

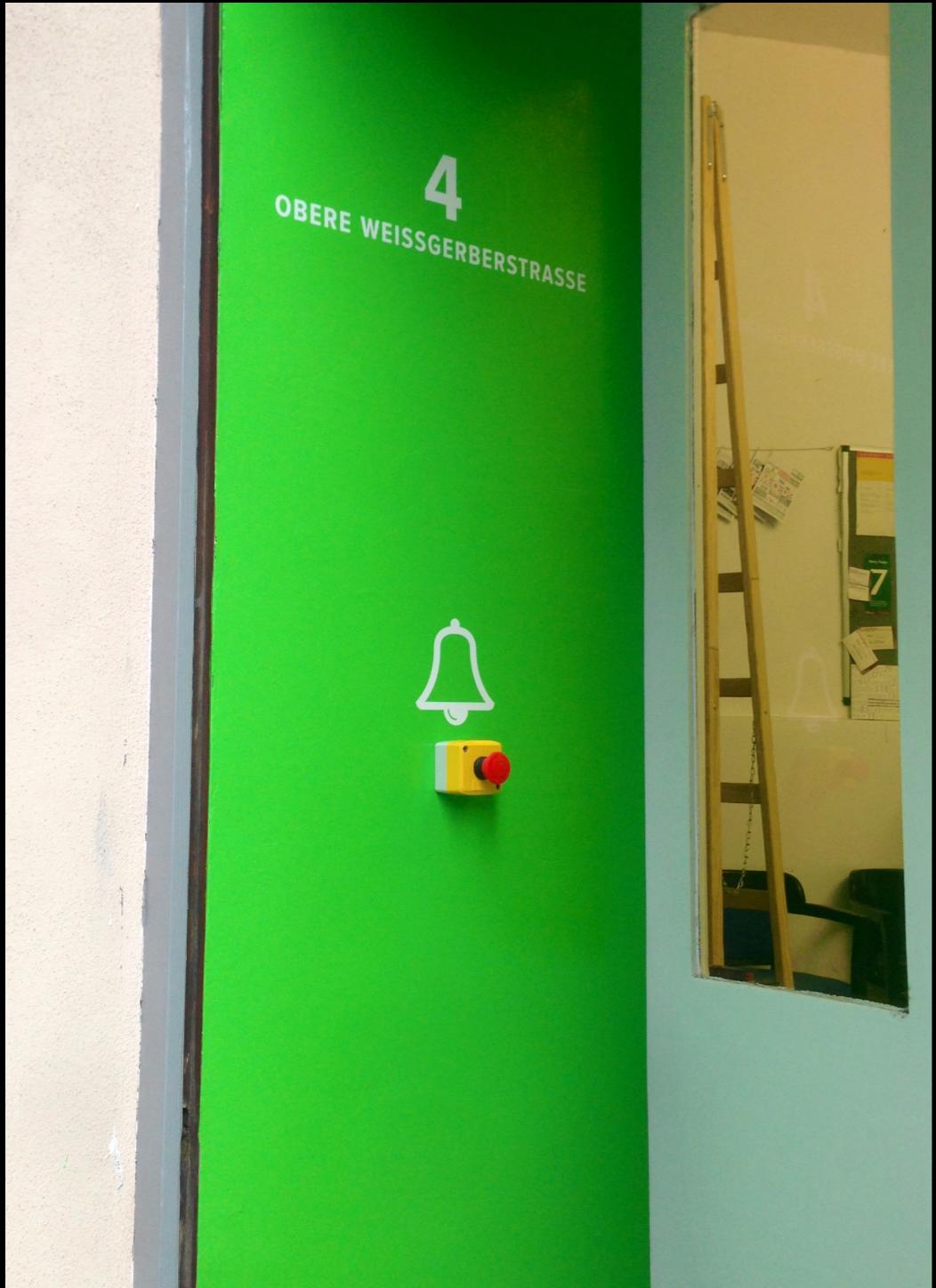
Stefan Daschek / @noniq

My first Ruby-Extension written in C

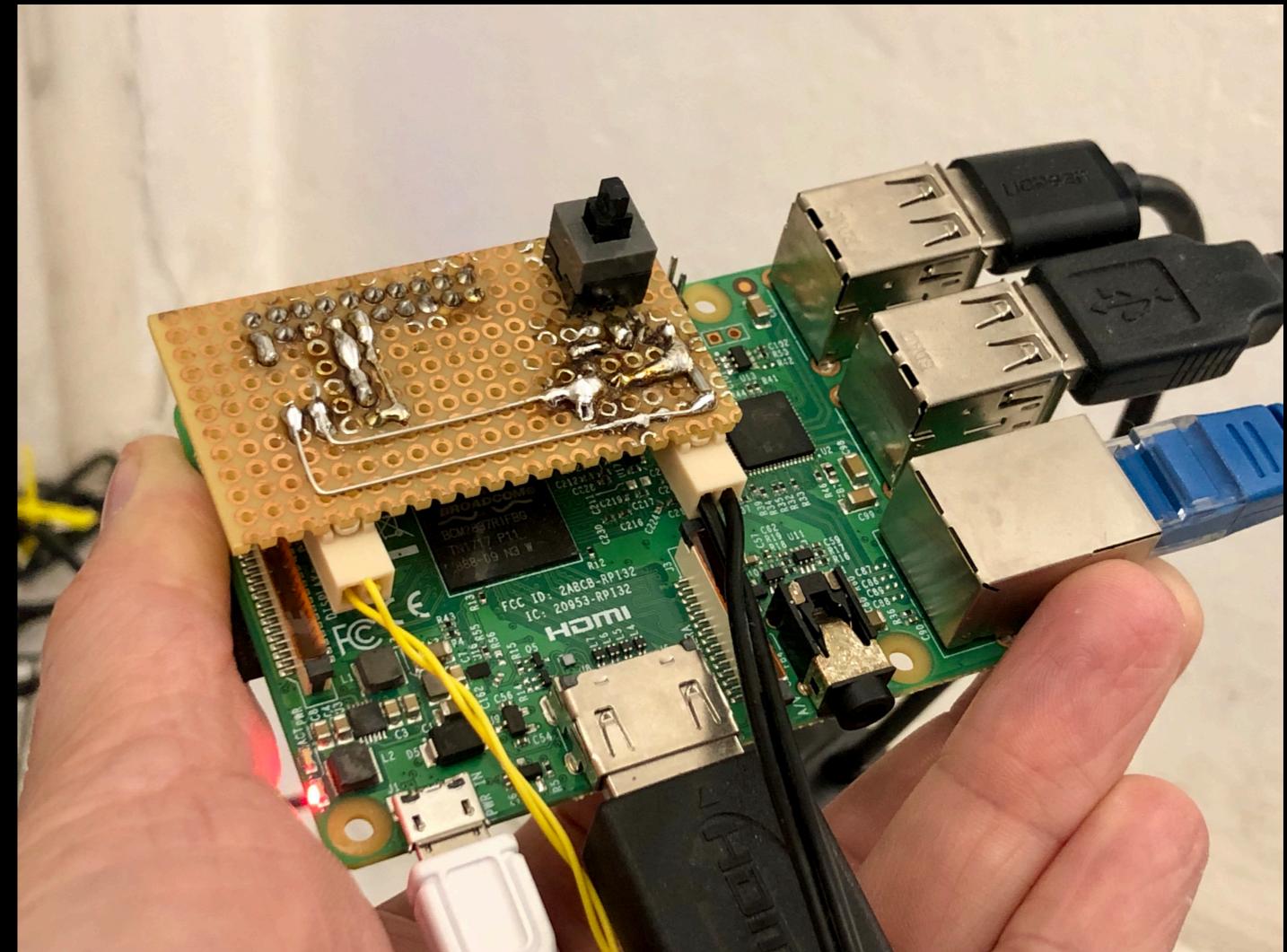
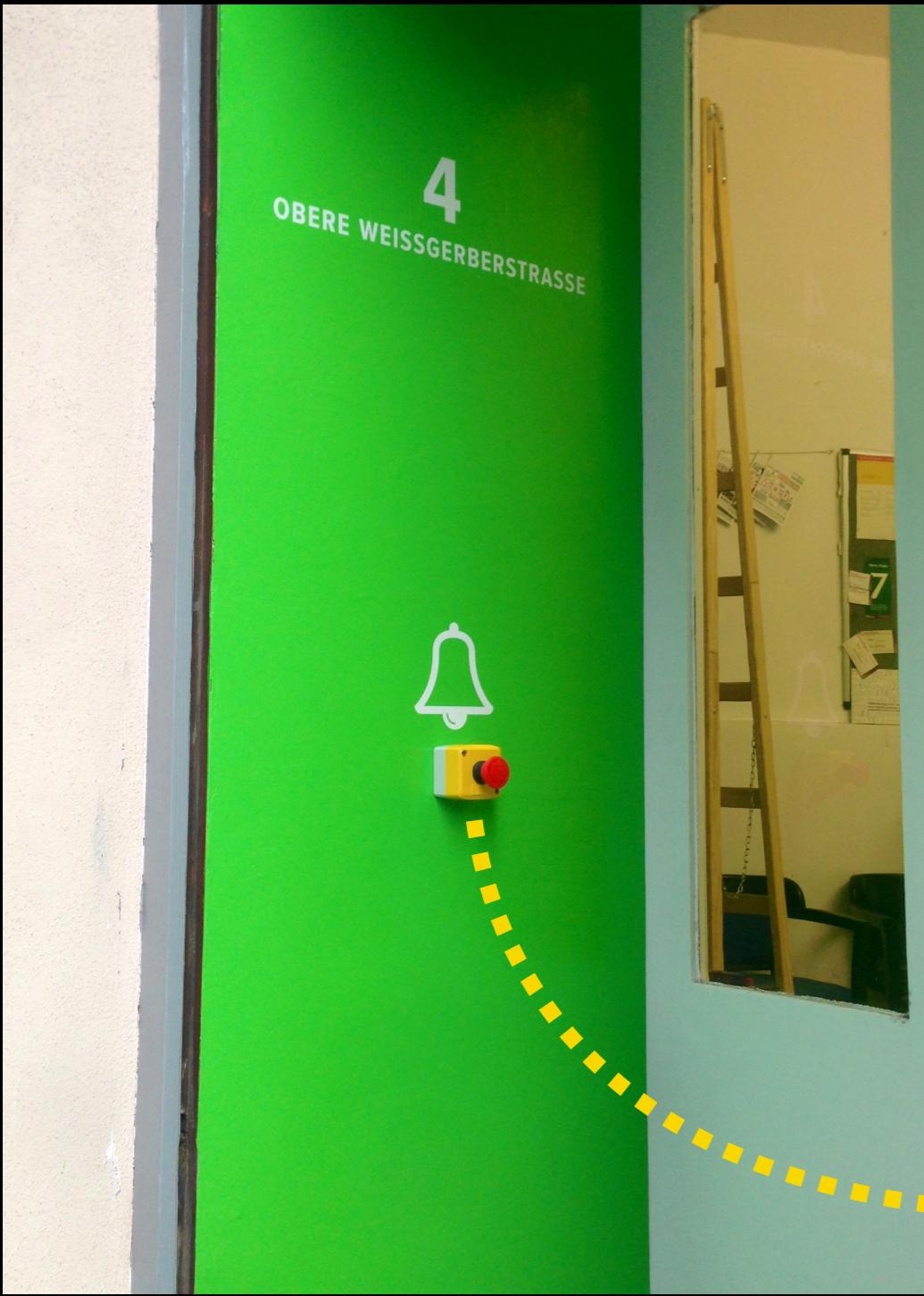
Performance, Performance, Performance!

