Stefan Daschek / @noniq

My first Ruby-Extension written in C

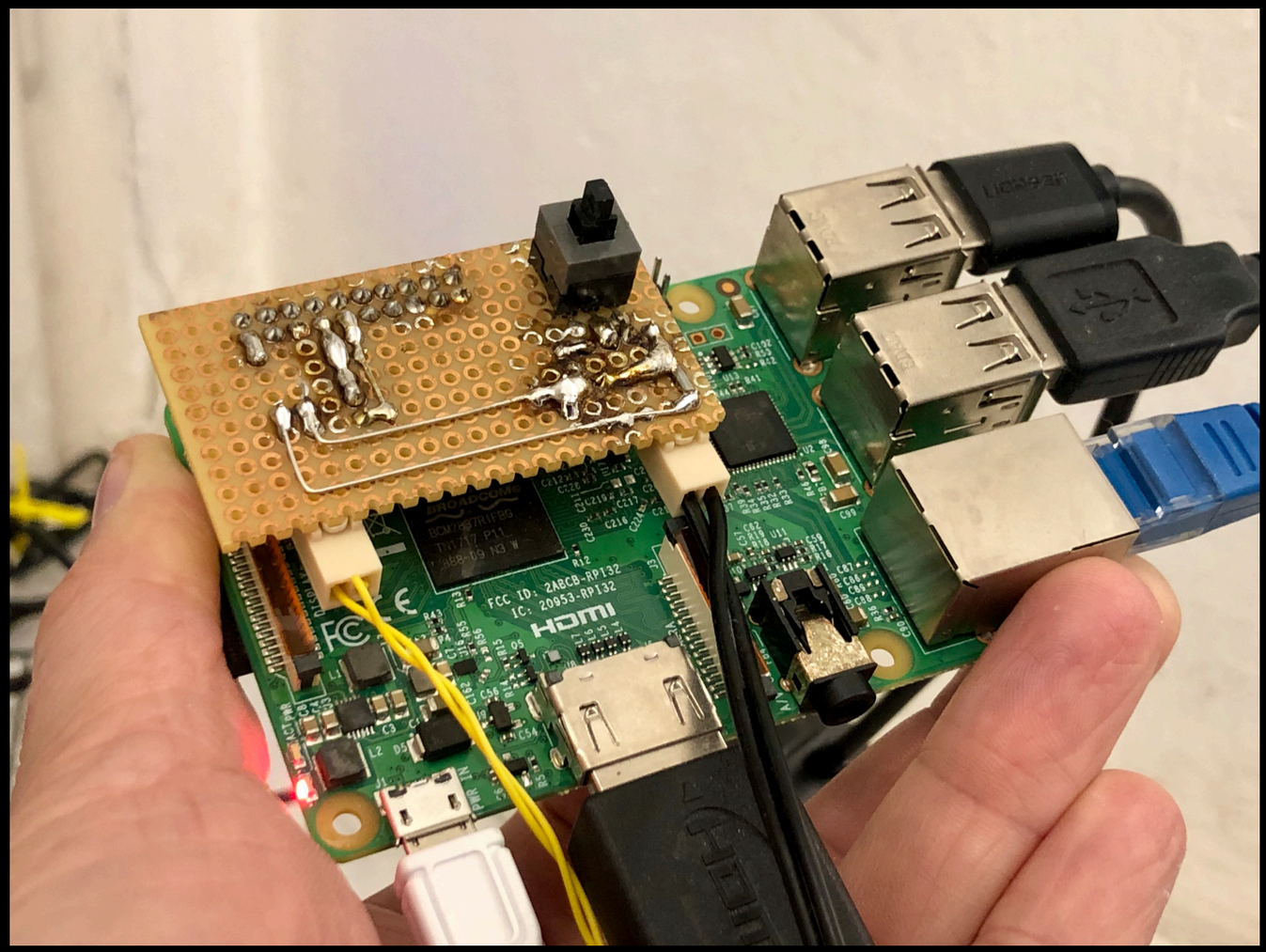
Performance, Performance, Performance!

Why?

Why?

Why?


4
OBERE WEISSGERBERSTRASSE


FCC ID: 2ABCB-RPI32
IC: 20953-RPI32
HDMI

Why?

