



# A Beginner's Complete Guide to Microcontroller Programming with Ruby

hasumikin

**RubyConf Africa 2024**

Nairobi, Kenya  
27 July 2024

# Today's content

---

Part 1

## Preparation

Part 2

## Getting Started with Microcontroller

Part 3

## Exploring PicoRuby Further

Part 4

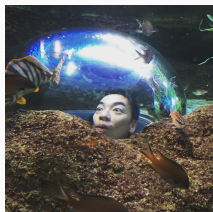
## PicoRuby Under the Hood





# self.inspect

- Hitoshi HASUMI
- @hasumikin (GitHub and Twitter)
- ANDPAD: construction tech 🧑‍🔧
- Creator of PicoRuby
- Contributor to CRuby and mruby
- Member of IRB maintainer team



# The former IRB maintainer

**Itoyanagi-san**



RubyConf Africa 2019



# Part 1

# Preparation



# Setup (minimal)

---

- Raspberry Pi Pico
  - Or other \*RP2040)-based controller
- USB cable
- Terminal emulator on laptop

# Raspberry Pi Pico

- Raspberry Pi Pico: Microcontroller board
  - MCU: RP2040
    - Cortex-Mzero+ (dual)
    - 264 KB RAM
    - 2 MB flash ROM
  - Generally runs without an OS
- Raspberry Pi: Single-board computer
  - Generally needs an OS like Raspberry Pi OS or Windows for Arm



# Terminal emulator (recommendation)

- Linux -> GTKTerm
- Windows -> Tera Term
- macOS -> PuTTY



# Let's begin 1/4

- Download the latest R2P2-\*.uf2 from GitHub

Jun 5

hasumikin

0.1.1

85a6b89

Compare

## Shell improved Latest

- You can copy & paste into the R2P2 shell (line by line. Multiple lines will fail)
- You can interrupt a task by `Ctrl-C`
- Shell can start even if the terminal size is small

▼ Assets 4

R2P2-FLASH_MSC-0.11-20230605-85a6b89.uf2.gz	571 KB	Jun 5
R2P2-FLASH_MSC-0.11-20230605-85a6b89.uf2.zip	571 KB	Jun 5
Source code (zip)		Jun 5
Source code (tar.gz)		Jun 5

1 person reacted

# Let's begin 1/4

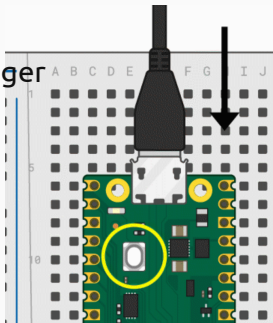
BTW, R2P2 stands for  
**R**uby on **R**aspberry **P**i **P**ico





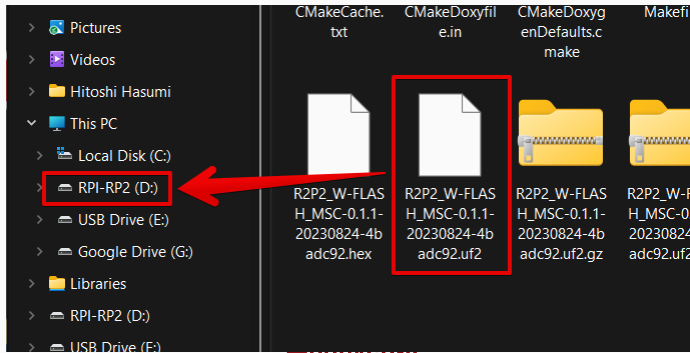
# Let's begin 2/4

- Connect Pi Pico and PC while pressing the BOOTSEL button
- You'll find "RPI-RP2" drive in file manager



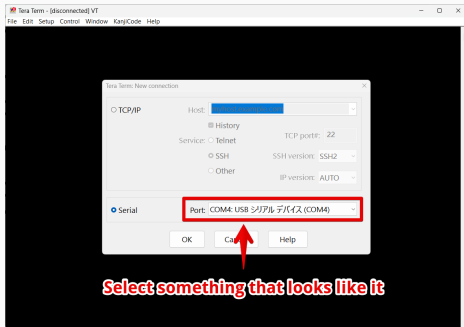
# Let's begin 3/4

- Drag & drop R2P2 - \* .uf2 into RPI-RP2 drive



# Let's begin 4/4

- Open a proper serial port on terminal emulator



## R2P2 Shell should start [Demo]

- Unix-like shell running on Raspberry Pi Pico
- You can use some commands like `cd`, `ls`, `mkdir`, and `irb`

COM10 - Tera Term VT

File Edit Setup Control Window KanjiCode Help

Initializing FLASH disk as the root volume...