Why?

Why?



Why?



Why?



Powered by Ruby!

```
require "pi_piper"
include PiPiper

after pin: $config.gpio_pin, goes: :low do
  # button pressed, now do all kinds of stuff
  # (actual code omitted)

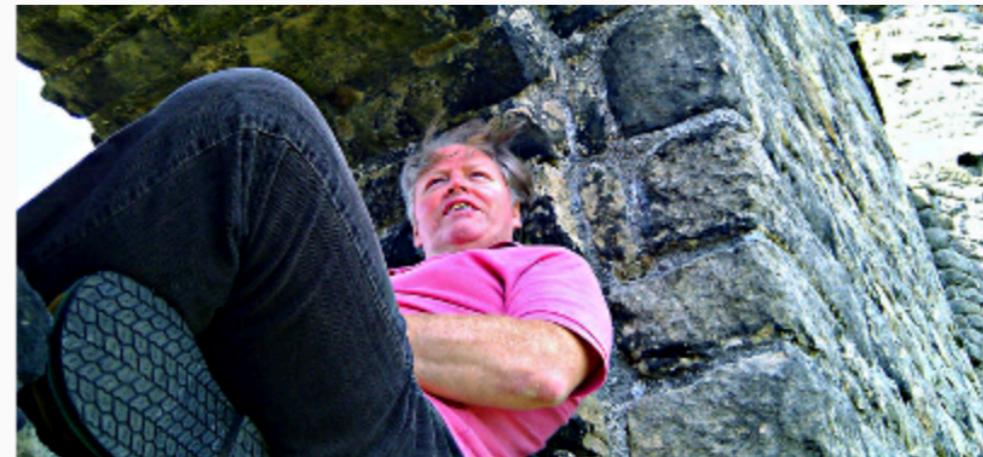
  sleep 0.5
end

PiPiper.wait
```

Under the hood: WiringPi ...

Wiring Pi

GPIO Interface library for the Raspberry Pi



[Home](#) [News](#) [Examples/How-To](#) [Reference](#) [Extensions](#) [Download and Install](#) [Pins](#) [The GPIO utility](#) [Dev Lib](#)

[Contact](#)

[Home](#)

About

WiringPi is a **PIN** based GPIO access library written in C for the BCM2835, BCM2836 and BCM2837 SoC devices used in all **Raspberry Pi** versions. It's released under the [GNU LGPLv3](#) license and is usable from C, C++ and RTB (BASIC) as well as many other languages with suitable wrappers (See below) It's designed to be familiar to people who have used the Arduino "wiring" system¹ and is intended for use by experienced C/C++ programmers. It is not a newbie learning tool.

Recent Posts

- [wiringPi – deprecated...](#)
- [wiringPi updated to 2.52 for the Raspberry Pi 4B](#)
- [wiringPi updated to 2.46 for the new Pi v3+](#)
- [wiringPi updated to 2.36](#)
- [wiringPi update to 2.29](#)

... with PiPiper

☰ README.md

Overview

build passing

Pi Piper brings event driven programming to the Raspberry Pi's GPIO pins. Pi Piper works with all revisions of the Raspberry Pi, and has been tested with Ruby 1.9.3 & 2.0 under both [Raspbian "wheezy"](#) and [Occidentalis v0.2](#).

To get started:

If you do not already have Ruby installed, first you'll need to:

```
sudo apt-get install ruby ruby1.9.1-dev libssl-dev
```

Despite one of the packages being titled "ruby1.9.1-dev", the above command will install Ruby 1.9.3 (as of January 2013) and the Ruby dev tools.

To install Pi Piper:

```
sudo gem install pi_piper
```

GPIO

The GPIO pins (or General Purpose I/O pins) are the primary "do anything" pins on the Raspberry Pi. Reading inputs from them is as simple as:

... with PiPiper

README.md

Overview

build passing

Pi Piper brings event driven programming to the Raspberry Pi's GPIO pins. Pi Piper works with all revisions of the Raspberry Pi, and has been tested with Ruby 1.9.3 & 2.0 under both Raspbian "wheezy" and Occidentalis v0.2.

To get started:

If you do not already have Ruby installed, first you'll need to:

```
sudo apt-get install ruby ruby1.9.1-dev libssl-dev
```

Despite one of the packages being titled "ruby1.9.1-dev", the above command will install Ruby 1.9.3 (as of January 2013) and the Ruby dev tools.

To install Pi Piper:

```
sudo gem install pi_piper
```

GPIO

The GPIO pins (or General Purpose I/O pins) are the primary "do anything" pins on the Raspberry Pi. Reading inputs from them is as simple as:

... with PiPiper

☰ README.md

Overview

build passing

Pi Piper brings event driven programming to the Raspberry Pi's GPIO pins. Pi Piper works with all revisions of the Raspberry Pi, and has been tested with Ruby 1.9.3 & 2.0 under both [Raspbian "wheezy"](#) and [Occidentalis v0.2](#).

To get started:



If you do not already have Ruby installed, first you'll need to:

```
sudo apt-get install ruby ruby1.9.1-dev libssl-dev
```

Despite one of the packages being titled "ruby1.9.1-dev", the above command will install Ruby 1.9.3 (as of January 2013) and the Ruby dev tools.