Powered by Ruby!

```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*" system[1] and is intended for use by experienced C/C++ programmers. It is not a newbie learning tool.

Search Site

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:

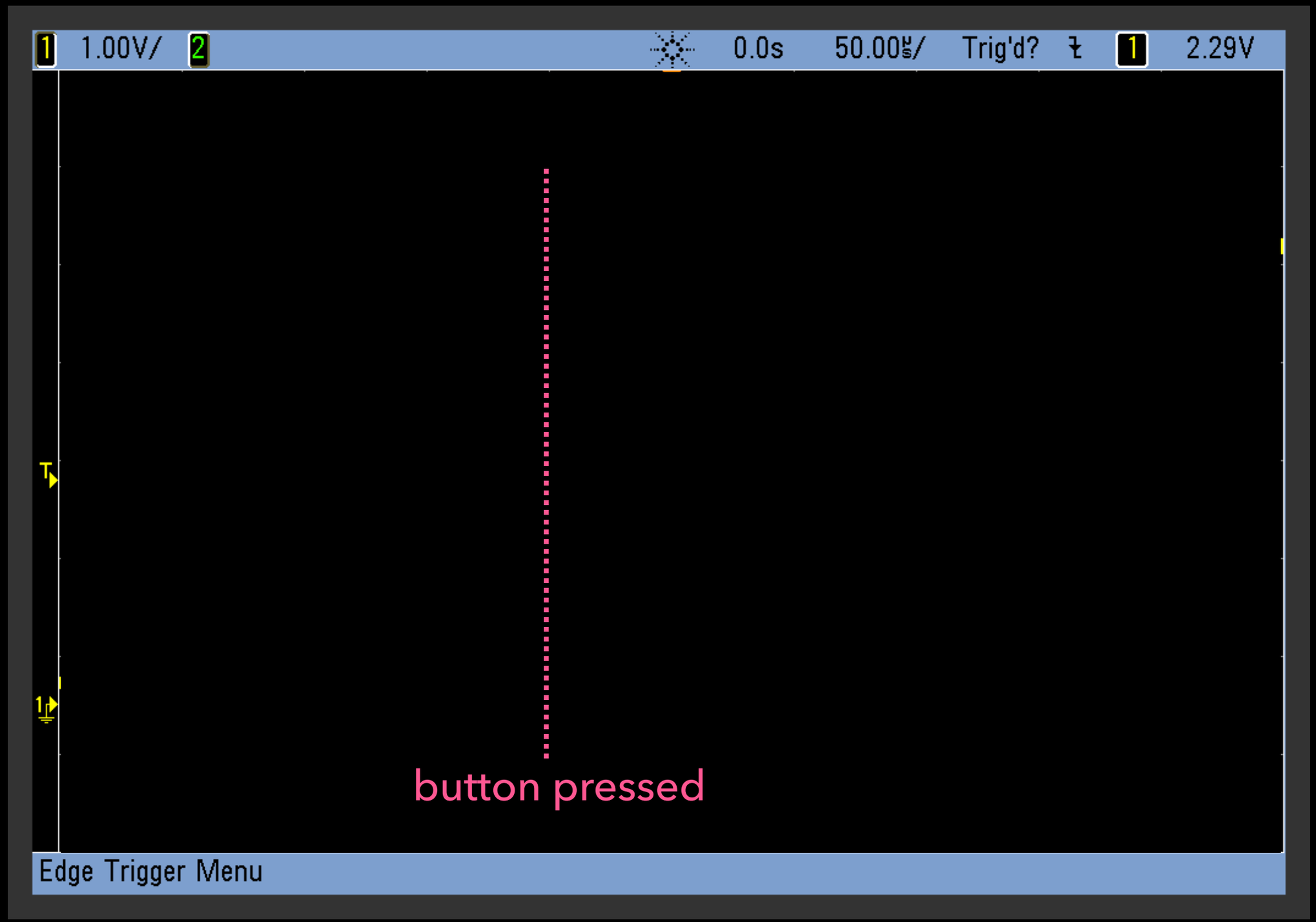
```
