# attachInterrupt()

## Description

Specifies a function to call when an external interrupt occurs. Replaces any previous function that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3). The table below shows the available interrupt pins on various boards.

| Board | int.0 | int.1 | int.2 | int.3 | int.4 | int.5 |
|---|---|---|---|---|---|---|
| Uno, Ethernet | 2 | 3 | | | | |
| Mega2560 | 2 | 3 | 21 | 20 | 19 | 18 |
| Leonardo | 3 | 2 | 0 | 1 | 7 | |
| Due | | | **(see below)** | | | |

The Arduino Due board has powerful interrupt capabilities that allows you to attach an interrupt function on all available pins. You can directly specify the pin number in attachInterrupt().

## Syntax

attachInterrupt(interrupt, function, mode)
attachInterrupt(pin, function, mode) (Arduino Due only)

## Parameters

| | | |
|---|---|---|
| **interrupt**: | the number of the interrupt (int) | |
| **pin**: | the pin number | (Arduino Due only) |
| **function**: | the function to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine. | |
| **mode**: | defines when the interrupt should be triggered. Four contstants are predefined as valid values:<br>- **LOW** to trigger the interrupt whenever the pin is low,<br>- **CHANGE** to trigger the interrupt whenever the pin changes value<br>- **RISING** to trigger when the pin goes from low to high,<br>- **FALLING** for when the pin goes from high to low.<br>The Due board allows also:<br>- **HIGH** to trigger the interrupt whenever the pin is high. | (Arduino Due only) |

## Returns

none

## Note

Inside the attached function, delay() won't work and the value returned by millis() will not increment. Serial data received while in the function may be lost. You should declare as volatile any variables that you modify within the attached function.

## Using Interrupts

Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. Good tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.

If you wanted to insure that a program always caught the pulses from a rotary encoder, so that it never misses a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, in order to catch

pulses when they occurred. Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all of these situations, using an interrupt can free the microcontroller to get some other work done while not missing the input.

## Example

```
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}

void blink()
{
  state = !state;
}
```

[Get Code] (http://arduino.cc/en/Reference/AttachInterrupt?action=sourceblock&num=1)

## See also

- detachInterrupt (http://arduino.cc/en/Reference/DetachInterrupt)

Reference Home (http://arduino.cc/en/Reference/HomePage)

Corrections, suggestions, and new documentation should be posted to the Forum (http://arduino.cc/forum/index.php/board,23.0.html).