To install Pi Piper:

```
sudo gem install pi_piper
```

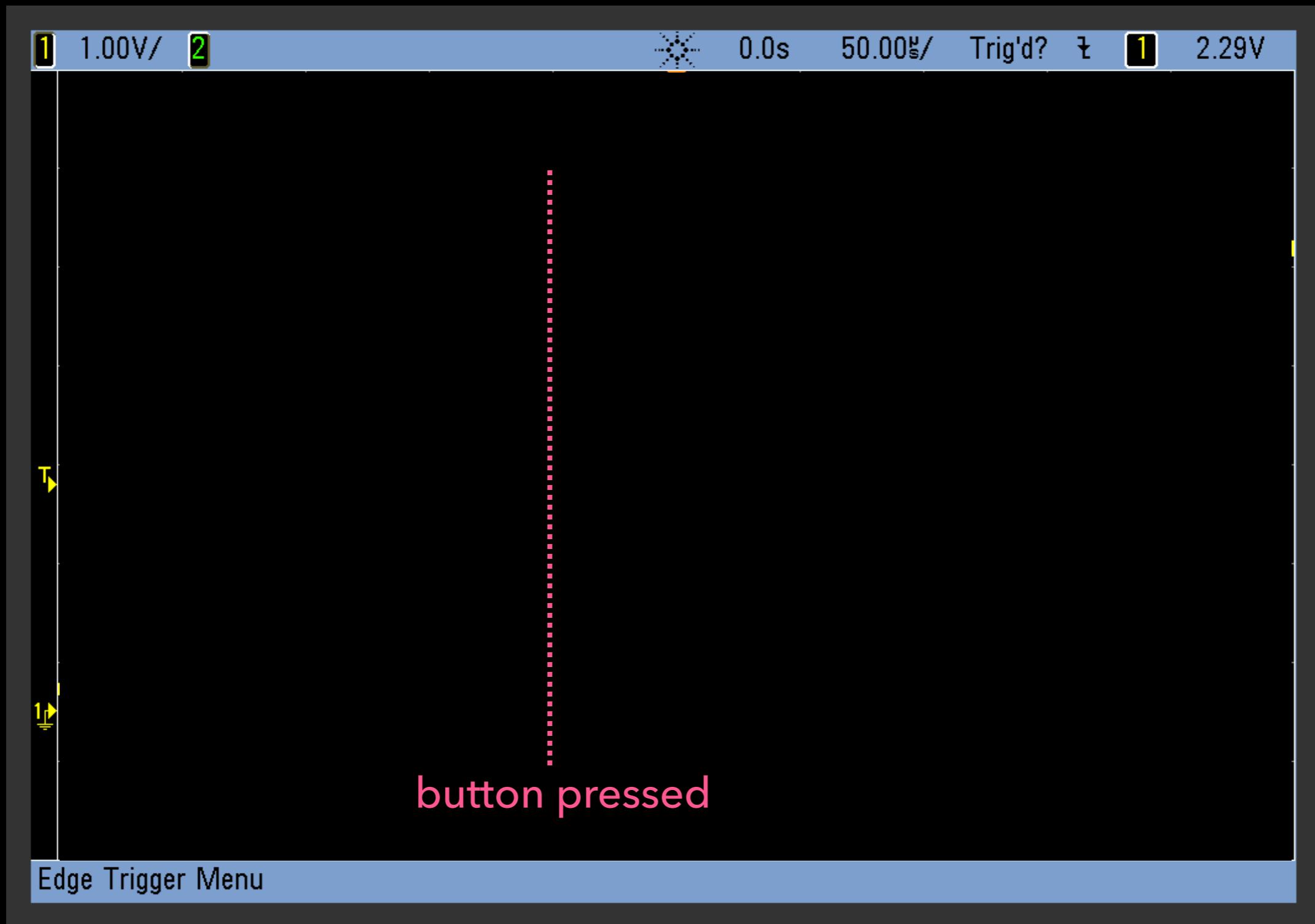
... also:



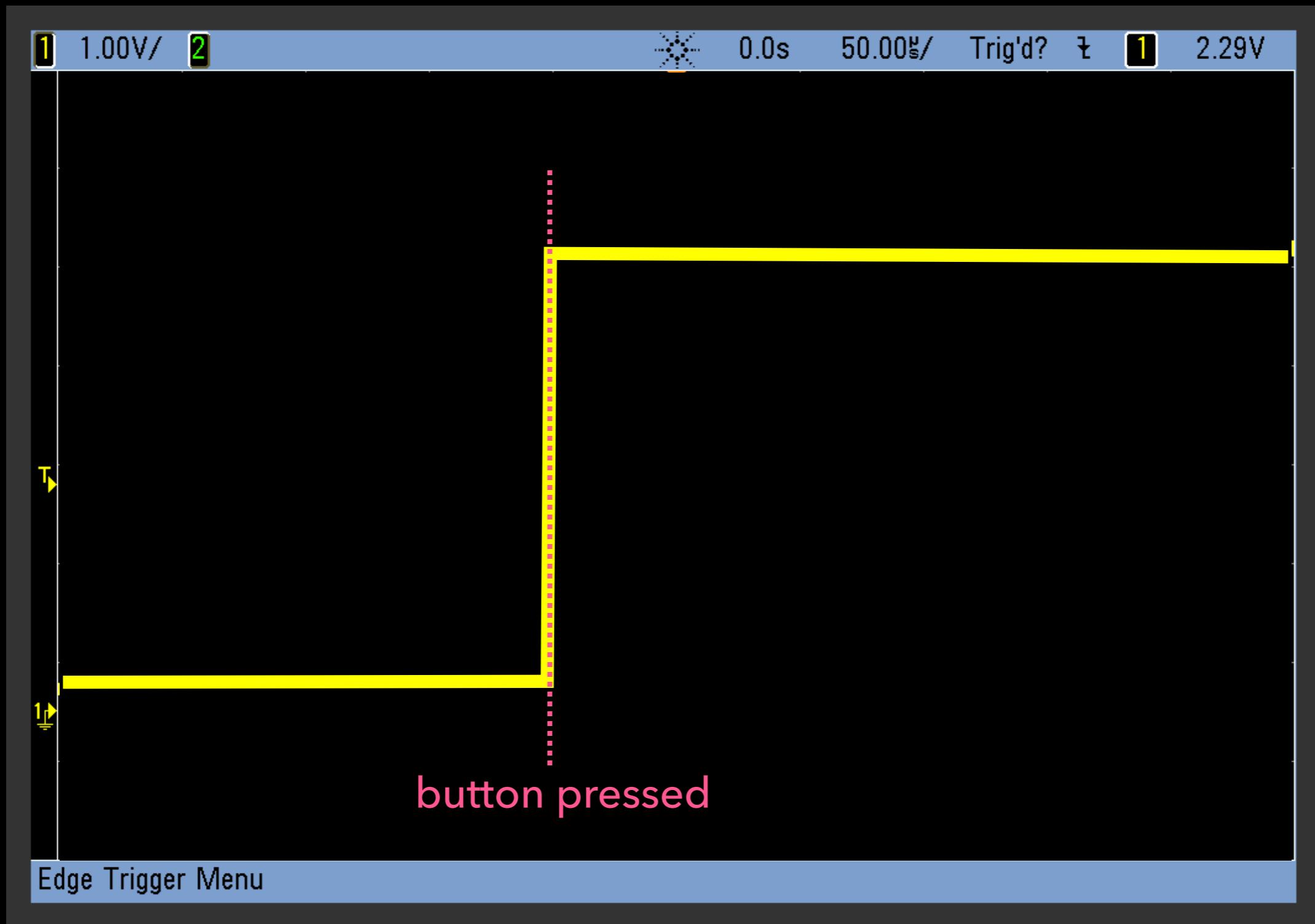
GPIO

The GPIO pins (or General Purpose I/O pins) are the primary "do anything" pins on the Raspberry Pi. Reading inputs from them is as simple as:

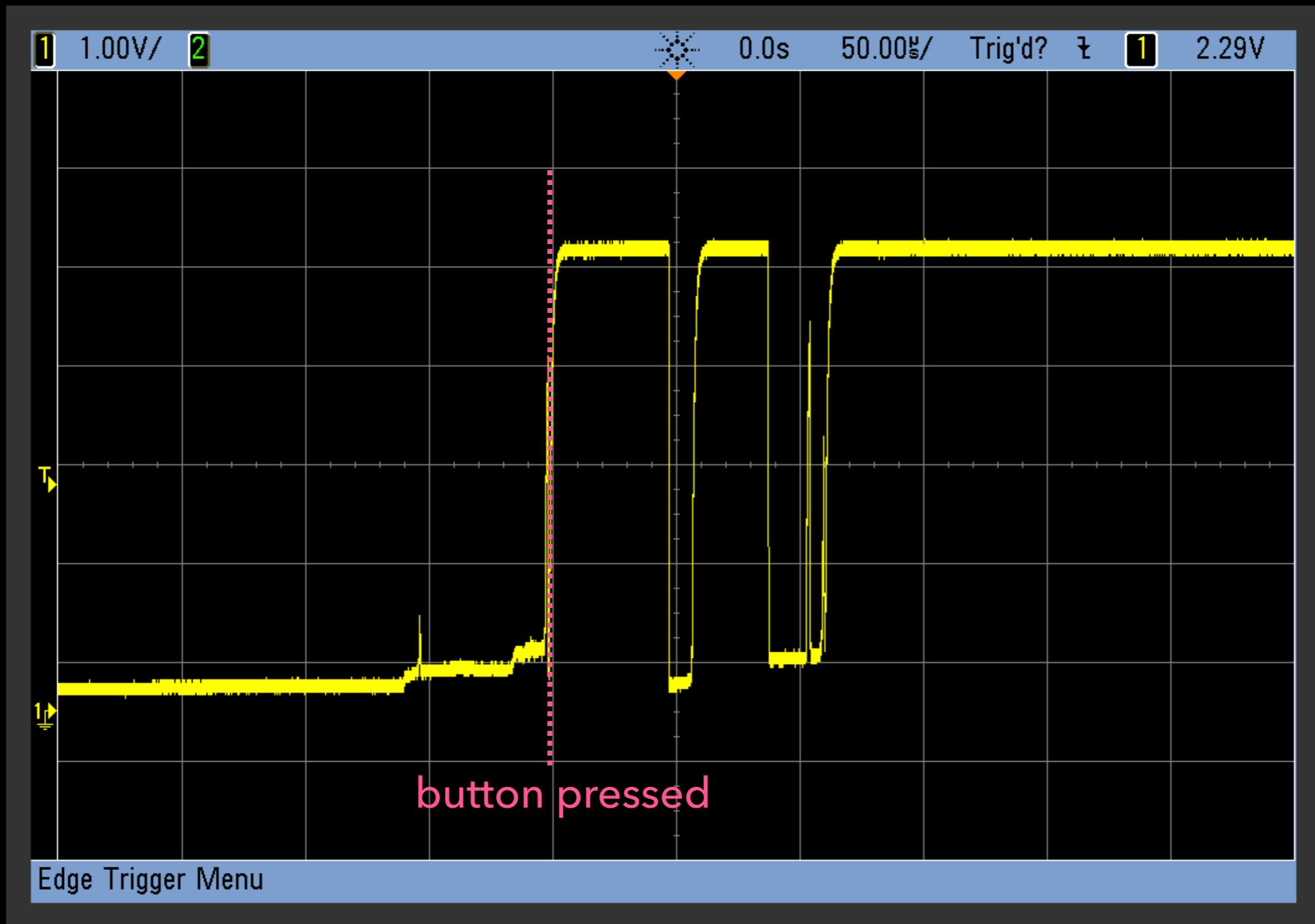
Why is speed relevant?