sudo gem install pi_piper
```

. . . also: 🐢 🐢 🐢

GPIO

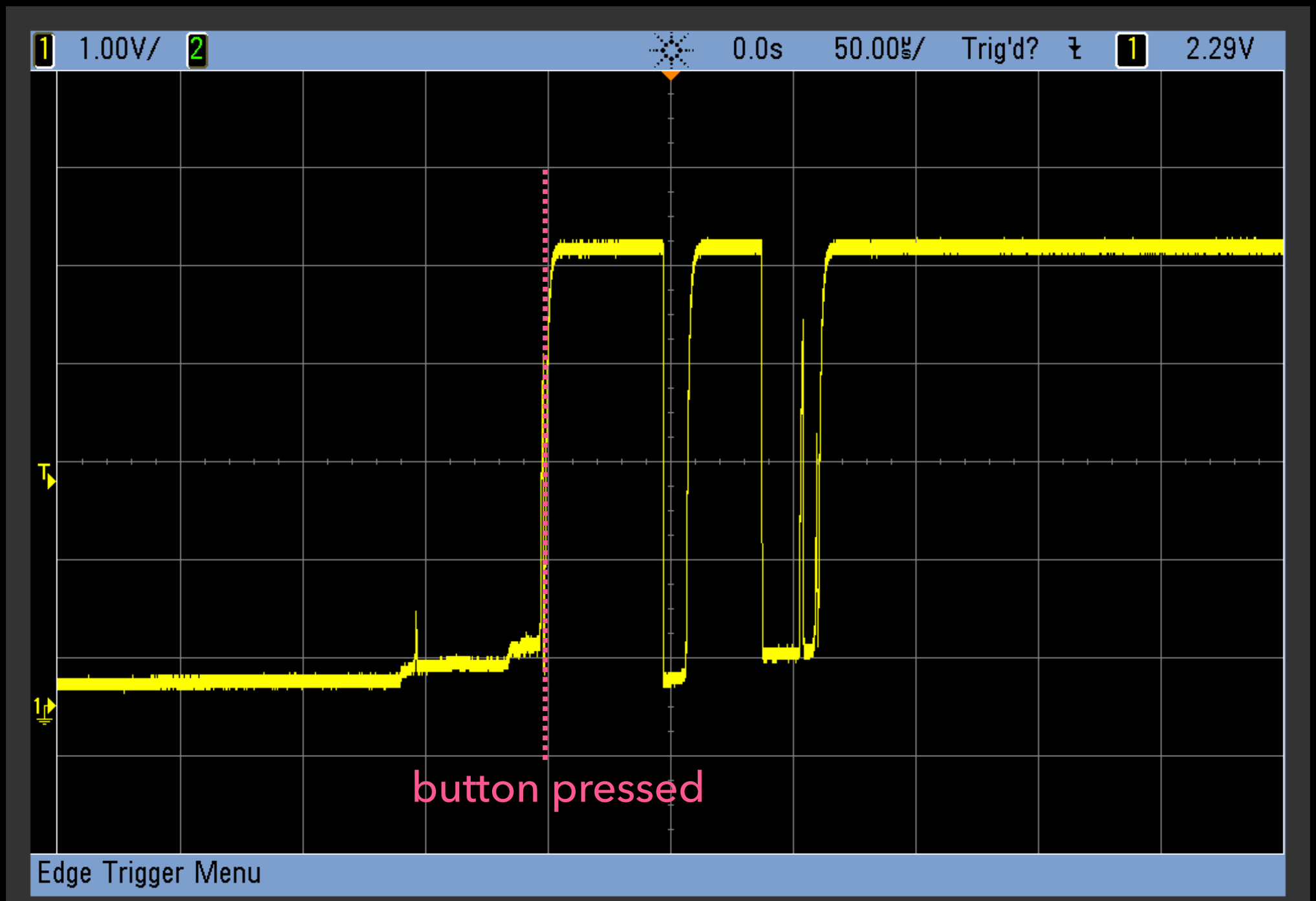
The GPIO pins (or General Purpose I/O pins) are the primary "do anything" pins on the Raspberry Pi. Reading inputs from them is as simple as:

Why is speed relevant?

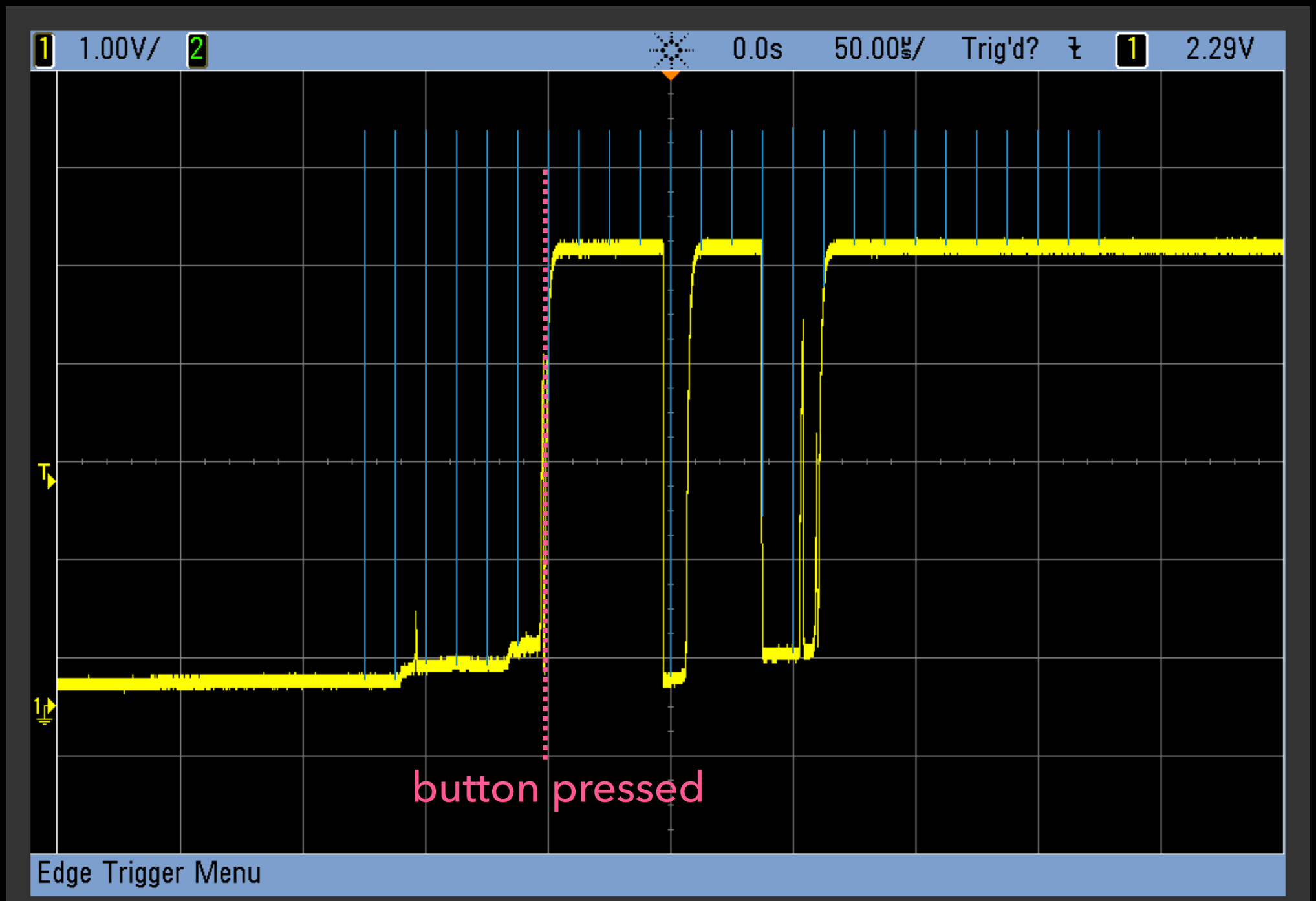


Why is speed relevant?

