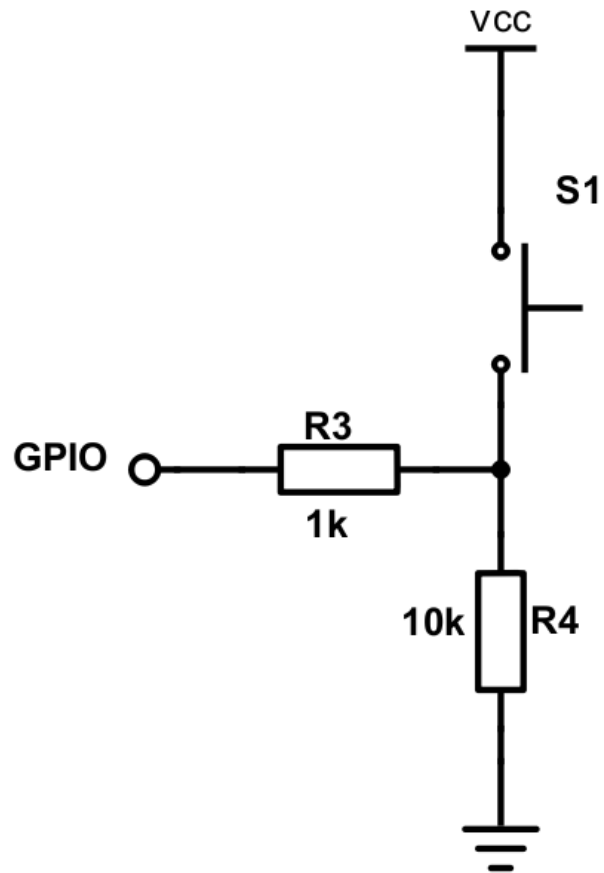


Figure 2. Button Switch Circuit Schematic

```

/* Make direction of PORT0.2 as input */
LPC_GPIO0->DIR &= ~(1 << 2);
/* Now, simply read the 32-bit PIN register, which corresponds to
 * 32 physical pins of PORT0. We use AND logic to test if JUST the
 * pin number 2 is set
 */
if (LPC_GPIO0->PIN & (1 << 2)) {
    // Switch is logical HIGH
} else {
    // Switch is logical LOW}

```

LED

We will interface our LED to PORT0.3, or port zero's 4th pin (counting from 0).

Given below are active-low config

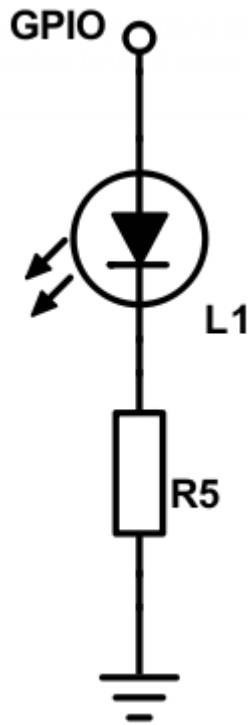


Figure 3. Active High LED circuit schematic

n LED. Usually, the "sink" current ten.

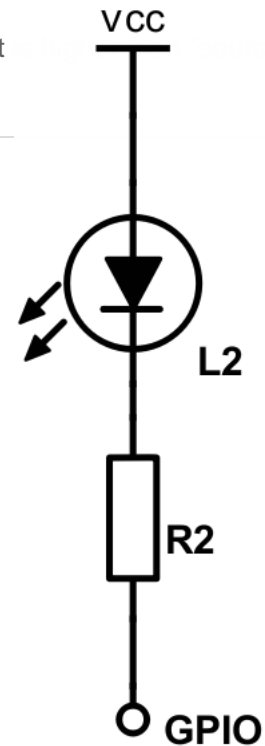


Figure 4. Active low LED circuit schematic

```
const uint32_t led3 = (1U << 3);
/* Make direction of PORT0.3 as OUTPUT */
LPC_GPIO0->DIR |= led3;
/* Setting bit 3 to 1 of IOPIN will turn ON LED
 * and resetting to 0 will turn OFF LED.
 */
LPC_GPIO0->PIN |= led3;
/* Faster, better way to set bit 3 (no OR logic needed) */
LPC_GPIO0->SET = led3;
/* Likewise, reset to 0 */LPC_GPIO0->CLR = led3;
```