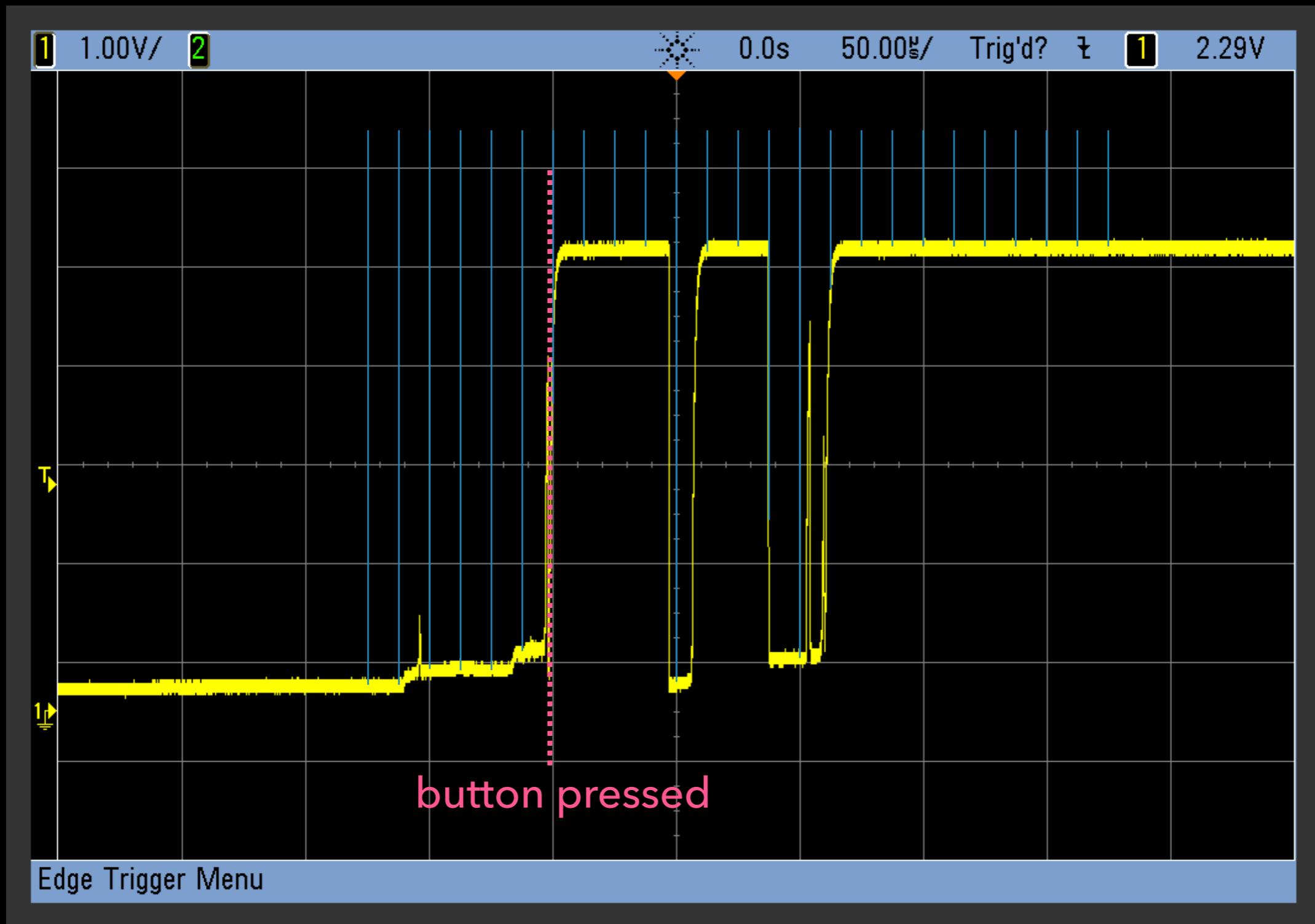
Why is speed relevant?



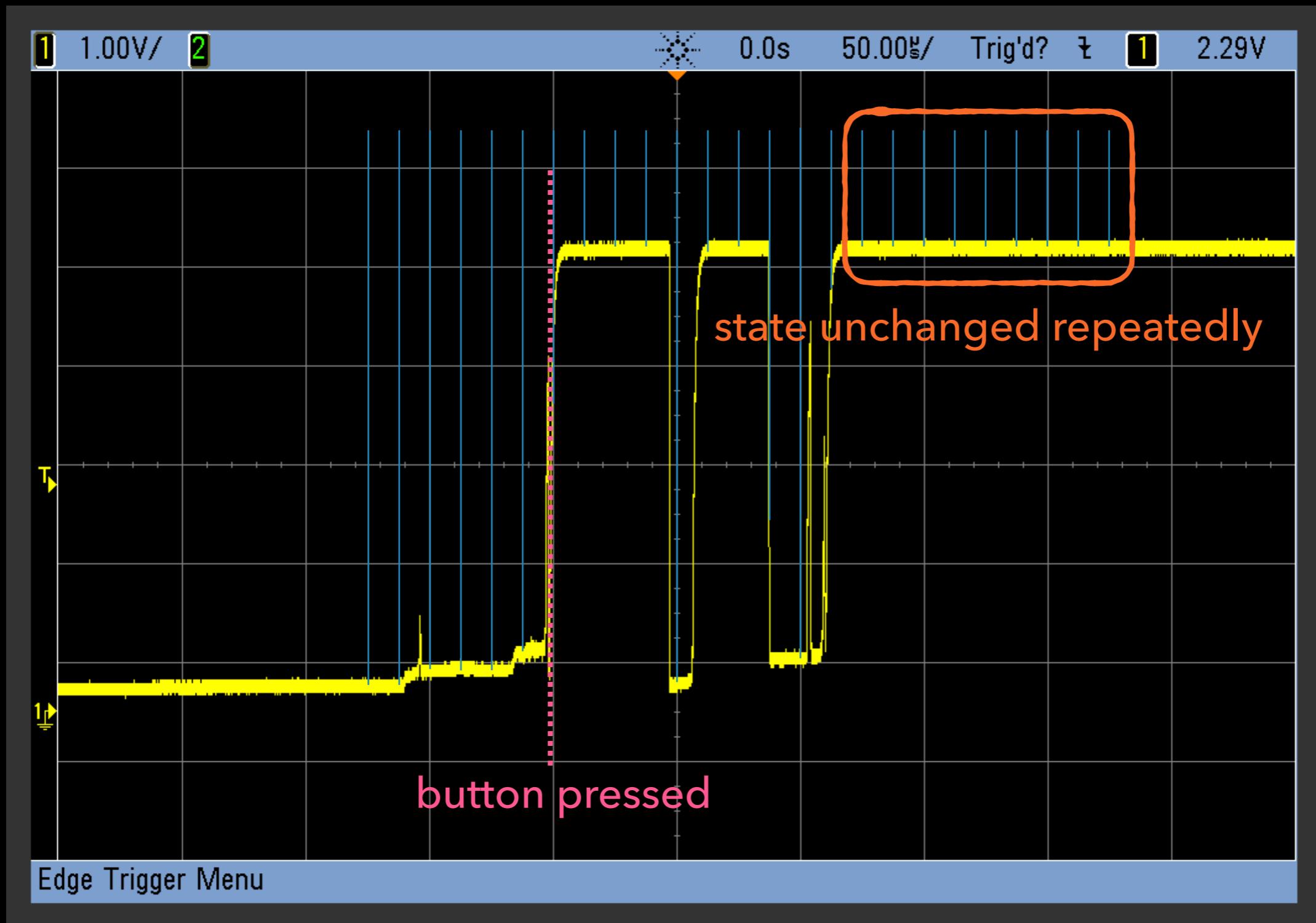
Why is speed relevant?