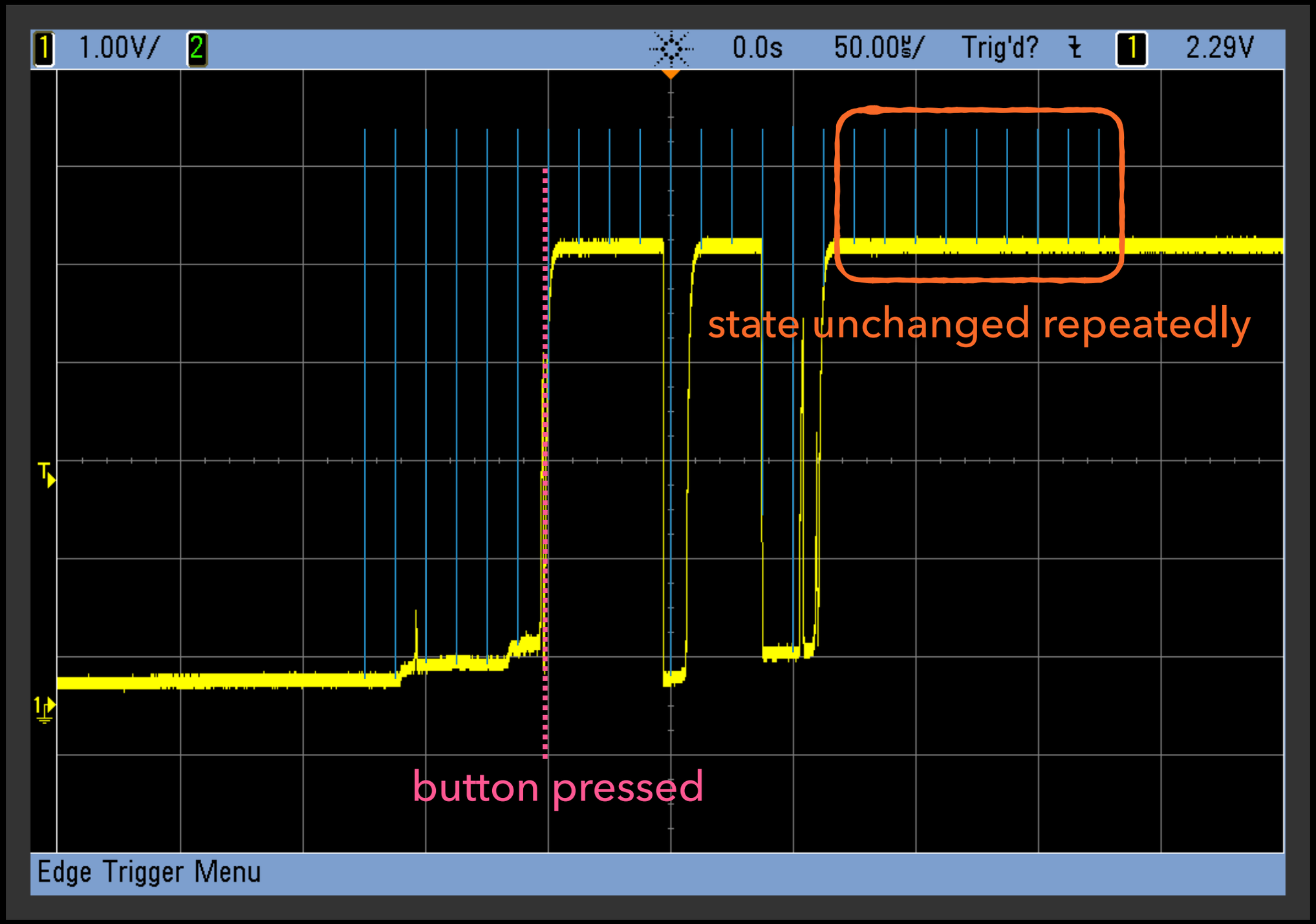
Why is speed relevant?



Why is speed relevant?



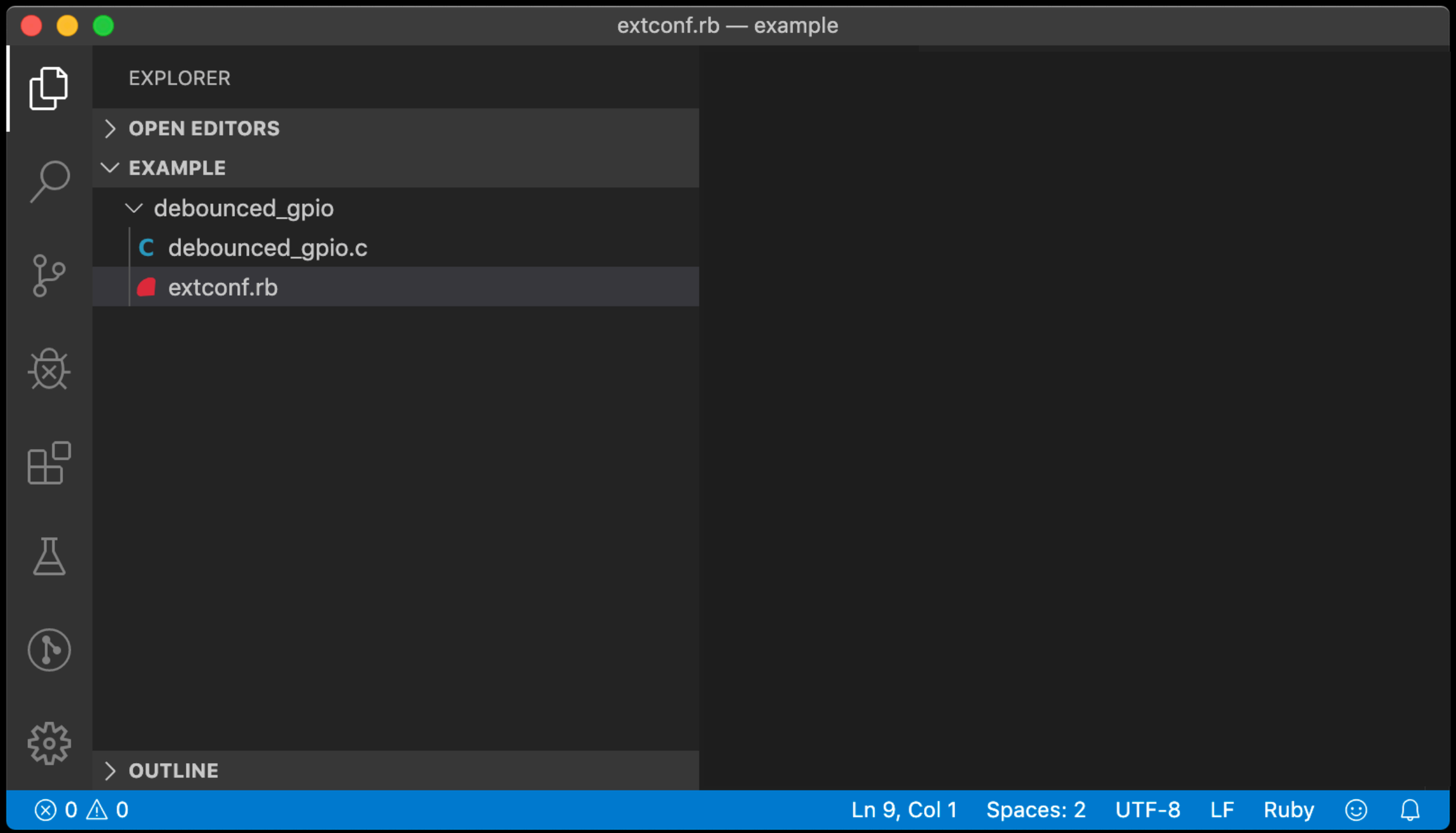
Why is speed relevant?



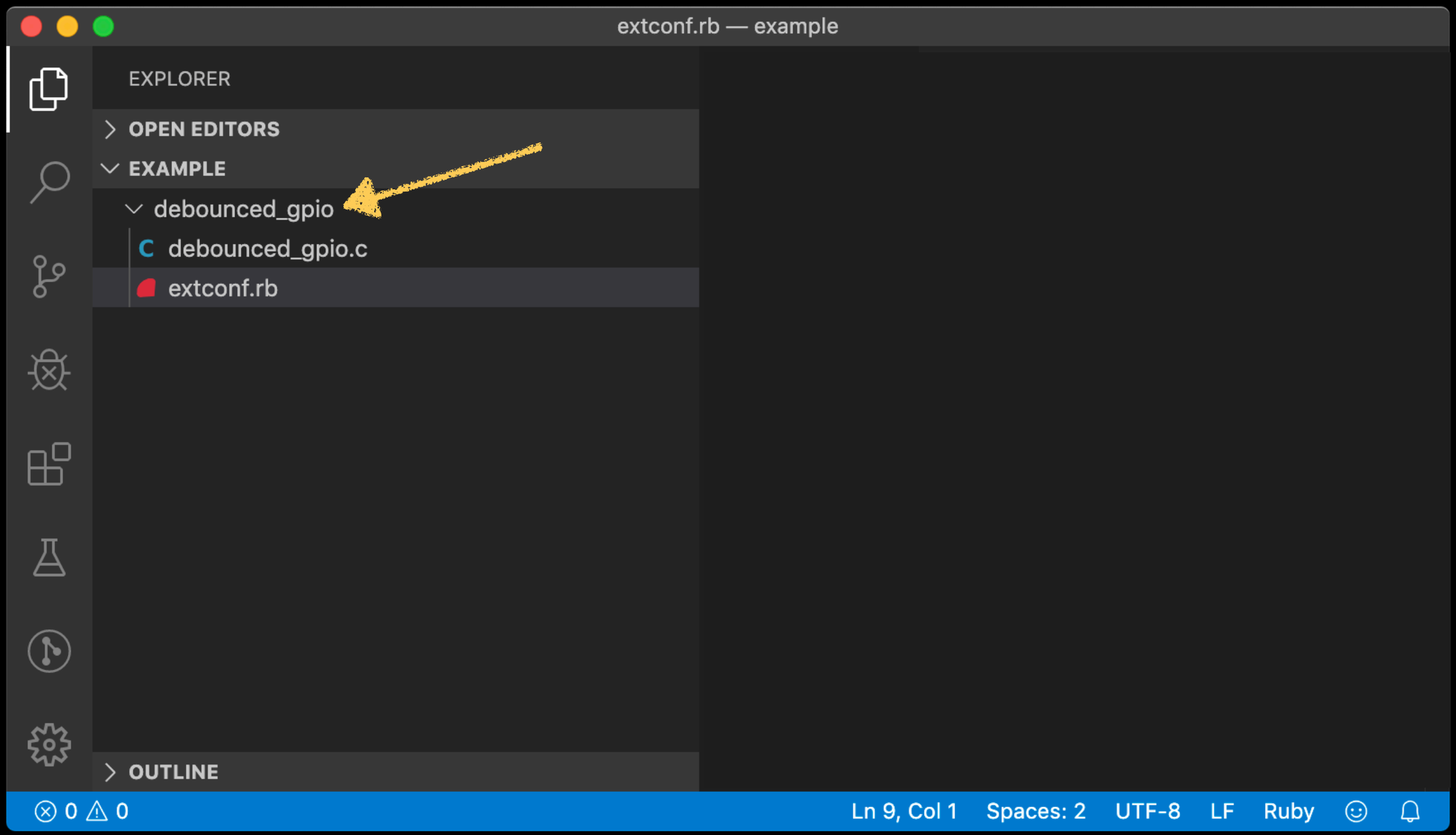
Back on track:

Let's make a C extension!

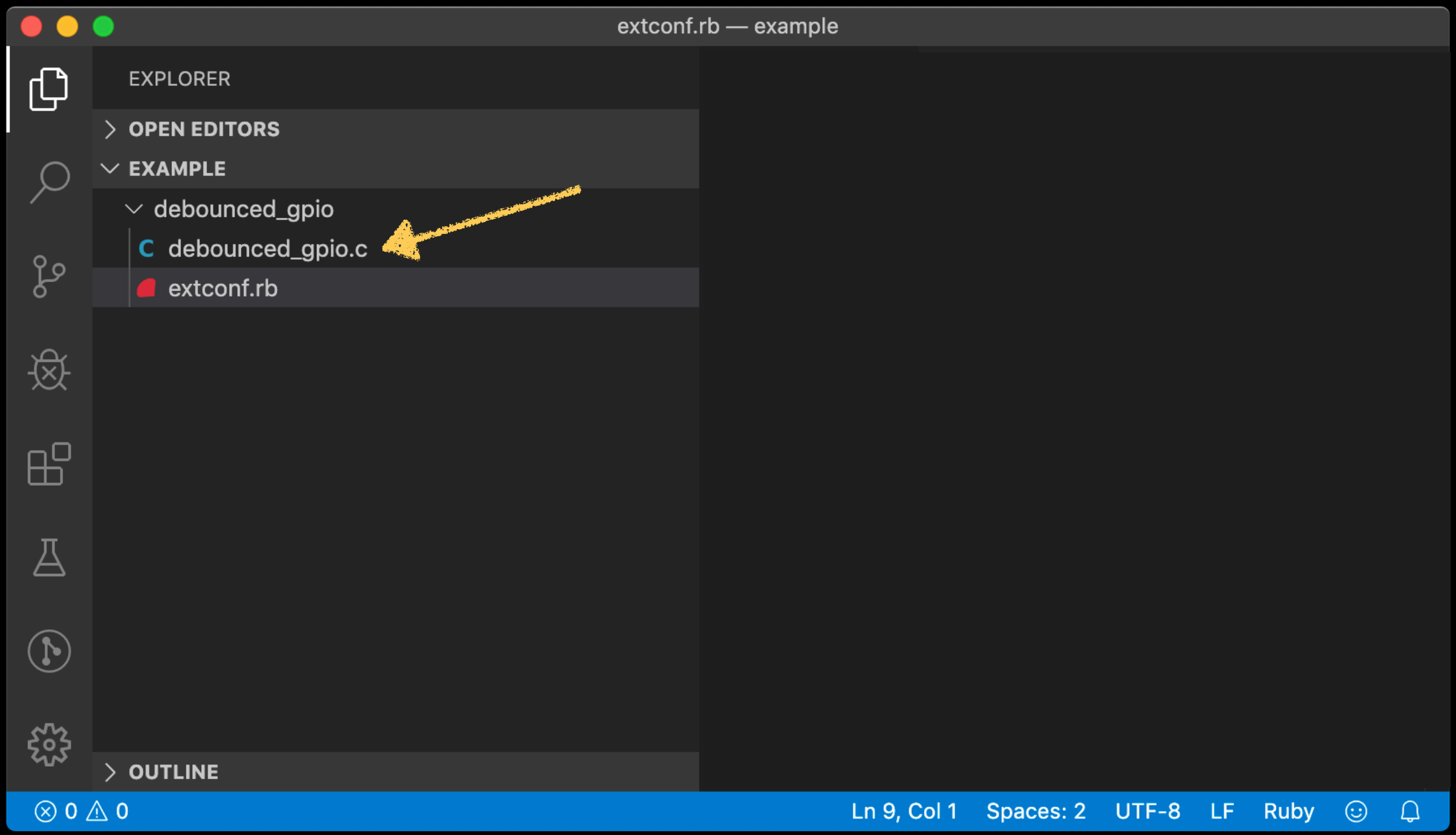