Available

Press 's' to skip running app.mrb or app.rb

Starting shell...

by hasumikin

\$>

# PicoIRB [Demo]

- PicoRuby's IRB is running within the R2P2 shell on Raspberry Pi Pico
- Your Ruby snippet is compiled into mruby VM code and executed **on the fly**
- It means PicoRuby contains an mruby compiler which can run on a one-chip microcontroller (will be mentioned later)

Part 2

# Getting Started with Microcontroller

# GPIO (General Purpose Input/Output)

- Fundamental digital I/O
- Variety of uses:
  - Input: Detects on-off state of switch and button
  - Output: Makes a voltage
  - You can even implement a communication protocol by controlling GPIO in milli/micro sec

# GPIO — Blinking LED [Demo]

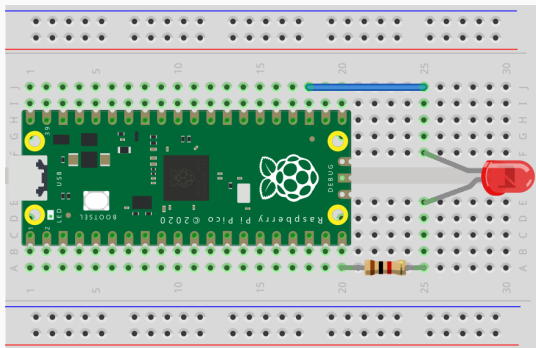
```
irb> led = GPIO.new(25, GPIO::OUT)
irb> 5.times do
irb*   led.write 1
irb*   sleep 1
irb*   led.write 0
irb*   sleep 1
irb* end
```

GPIO25 internally connects to  
on-board LED through a resistor



# GPIO — Blinking LED by discrete parts

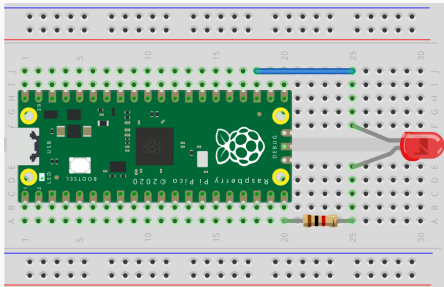
- Parts list:
  - LED (RED)
  - Resistor (1k $\Omega$ )



fritzing

# GPIO — Blinking LED by discrete parts

```
irb> pin = GPIO.new(15, GPIO::OUT)
```

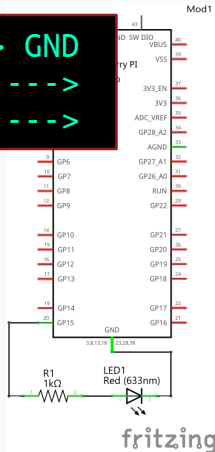


fritzing

# GPIO — Blinking LED by discrete parts

GPIO15 ==> 1kΩ ==> LED ==> GND  
<----- 1.5V -----><--- 1.8V ---->  
<----- 3.3V ----->

- RP2040's logic level: 3.3V
- LED voltage drop: **1.8V**  
(according to LED's datasheet)
- Current:  $(3.3V - 1.8V) / 1k\Omega = \mathbf{1.5mA}$   
(calculated by Ohm's Law)





# Study time: Physics

- Ohm's Law

- $V = I \cdot R \Leftrightarrow I = V / R \Leftrightarrow R = V / I$

- Kirchhoff's Circuit Laws

- Current law: The algebraic sum of currents in a network of conductors meeting at a point is zero
  - Voltage law: The directed sum of the potential differences (voltages) around any closed loop is zero