Why is speed relevant?



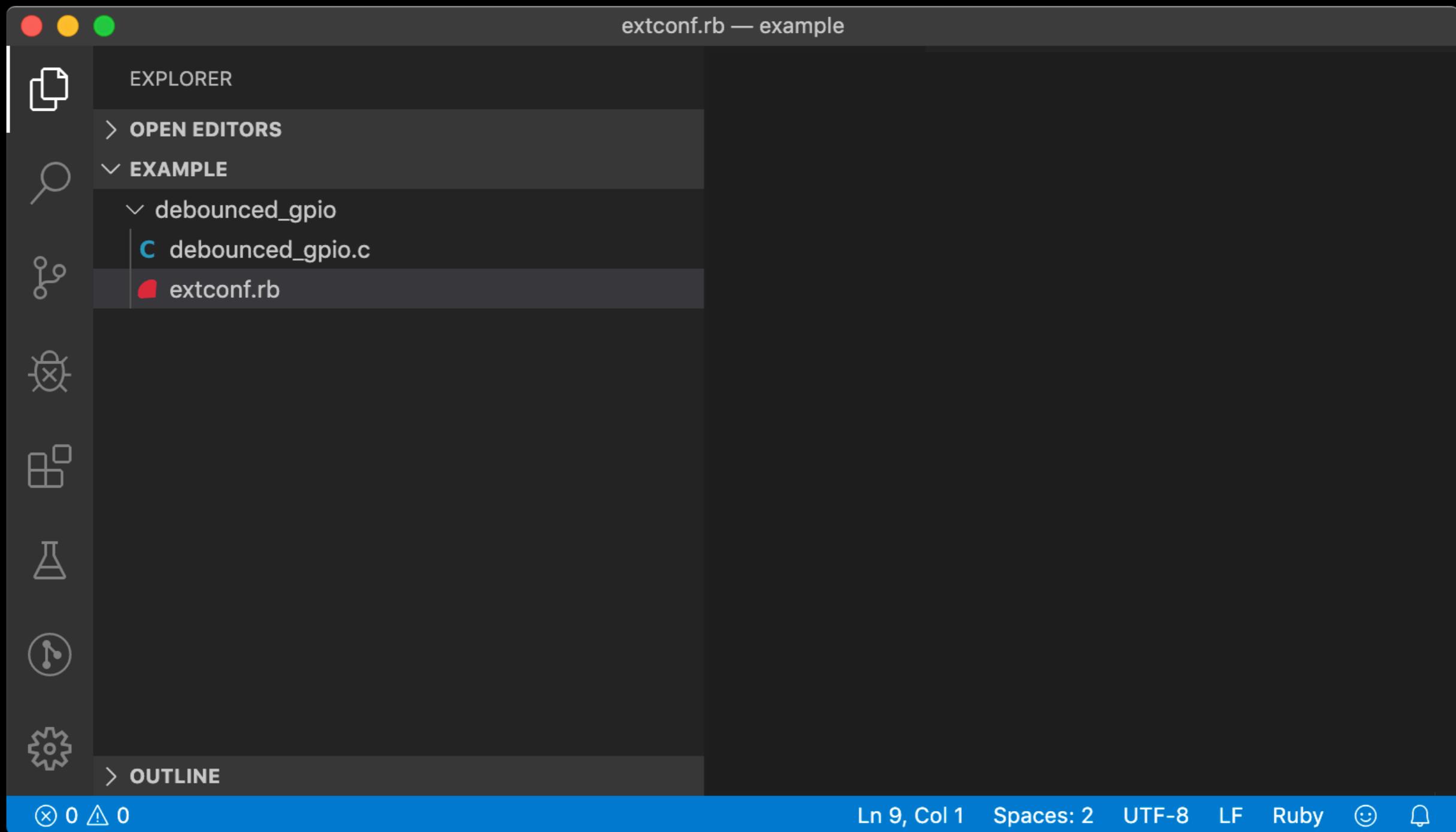
Why is speed relevant?



Back on track:

Let's make a C extension!

Let's make a C extension!



The screenshot shows a dark-themed code editor interface. The title bar reads "extconf.rb — example". The left sidebar contains icons for Explorer, Open Editors, Example, Find, Go To, Outline, and Settings. The "EXPLORER" section shows a project structure under "EXAMPLE": "debounced_gpio" contains "debounced_gpio.c" (marked with a blue C icon) and "extconf.rb" (marked with a red Ruby icon). The bottom status bar shows file statistics: "Ln 9, Col 1" and "Spaces: 2", encoding "UTF-8", line ending "LF", language "Ruby", and icons for a smiley face and a bell.

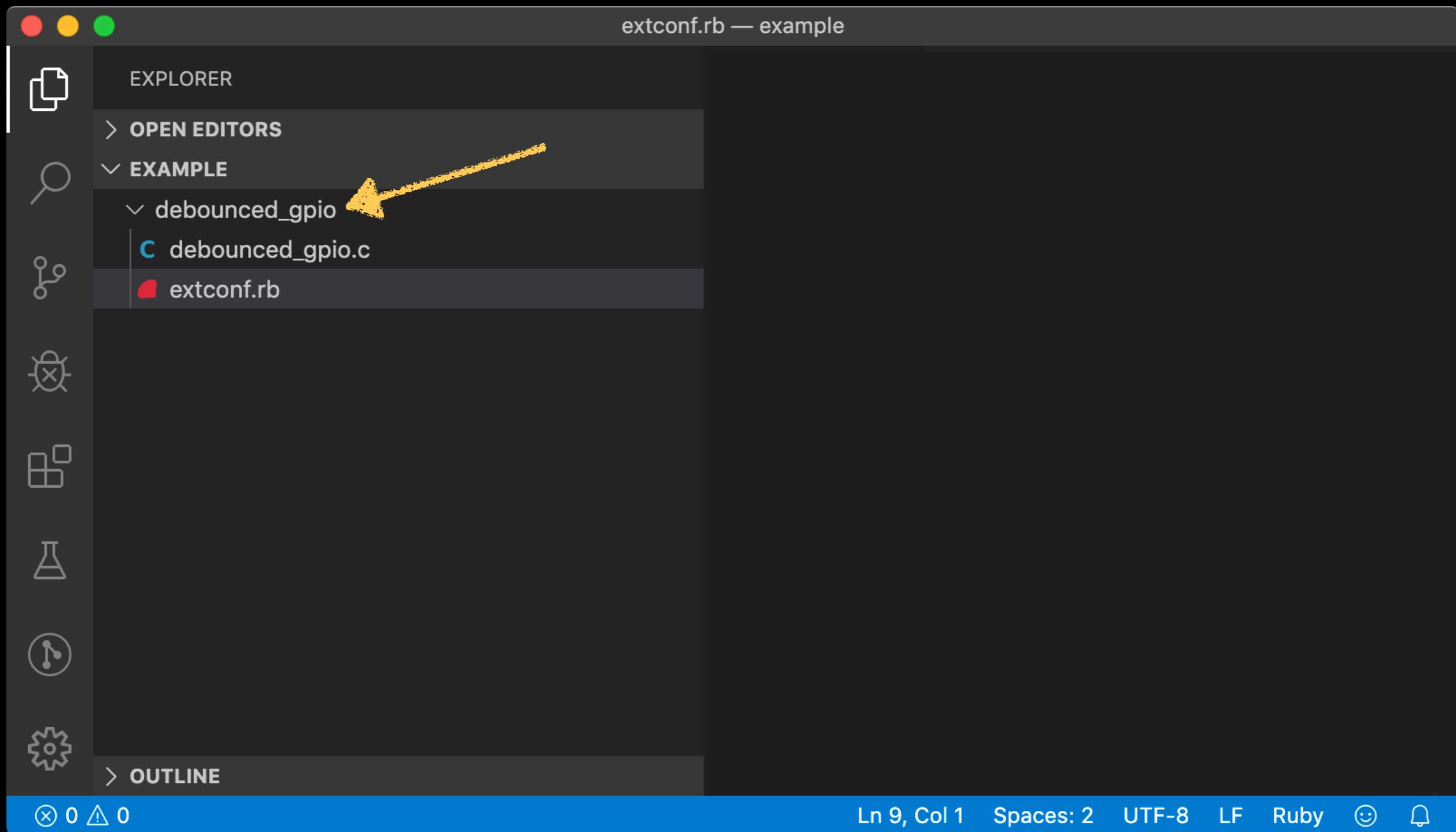
```
extconf.rb — example

EXPLORER
> OPEN EDITORS
EXAMPLE
debounced_gpio
  debounced_gpio.c
  extconf.rb

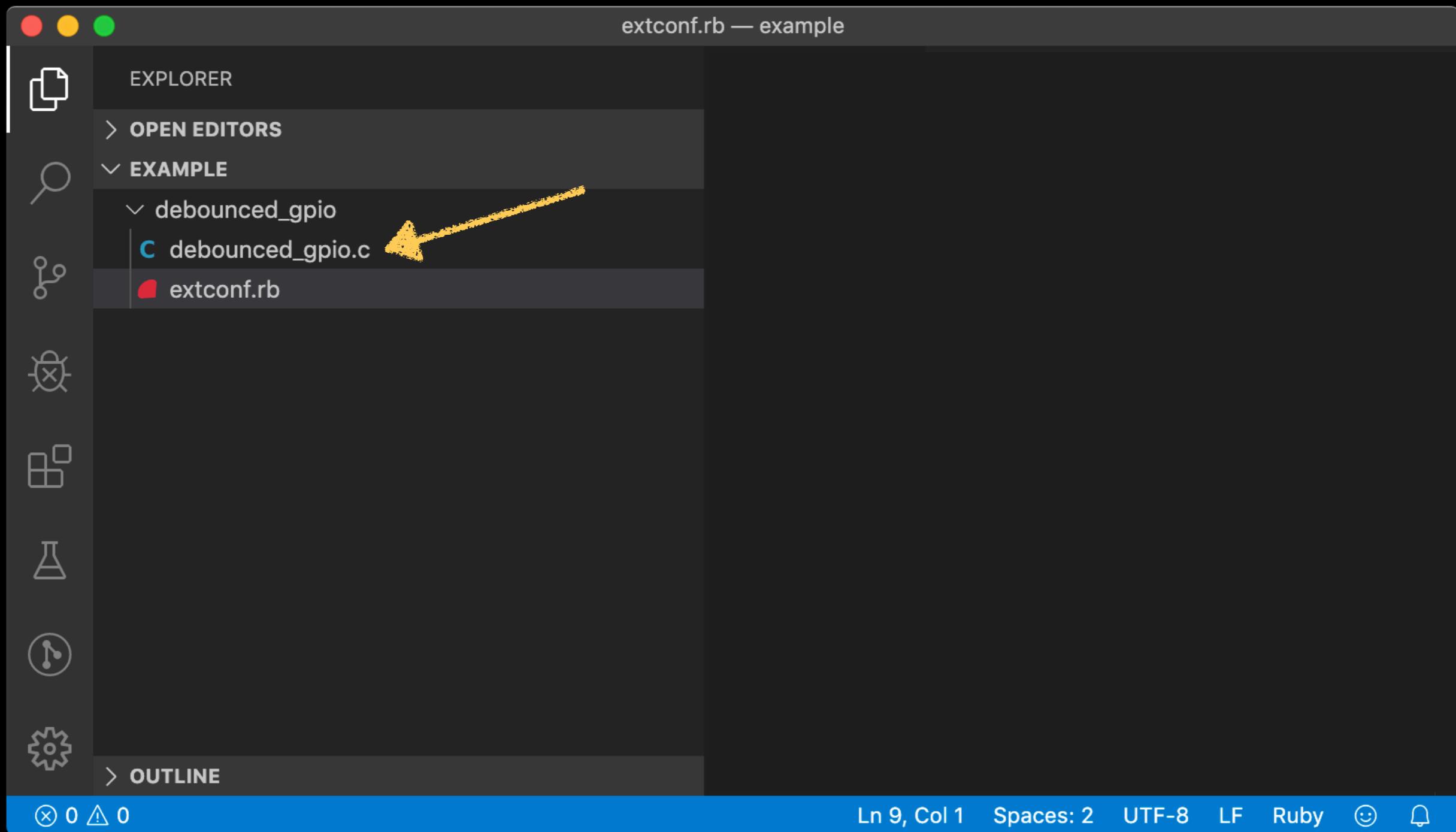
> OUTLINE

Ln 9, Col 1  Spaces: 2  UTF-8  LF  Ruby  ☺  📡
```