Let's make a C extension!



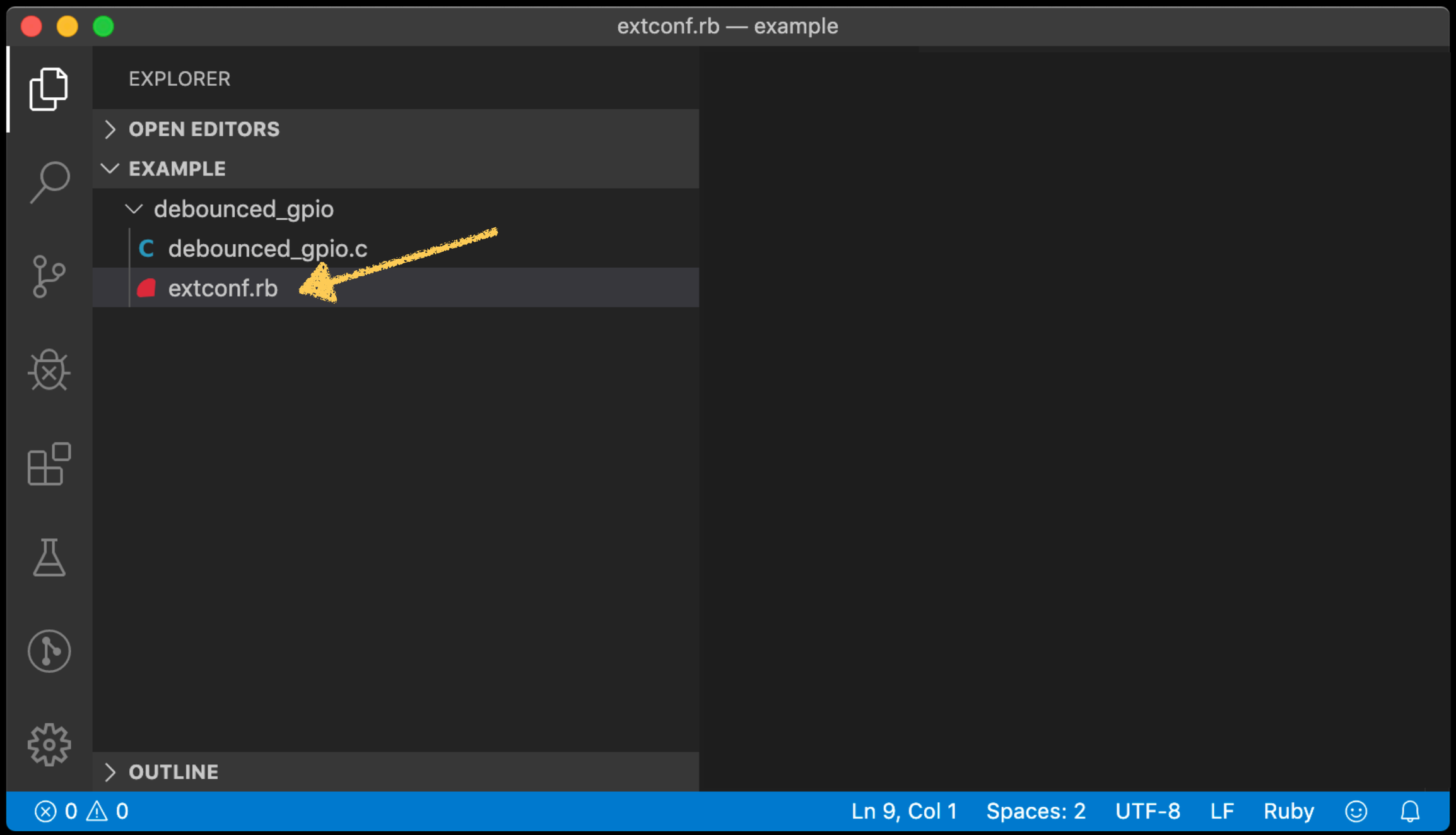
Let's make a C extension!



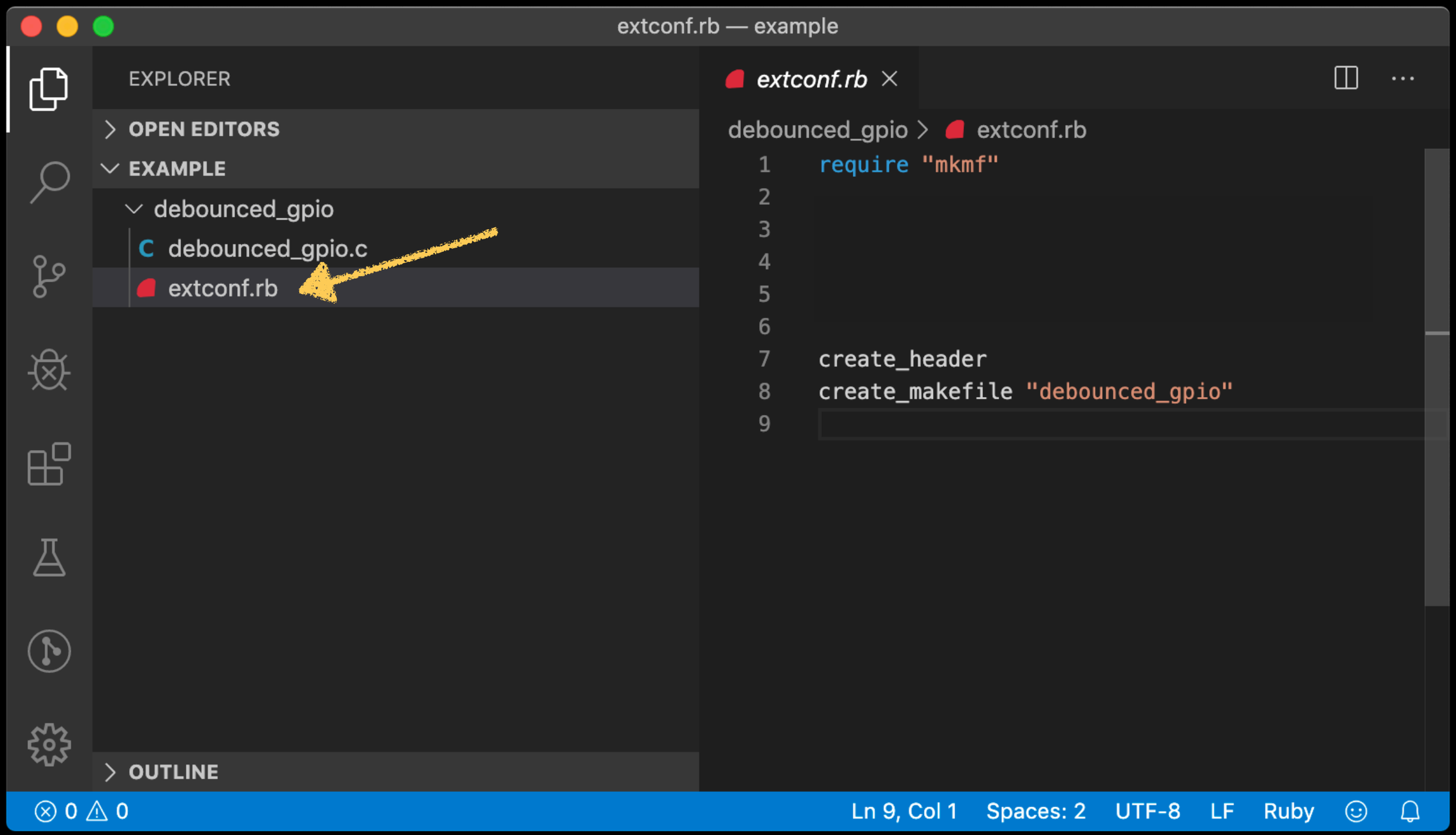
Let's make a C extension!



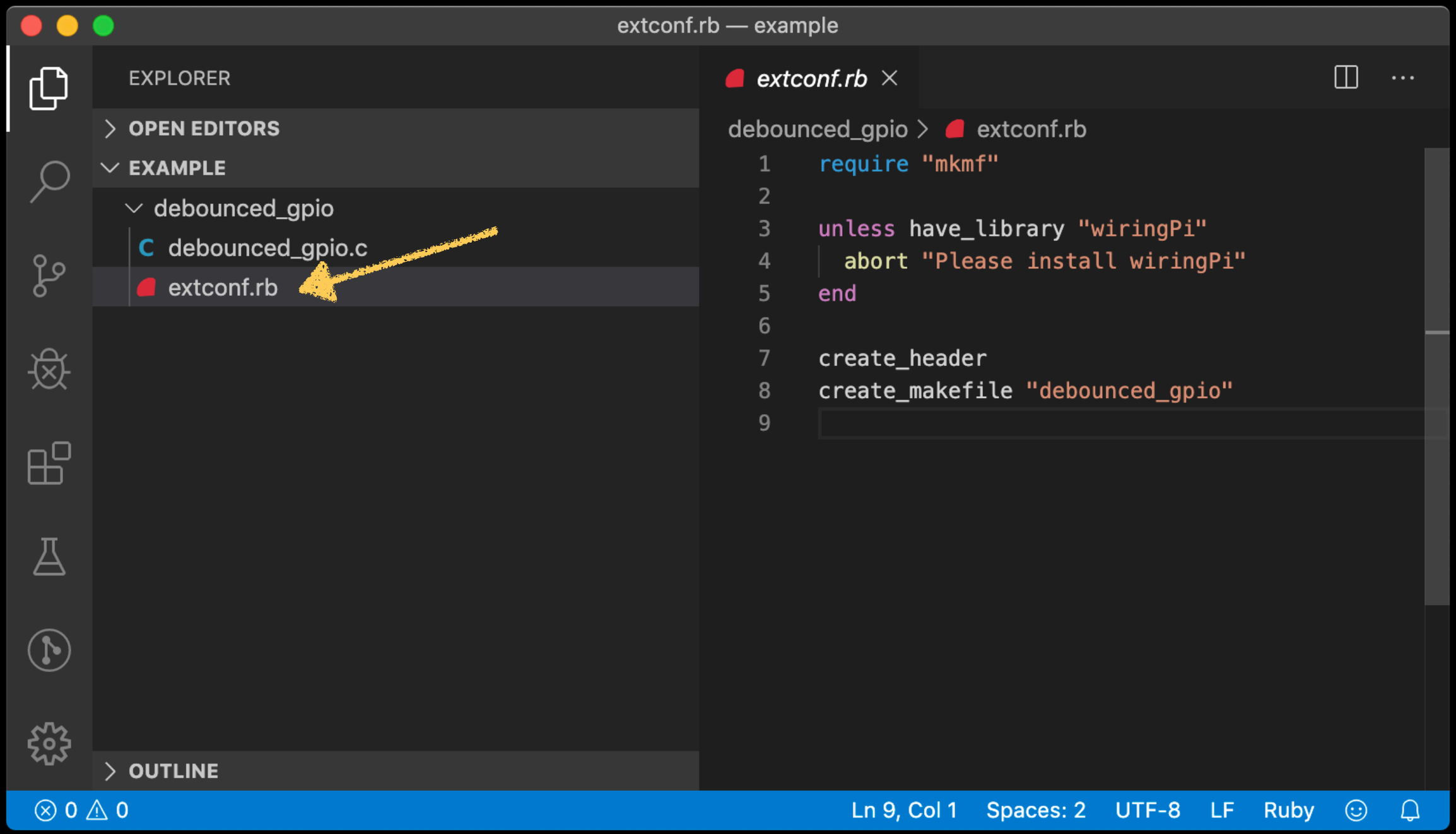
Let's make a C extension!



Let's make a C extension!

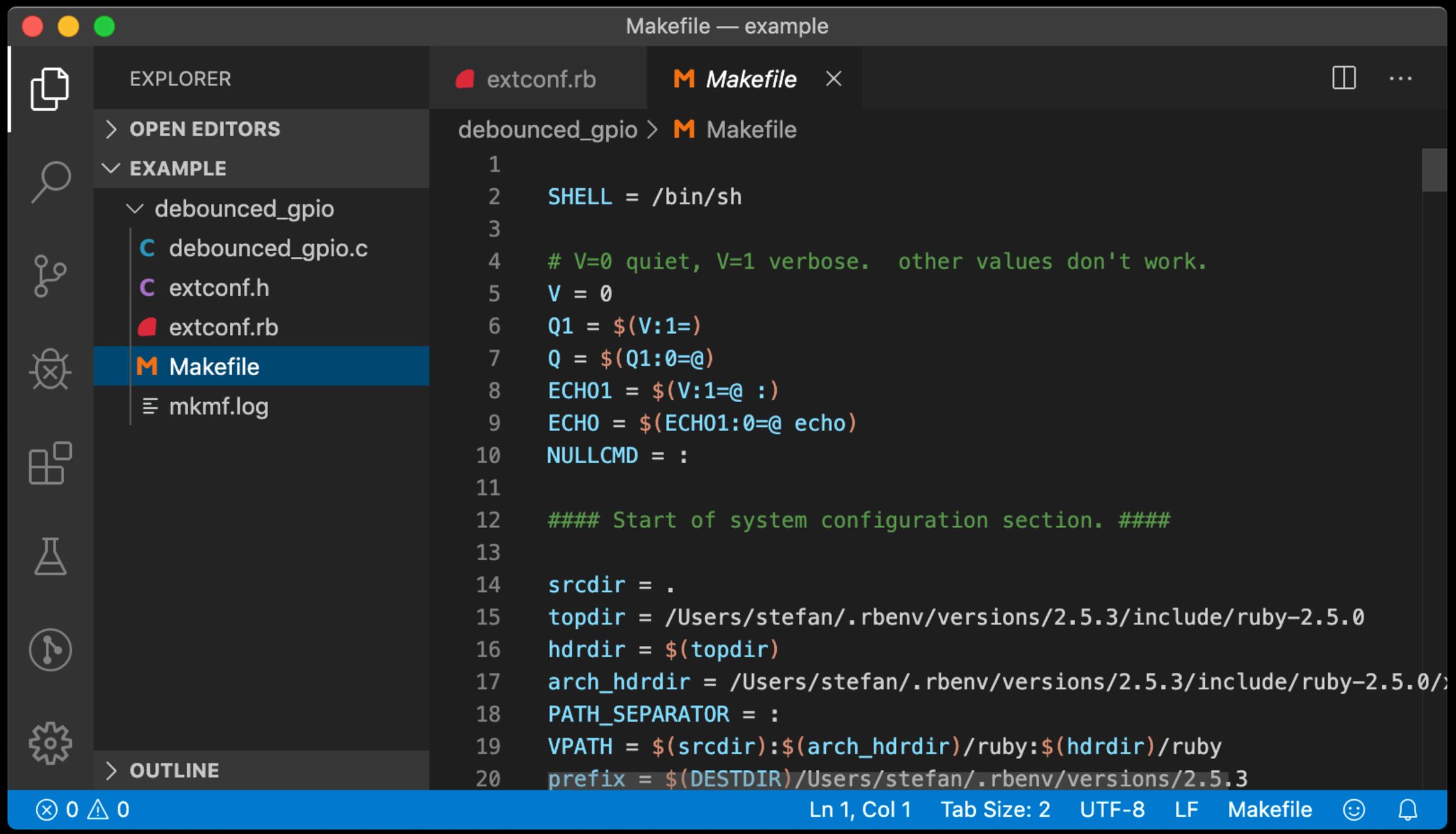


Let's make a C extension!



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

~$ ruby extconf.rb



~$ ruby extconf.rb

Makefile — example

EXPLORER

extconf.rb | Makefile

OPEN EDITORS

EXAMPLE

debounced_gpio

debounced_gpio.c

extconf.h

extconf.rb

Makefile

mkmf.log

OUTLINE

debounced_gpio > Makefile

