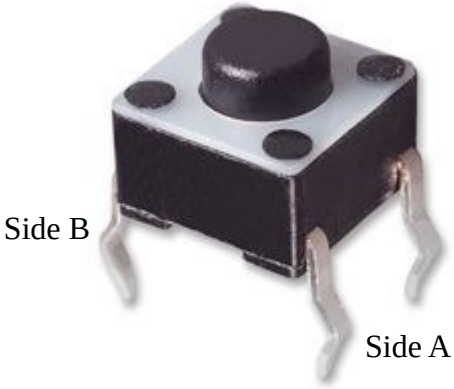
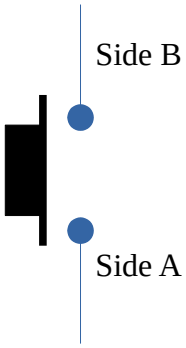
Digital Inputs

We have seen how general purpose output pins can be used to control LED's (and potentially other devices). We will now look at digital inputs – specifically buttons to see how our MCU handles them

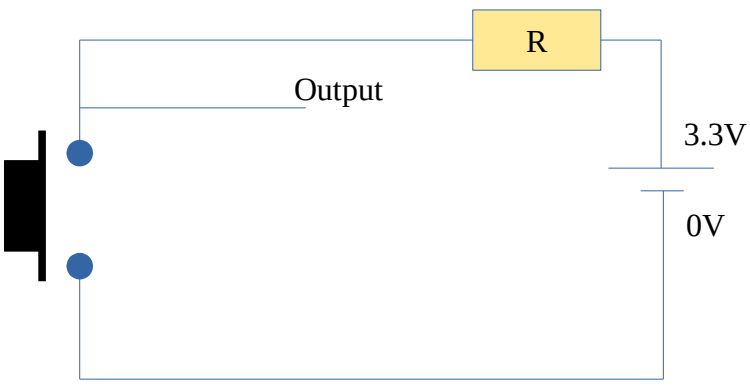
Turning a mechanical movement in to an electrical signal.



The push button shown above is a "momentary" push-button i.e. when pushed it connects the pins on side A to side B; once released the connection is broken. Electrically we can represent this as follows:



We can use this movement to create an electrical signal by connecting the button like this:

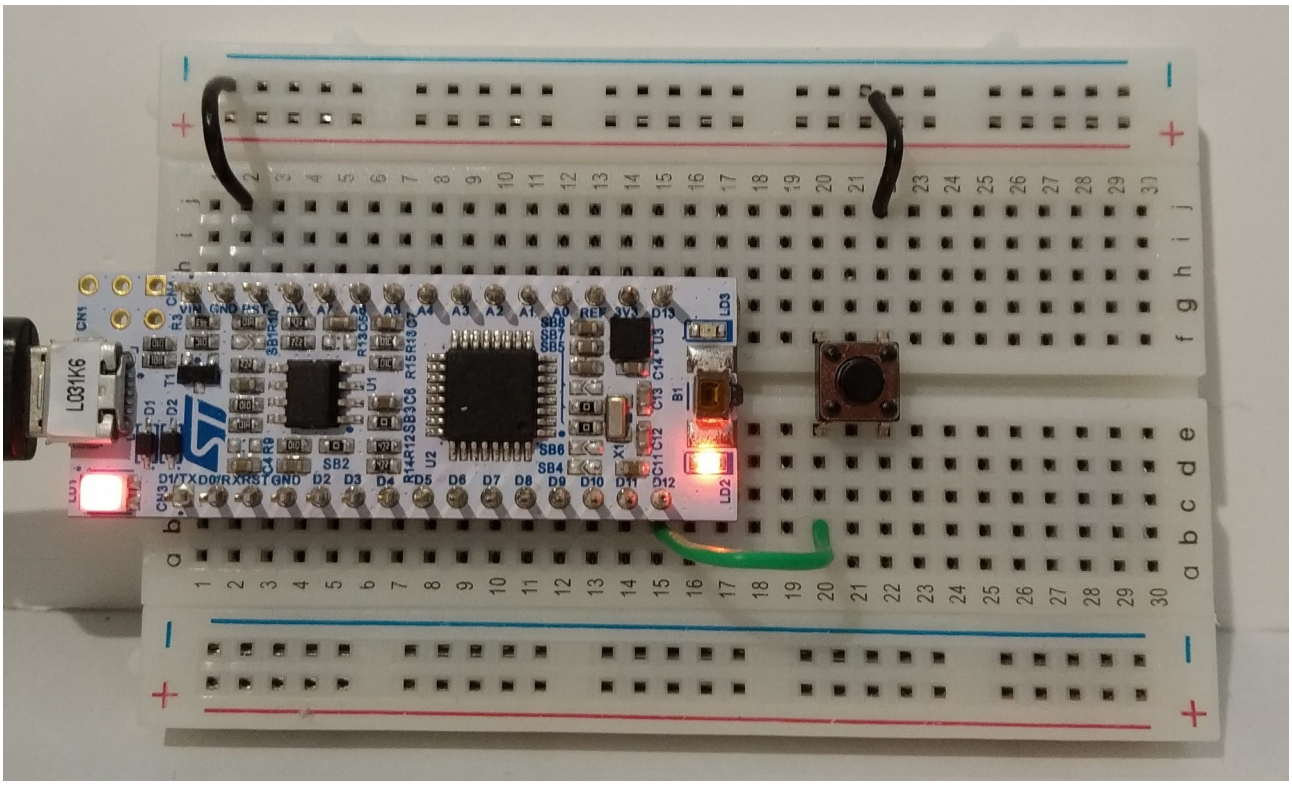


When the button is pressed, the Output signal wire is connected to 0V i.e. the output is Low. When the button is NOT pressed, the Output is connected via a Resistor (R) to 3.3V – the output is now High. The resistor is present to keep current flow to a low level when the button is pressed.

Resistors used in this fashion with buttons are often referred to as **Pull-Up** resistors. This is such a widely used meme that MCU manufacturers now often include the Pull-Up resistors in their devices. These internal Pull-Up resistors are usually disabled by default to save power but can be enabled by setting particular register bits. The STM32L031's internal pull-up resistors can be configured to pull a signal high or low depending on the setting of a pair of bits for each input. In the example that follows, a helper function called enablePullUp is included to simplify pull-up resistor configuration. Further details can be found in the reference manual page 224.

In this example you may notice that there is a 'u' after some of the constant values. This is to eliminate compiler warnings regarding signed/unsigned integer conversion. We will investigate this more later.

Example : Using a button to control the LED.



Code for button controlling an LED

```
#include <stdint.h>
#include <stm32l031xx.h>
void pinMode(GPIO_TypeDef *Port, uint32_t BitNumber, uint32_t Mode);
void enablePullUp(GPIO_TypeDef *Port, uint32_t BitNumber);
void delay(volatile uint32_t dly);
int main()
{
	RCC->IOPENR = RCC->IOPENR | (1 << 1); // Enable GPIOB
	pinMode(GPIOB,3,1); // Make GPIOB_3 (LD3) an output
	pinMode(GPIOB,4,0); // Make GPIOB_4 (Button) an input
	enablePullUp(GPIOB,4); // enable pull-up for GPIOB_4
	while(1)
	{
		if (0 == (GPIOB->IDR & (1u << 4)))
		{
			// Button pressed
			GPIOB->ODR = GPIOB->ODR | (1u << 3); // LD3 On
		}
		else
		{
			// Button not pressed
			GPIOB->ODR = GPIOB->ODR & ~(1u << 3); // LD3 Off
		}
	}
}
void enablePullUp(GPIO_TypeDef *Port, uint32_t BitNumber)
{
	Port->PUPDR = Port->PUPDR & ~(3u << BitNumber*2); // clear pull-up resistor bits
	Port->PUPDR = Port->PUPDR | (1u << BitNumber*2); // set pull-up bit
}
void pinMode(GPIO_TypeDef *Port, uint32_t BitNumber, uint32_t Mode)
{
	// This function writes the given Mode bits to the appropriate location for
	// the given BitNumber in the Port specified.  It leaves other bits unchanged
	// Mode values:
	// 0 : digital input
	// 1 : digital output
	// 2 : Alternative function
	// 3 : Analog input
	uint32_t mode_value = Port->MODER; // read current value of Mode register
	Mode = Mode << (2 * BitNumber);    // There are two Mode bits per port bit so need to shift

	   // the mask for Mode up to the proper location
	mode_value = mode_value & ~(3u << (BitNumber * 2)); // Clear out old mode bits
	mode_value = mode_value | Mode; // set new bits
	Port->MODER = mode_value; // write back to port mode register
}
void delay(volatile uint32_t dly)
{
	while(dly--);
}
```