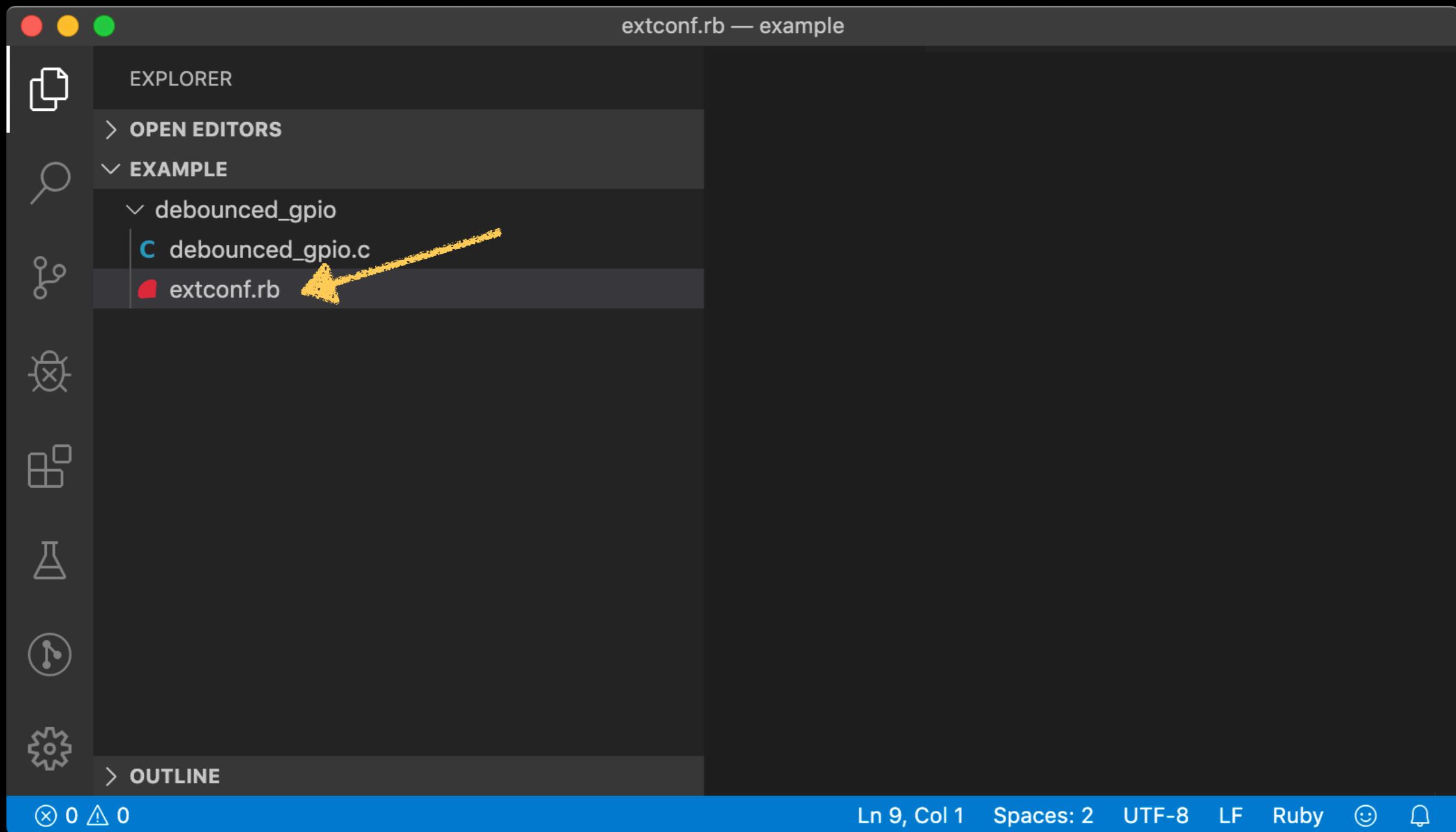
Let's make a C extension!



Let's make a C extension!



Let's make a C extension!



Let's make a C extension!

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'EXAMPLE'. It includes a folder 'debounced_gpio' containing a C source file 'debounced_gpio.c' and an 'extconf.rb' configuration file. A yellow arrow points from the text 'debounced_gpio' in the title bar towards the 'extconf.rb' file in the Explorer. The main editor area displays the 'extconf.rb' file with the following content:

```
extconf.rb — example
extconf.rb ×
debounced_gpio > extconf.rb
1 require "mkmf"
2
3
4
5
6
7 create_header
8 create_makefile "debounced_gpio"
9
```

The status bar at the bottom shows the file path 'Ln 9, Col 1', encoding 'UTF-8', line endings 'LF', and the language 'Ruby'. There are also icons for smiley faces and notifications.

Let's make a C extension!