```
1
2   SHELL = /bin/sh
3
4   # V=0 quiet, V=1 verbose.  other values don't work.
5   V = 0
6   Q1 = $(V:1=)
7   Q = $(Q1:0=@)
8   ECHO1 = $(V:1=@ :)
9   ECHO = $(ECHO1:0=@ echo)
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
```

0 0 Ln 1, Col 1 Tab Size: 2 UTF-8 LF Makefile

Minimum Viable C Extension

```c
#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

```c
#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

```c
#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

```c
#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
  );
}
```

```ruby
require_relative "debounced_gpio/debounced_gpio"

DebouncedGPIO.wait_for_state_change(42)

puts "Success!"
```

Minimum Viable C Extension

```c
#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
  );
}
```

```ruby
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

T_NIL	nil
T_OBJECT	ord
T_CLASS	cla
T_MODULE	mo
T_FLOAT	floa
T_STRING	stri
T_REGEXP	reg
T_ARRAY	arra
T_HASH	ass
T_STRUCT	(Ru
T_BIGNUM	mu
T_FIXNUM	Fixn
T_COMPLEX	cor
T_RATIONAL	rati
T_FILE	IO
T_TRUE	true

Manipulating Ruby Data

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:

String Functions

rb_str_new(const char *ptr, long len)	Creates a new Ruby string.
rb_str_new2(const char *ptr)	
rb_str_new_cstr(const char *ptr)	Creates a new Ruby string from a C string. This is equivalent to rb_str_new(ptr, strlen(ptr)).
rb_str_new_literal(const char *ptr)	Creates a new Ruby string from a C string literal.
rb_sprintf(const char *format, ...)	
rb_vsprintf(const char *format, va_list ap)	Creates a new Ruby string with printf(3) format. Note: In the format string, “%”PRIsVALUE 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”.
rb_str_append(VALUE str1, VALUE str2)	Appends Ruby string str2 to Ruby string str1.
rb_str_cat(VALUE str, const char *ptr, long len)	Appends len bytes of data from ptr to the Ruby string.

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

Data Types

The Ruby interpreter

T_NIL	nil
T_OBJECT	ord
T_CLASS	cla
T_MODULE	mo
T_FLOAT	floa
T_STRING	stri
T_REGEXP	reg
T_ARRAY	arra
T_HASH	ass
T_STRUCT	(Ru
T_BIGNUM	mu
T_FIXNUM	Fixn
T_COMPLEX	cor
T_RATIONAL	rati
T_FILE	IO
T_TRUE	true

Manipulating Ruby Data

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:

String Func

rb_str_new len)	
rb_str_new	
rb_str_new	
rb_str_new	
rb_sprintf	
rb_vsprint va_list ap)	
rb_str_append(VALUE str1, VALUE str2)	Appends Ruby string str2 to Ruby string str1.
rb_str_cat(VALUE str, const char *ptr, long len)	Appends len bytes of data from ptr to the Ruby string.

Use Ruby Features from C

There are several ways to invoke Ruby's features from C code.

Evaluate Ruby Programs in a String

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:

```
VALUE rb_eval_string(const char *str)
```

Evaluation is done under the current context, thus current local variables of the innermost method (which is defined by Ruby) can be accessed.

Note that the evaluation can raise an exception. There is a safer function:

```
VALUE rb_eval_string_protect(const char *str, int *state)
```

It returns nil when an error occurred. Moreover, *state is zero if str was successfully evaluated, or nonzero otherwise.

Odds and ends: Pardon the interruption!

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

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

```c
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!

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

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

```c
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!

```c
#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!

```c
#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!

```c
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!

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

VALUE rb_wait_for_state_change
  struct no_gvl_args args;
  args.pin = NUM2INT(pin);
  rb_thread_call_without_gvl(w
  return INT2NUM(args.newState
}
```

ruby/thread.c

```c
/*
 * rb_thread_call_without_gvl - permit concurrent/parallel execution.
 * rb_thread_call_without_gvl2 - permit concurrent/parallel execution
 *                               without interrupt process.
 *
 * rb_thread_call_without_gvl() does:
 *   (1) Check interrupts.
 *   (2) release GVL.
 *       Other Ruby threads may run in parallel.
 *   (3) call func with data1
 *   (4) acquire GVL.
 *       Other Ruby threads can not run in parallel any more.
 *   (5) Check interrupts.
 *
 * rb_thread_call_without_gvl2() does:
 *   (1) Check interrupt and return if interrupted.
 *   (2) release GVL.
 *   (3) call func with data1 and a pointer to the flags.
 *   (4) acquire GVL.
 *
 * If another thread interrupts this thread (Thread#kill, signal delivery,
 * VM-shutdown request, and so on), `ubf()' is called (`ubf()' means
 * "un-blocking function").  `ubf()' should interrupt `func()' execution by
 * toggling a cancellation flag, canceling the invocation of a call inside
 * `func()' or similar.  Note that `ubf()' may not be called with the GVL.
```

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

```ruby
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);
  }
}
```