# Peripherals for serial communication

- I<sup>2</sup>C: To communicate between integrated circuits with support for multiple devices connected to the same bus
- SPI: To facilitate high-speed communication between microcontrollers and peripheral devices
- UART: To establish asynchronous serial communication between devices

# I<sup>2</sup>C — Inter-Integrated Circuit

```
irb> require 'i2c'  
irb> i2c = I2C.new(unit: :RP2040_I2C1, sda_pin: 26, scl_pin: 27)  
irb> [0x38, 0x39, 0x14, 0x70, 0x56, 0x6c].each { |i| i2c.write(0x3e, 0, i); sleep_ms 1 }  
irb> [0x38, 0x0c, 0x01].each { |i| i2c.write(0x3e, 0, i); sleep_ms 1 }  
irb> "Hello,".bytes.each { |c| i2c.write(0x3e, 0x40, c); sleep_ms 1 }  
irb> i2c.write(0x3e, 0, 0x80|0x40)  
irb> "Nairobi!".bytes.each { |c| i2c.write(0x3e, 0x40, c); sleep_ms 1 }
```



# LCD wraps I<sup>2</sup>C

```
# /lib/lcd.rb in R2P2 drive
require 'i2c'
class LCD
  ADDRESS = 0x3e # 0x7c == (0x3e << 1) + 0 (R/W)
  def initialize(i2c:)
    @i2c = i2c
    reset
  end
  def reset
    [0x38, 0x39, 0x14, 0x70, 0x56, 0x6c].each { |i| @i2c.write(ADDRESS, 0, i) }
    sleep_ms 200
    [0x38, 0x0c, 0x01].each { |i| @i2c.write(ADDRESS, 0, i) }
  end
  def putc(c)
    @i2c.write(ADDRESS, 0x40, c)
    sleep_ms 1
  end
  def print(line)
    line.bytes.each { |c| putc c }
  end
end
# ...
# See https://github.com/picoruby/picoruby/tree/master/mrbgems
# /picoruby-ble/example/broadcaster-observer
```

# LCD wraps I<sup>2</sup>C

```
irb> require 'lcd'  
irb> lcd = LCD.new(i2c: I2C.new(unit: :RP2040_I2C1, sda_pin: 26, scl_pin: 27))  
irb> lcd.print "Hello,"  
irb> lcd.break_line  
irb> lcd.print "Nairobi!"
```





# SPI — Serial Peripheral Interface

```
irb> require 'spi'
irb> spi = SPI.new(unit: :RP2040_SPI0, cipo_pin: 16,
                  cs_pin: 17, sck_pin: 18, copi_pin: 19)
irb> spi.select
irb> spi.write(255,255,255,255) # Reset
irb> spi.write(0x54)           # Start continuous mode
irb> data = spi.read(2).bytes
irb> temp = data[0] << 8 | data[1]
irb> temp / 128.0              # Convert to Celsius
=> 19.5621
```

# THERMO wraps SPI

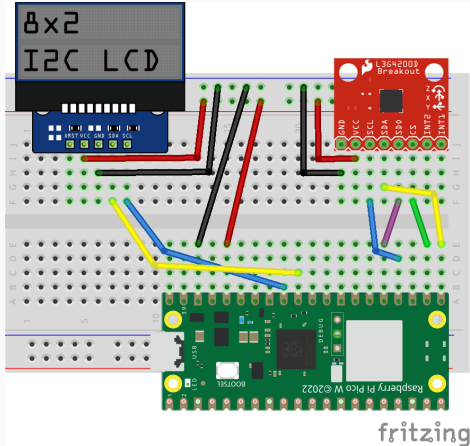
```
#!/lib/thermo.rb in R2P2 drive
require 'spi'
class THERMO
  def initialize(unit:, sck_pin:, cipo_pin:, copi_pin:, cs_pin:)
    @spi = SPI.new(unit: unit, frequency: 500_000, mode: 0, cs_pin: cs_pin,
      sck_pin: sck_pin, cipo_pin: cipo_pin, copi_pin: copi_pin
    )
    @spi.select
    @spi.write 0xFF, 0xFF, 0xFF, 0xFF # Reset
    @spi.write 0x54 # Start continuous mode
    sleep_ms 240
  end

  def read
    data = @spi.read(2).bytes
    temp = (data[0] << 8 | data[1]) >> 3
    # If it minus?
    temp -= 0x2000 if 0 < temp & 0b1_0000_0000_0000
    temp / 16.0 # Convert to Celsius
  end
end
# See https://github.com/picoruby/picoruby/tree/master/mrbgems
```