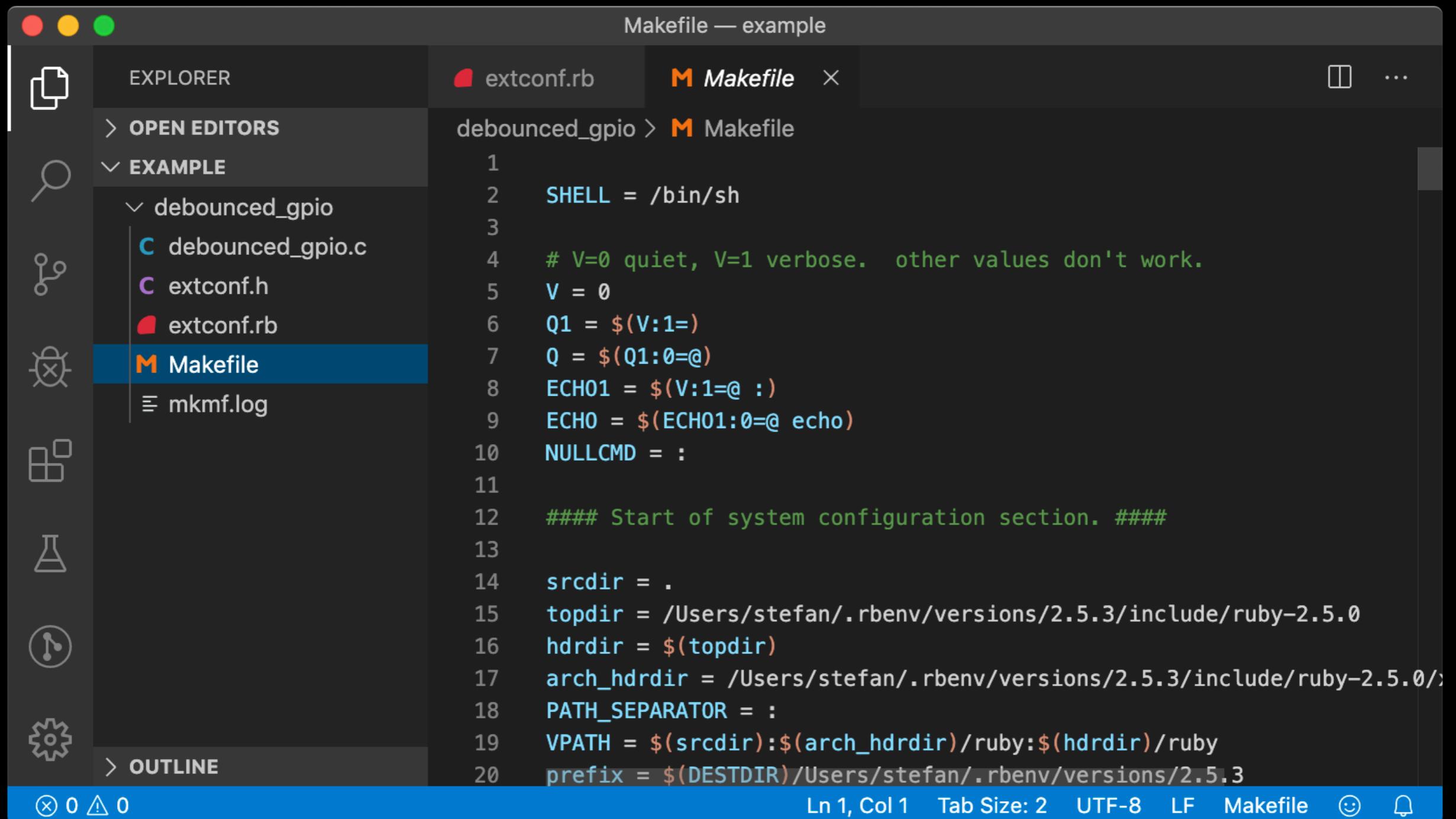
The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists a project structure under 'EXAMPLE'. The 'debounced_gpio' folder contains two files: 'debounced_gpio.c' (highlighted with a yellow arrow) and 'extconf.rb'. The main editor area displays the 'extconf.rb' file with the following content:

```
extconf.rb — example
extconf.rb ×
debounced_gpio > extconf.rb
1 require "mkmf"
2
3 unless have_library "wiringPi"
4   abort "Please install wiringPi"
5 end
6
7 create_header
8 create_makefile "debounced_gpio"
9
```

The status bar at the bottom shows the file is at 'Ln 9, Col 1' with 'Spaces: 2' and encoding 'UTF-8'. It also indicates the file is saved ('LF') and the language is 'Ruby'. There are icons for a smiley face and a bell通知.

```
~$ ruby extconf.rb
```

~\$ ruby extconf.rb



Makefile — example

EXPLORER extconf.rb Makefile debounced_gpio > Makefile

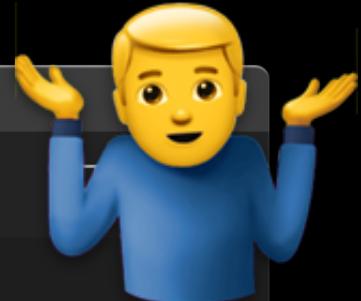
```
1 SHELL = /bin/sh
2 # V=0 quiet, V=1 verbose. other values don't work.
3 V = 0
4 Q1 = $(V:1=@)
5 Q = $(Q1:0=@)
6 ECHO1 = $(V:1=@ :)
7 ECHO = $(ECHO1:0=@ echo)
8 NULLCMD =
9
10 #### Start of system configuration section. ####
11
12 srcdir =
13 topdir = /Users/stefan/.rbenv/versions/2.5.3/include/ruby-2.5.0
14 hdrdir = $(topdir)
15 arch_hdrdir = /Users/stefan/.rbenv/versions/2.5.3/include/ruby-2.5.0/
16 PATH_SEPARATOR =
17 VPATH = $(srcdir):$(arch_hdrdir)/ruby:$(hdrdir)/ruby
18 prefix = $(DESTDIR)/Users/stefan/.rbenv/versions/2.5.3
```

OPEN EDITORS
EXAMPLE
debounced_gpio
debounced_gpio.c
extconf.h
extconf.rb
Makefile
mkmf.log

OUTLINE

Ln 1, Col 1 Tab Size: 2 UTF-8 LF Makefile ☺ 🔔

~\$ ruby extconf.rb



Makefile — example

EXPLORER

> OPEN EDITORS

EXAMPLE

debounced_gpio > Makefile

debounced_gpio.c
extconf.h
extconf.rb
Makefile
mkmf.log

```
1 SHELL = /bin/sh
2
3 # V=0 quiet, V=1 verbose. other values don't work.
4 V = 0
5 Q1 = $(V:1=)
6 Q = $(Q1:0=@)
7 ECHO1 = $(V:1=@ :)
8 ECHO = $(ECHO1:0=@ echo)
9
10 NULLCMD =
11
12 ##### Start of system configuration section. #####
13
14 srcdir =
15 topdir = /Users/stefan/.rbenv/versions/2.5.3/include/ruby-2.5.0
16 hdrdir = $(topdir)
17 arch_hdrdir = /Users/stefan/.rbenv/versions/2.5.3/include/ruby-2.5.0/
18 PATH_SEPARATOR =
19 VPATH = $(srcdir):$(arch_hdrdir)/ruby:$(hdrdir)/ruby
20 prefix = $(DESTDIR)/Users/stefan/.rbenv/versions/2.5.3
```

Ln 1, Col 1 Tab Size: 2 UTF-8 LF Makefile ☺ 🔔

Minimum Viable C Extension

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    // TODO
}

void Init_debounced_gpio() {
    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(
        module,
        "wait_for_state_change",
        rb_wait_for_state_change,
        1
    );
}
```

Minimum Viable C Extension

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    // TODO
}

void Init_debounced_gpio() {
    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(
        module,
        "wait_for_state_change",
        rb_wait_for_state_change,
        1
    );
}
```

debounced_gpio \$

Minimum Viable C Extension

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    // TODO
}

void Init_debounced_gpio() {
    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(
        module,
        "wait_for_state_change",
        rb_wait_for_state_change,
        1
    );
}
```

debounced_gpio \$ make
compiling debounced_gpio.c
linking shared-object debounced_gpio.bundle

debounced_gpio \$

Minimum Viable C Extension

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    // TODO
}

void Init_debounced_gpio() {
    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(
        module,
        "wait_for_state_change",
        rb_wait_for_state_change,
        1
    );
    require_relative "debounced_gpio/debounced_gpio"
    DebouncedGPIO.wait_for_state_change(42)
    puts "Success!"
}
```

Minimum Viable C Extension

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    // TODO
}

void Init_debounced_gpio() {
    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(
        module,
        "wait_for_state_change",
        rb_wait_for_state_change,
        1
    );
}

require_relative "debounced_gpio/debounced_gpio"
DebouncedGPIO.wait_for_state_change(42)
puts "Success!"
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    int pin = NUM2INT(rb_pin);  
  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    int pin = NUM2INT(rb_pin);  
    if (pin == 42) {  
  
    }  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    int pin = NUM2INT(rb_pin);  
    if (pin == 42) {  
        return INT2NUM(pin * 2);  
    }  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    int pin = NUM2INT(rb_pin);  
    if (pin == 42) {  
        return INT2NUM(pin * 2);  
    }  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    int pin = NUM2INT(rb_pin);  
    if (pin == 42) {  
        return INT2NUM(pin * 2);  
    }  
    return Qnil;  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    int pin = NUM2INT(rb_pin);  
    if (pin == 42) {  
        return INT2NUM(pin * 2);  
    }  
    return Qnil;  
}
```

Let's return something:

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    // TODO  
    return INT2NUM(1337);  
}
```

Looking for an argument?

```
VALUE rb_wait_for_state_change(VALUE self, VALUE rb_pin) {  
    int pin = NUM2INT(rb_pin);  
    if (pin == 42) {  
        return INT2NUM(pin * 2);  
    }  
    return Qnil;  
}
```

Documentation: github.com/ruby/ruby/blob/master/doc/extension.rdoc

Documentation: github.com/ruby/ruby/blob/master/doc/extension.rdoc

Data Types

The Ruby interpreter has the following data types:

T_NIL	nil
T_OBJECT	ordinary object
T_CLASS	class
T_MODULE	module
T_FLOAT	floating point number
T_STRING	string
T_REGEXP	regular expression
T_ARRAY	array
T_HASH	associative array
T_STRUCT	(Ruby) structure
T_BIGNUM	multi precision integer
T_FIXNUM	Fixnum(31bit or 63bit integer)
T_COMPLEX	complex number
T_RATIONAL	rational number
T_FILE	IO
T_TRUE	true

Documentation: github.com/ruby/ruby/blob/master/doc/extension.rdoc

Data Types

The Ruby interpreter

		Manipulating Ruby Data	
T_NIL	nil	As I already mentioned, it is not recommended to modify an object's internal structure. To manipulate objects, use the functions supplied by the Ruby interpreter. Some (not all) of the useful functions are listed below:	
T_OBJECT	ord		
T_CLASS	class	String Functions	
T_MODULE	mod	<code>rb_str_new(const char *ptr, long len)</code>	Creates a new Ruby string.
T_FLOAT	float	<code>rb_str_new2(const char *ptr)</code>	
T_STRING	string	<code>rb_str_new_cstr(const char *ptr)</code>	Creates a new Ruby string from a C string. This is equivalent to <code>rb_str_new(ptr, strlen(ptr))</code> .
T_REGEXP	regexp	<code>rb_str_new_literal(const char *ptr)</code>	Creates a new Ruby string from a C string literal.
T_ARRAY	array	<code>rb_sprintf(const char *format, ...)</code>	
T_HASH	assoc		Creates a new Ruby string with printf(3) format.
T_STRUCT	(Ruby)	<code>rb_vsprintf(const char *format, va_list ap)</code>	Note: In the format string, "%<PRI>VALUE can be used for Object#to_s (or Object#inspect if '+' flag is set) output (and related argument must be a VALUE). Since it conflicts with "%i", for integers in format strings, use "%d".
T_BIGNUM	mu		
T_FIXNUM	Fixnum	<code>rb_str_append(VALUE str1, VALUE str2)</code>	Appends Ruby string str2 to Ruby string str1.
T_COMPLEX	complex	<code>rb_str_cat(VALUE str, const char *ptr, long len)</code>	Appends len bytes of data from ptr to the Ruby string.
T_RATIONAL	rational		
T_FILE	IO		
T_TRUE	true		

Documentation: github.com/ruby/ruby/blob/master/doc/extension.rdoc

Data Types

The Ruby interpreter

		Manipulating Ruby Data
T_NIL	nil	As I already mentioned, it is not recommended to modify an object's internal structure. To manipulate objects, use the functions supplied by the Ruby interpreter. Some (not all) of the useful functions are listed below:
T_OBJECT	ord	
T_CLASS	class	String Functions
T_MODULE	mod	Use Ruby Features from C
T_FLOAT	float	There are several ways to invoke Ruby's features from C code.
T_STRING	string	Evaluate Ruby Programs in a String
T_REGEXP	regexp	The easiest way to use Ruby's functionality from a C program is to evaluate the string as Ruby program. This function will do the job:
T_ARRAY	array	<pre>VALUE rb_eval_string(const char *str)</pre>
T_HASH	ass	Evaluation is done under the current context, thus current local variables of the innermost method (which is defined by Ruby) can be accessed.
T_STRUCT	(Ru	Note that the evaluation can raise an exception. There is a safer function:
T_BIGNUM	mu	<pre>VALUE rb_eval_string_protect(const char *str, int *state)</pre>
T_FIXNUM	Fix	It returns nil when an error occurred. Moreover, *state is zero if str was successfully evaluated, or nonzero otherwise.
T_COMPLEX	cor	<pre>rb_str_append(VALUE str1, VALUE str2)</pre>
T_RATIONAL	ratio	Appends Ruby string str2 to Ruby string str1.
T_FILE	IO	<pre>rb_str_cat(VALUE str, const char *ptr, long len)</pre>
T_TRUE	true	Appends len bytes of data from ptr to the Ruby string.

Odds and ends: Pardon the interruption!

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    while(1);
}

void Init_debounced_gpio() {

    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(module, "wait_for_state_change",
    rb_wait_for_state_change, 1);
}
```

Odds and ends: Pardon the interruption!

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    while(1);
}

void Init_debounced_gpio() {
    signal(SIGINT, handle_signal);
    signal(SIGTERM, handle_signal);

    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(module, "wait_for_state_change",
    rb_wait_for_state_change, 1);
}
```

Odds and ends: Pardon the interruption!

```
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    while(1);
}

void handle_signal(int sig) {
    if (sig == SIGINT) {
        rb_interrupt();
    } else {
        ruby_default_signal(sig);
    }
}

void Init_debounced_gpio() {
    signal(SIGINT, handle_signal);
    signal(SIGTERM, handle_signal);

    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(module, "wait_for_state_change",
    rb_wait_for_state_change, 1);
}
```

Odds and ends: Pardon the interruption!

```
#include <signal.h>
#include "ruby.h"
#include "extconf.h"

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    while(1);
}

void handle_signal(int sig) {
    if (sig == SIGINT) {
        rb_interrupt();
    } else {
        ruby_default_signal(sig);
    }
}

void Init_debounced_gpio() {
    signal(SIGINT, handle_signal);
    signal(SIGTERM, handle_signal);

    VALUE module = rb_define_module("DebouncedGPIO");
    rb_define_singleton_method(module, "wait_for_state_change",
    rb_wait_for_state_change, 1);
}
```

Odds and ends: Do not block other threads!

```
struct no_gvl_args {
    int pin;
    int newState;
};

VALUE rb_wait_for_state_change(VALUE self, VALUE pin) {
    struct no_gvl_args args;
    args.pin = NUM2INT(pin);
    rb_thread_call_without_gvl(waitForStateChange, &args, NULL, NULL);
    return INT2NUM(args.newState);
}
```

Odds and ends: Do not block other threads!

```
struct no_gvl_args {  
    int pin;  
    int newState;  
};  
  
VALUE rb_wait_for_state_change(VALUE self)  
{  
    struct no_gvl_args args;  
    args.pin = NUM2INT(pin);  
    rb_thread_call_without_gvl(rb_wait_for_state_change,(void*)&args,1);  
    return INT2NUM(args.newState);  
}
```

ruby/thread.c

```
1495  /*  
1496   * rb_thread_call_without_gvl - permit concurrent/parallel execution.  
1497   * rb_thread_call_without_gvl2 - permit concurrent/parallel execution  
1498   *                                     without interrupt process.  
1499   */  
1500  * rb_thread_call_without_gvl() does:  
1501  *   (1) Check interrupts.  
1502  *   (2) release GVL.  
1503  *       Other Ruby threads may run in parallel.  
1504  *   (3) call func with data1  
1505  *   (4) acquire GVL.  
1506  *       Other Ruby threads can not run in parallel any more.  
1507  *   (5) Check interrupts.  
1508  *  
1509  * rb_thread_call_without_gvl2() does:  
1510  *   (1) Check interrupt and return if interrupted.  
1511  *   (2) release GVL.  
1512  *   (3) call func with data1 and a pointer to the flags.  
1513  *   (4) acquire GVL.  
1514  *  
1515  * If another thread interrupts this thread (Thread#kill, signal delivery,  
1516  * VM-shutdown request, and so on), `ubf()` is called (`ubf()` means  
1517  * "un-blocking function"). `ubf()` should interrupt `func()` execution by  
1518  * toggling a cancellation flag, canceling the invocation of a call inside  
1519  * `func()` or similar. Note that `ubf()` may not be called with the GVL.
```

Finally – this is what the Ruby code now looks like:

```
require_relative "debounced_gpio/debounced_gpio"

loop do
  value = DebouncedGPIO.wait_for_state_change($config gpio_pin)
  if value == 1
    # button pressed
    # ...
  else
    # button released
    # ...
  end
end
```

Thanks! Questions?

Stefan Daschek / @noniq

Actual debouncing routine (in C)

```
int getDebouncedState(int pin) {  
    int lastValue = digitalRead(pin);  
    int debounceCount = 0;  
    while (1) {  
        int value = digitalRead(pin);  
        if (value == lastValue) {  
            debounceCount++;  
        } else {  
            debounceCount = 0;  
        }  
        if (debounceCount == DEBOUNCE_CYCLES) {  
            return value;  
        }  
        lastValue = value;  
        delay(DELAY);  
    }  
}
```