# THERMO wraps SPI

```
irb> require 'thermo'  
irb> thermo = THERMO.new(unit: :RP2040_SPI0,  
                           cipo_pin: 16, cs_pin: 17, sck_pin: 18, copi_pin: 19)  
irb> thermo.read  
=> 19.5621
```

# LCD and THERMO



# LCD and THERMO

```
irb> require 'lcd'  
irb> lcd = LCD.new(i2c: I2C.new(unit: :RP2040_I2C1, sda_pin: 26, scl_pin: 27))  
irb> require 'thermo'  
irb> thermo = THERMO.new(unit: :RP2040_SPI0,  
                           cipo_pin: 16, cs_pin: 17, sck_pin: 18, copi_pin: 19)  
irb> lcd.print sprintf("%5.2f \xdfC", thermo.read)
```



# Part 3

## Exploring PicoRuby Further



# PicoRuby killer applications

## • R2P2

- Microcontroller application framework
- Unix-like shell system and IRB written in PicoRuby

## • PRK Firmware

- Keyboard firmware framework for DIY keyboard
- You can write your keymap and keyboard's behavior with Ruby

# R2P2 (again)

- IRB
  - Multiple-line editor
- Built-in commands and executables (all written in Ruby)
- You can write your own external command



# Executables in R2P2, for example,

```
# date  
puts Time.now.to_s
```

```
# mkdir  
Dir.mkdir(ARGV[0])
```

# Write a Ruby script file [Demo]

```
$> vim hello.rb
```

Edit the file and save it.

```
puts "Hello World!"
```

Then run it.

```
$> ./hello.rb
```

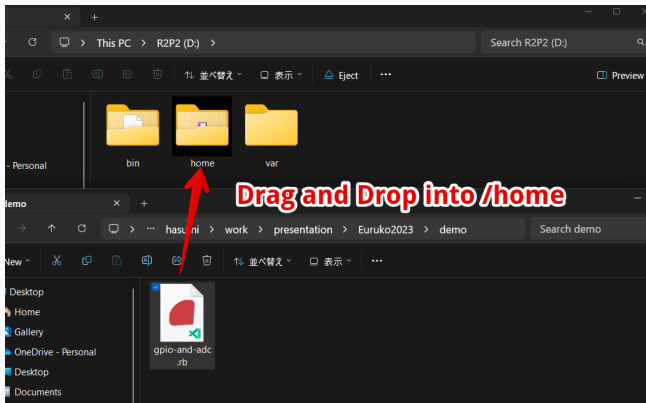
# Make a standalone IoT device

```
# `/home/app.rb` automatically runs
require 'lcd'
require 'thermo'

led = GPIO.new(25, GPIO::OUT)
lcd = LCD.new(i2c: I2C.new(unit: :RP2040_I2C1,
                           sda_pin: 26, scl_pin: 27))
thermo = THERMO.new(unit: :RP2040_SPI0, cipo_pin: 16,
                    cs_pin: 17, sck_pin: 18, copi_pin: 19)

# Stop infinite loop by Ctrl-C
while true
  temp = thermo.read
```

# Make a standalone IoT device



# Part 4

## PicoRuby Under the Hood



# mruby and PicoRuby

- mruby
  - General purpose embedded Ruby implementation written by Matz
- PicoRuby (PicoRuby compiler + mruby/c VM)
  - Another implementation of mruby targeting on one-chip microcontroller (**smaller foot print**)
  - Based on the mruby's VM code standard

# Small foot print

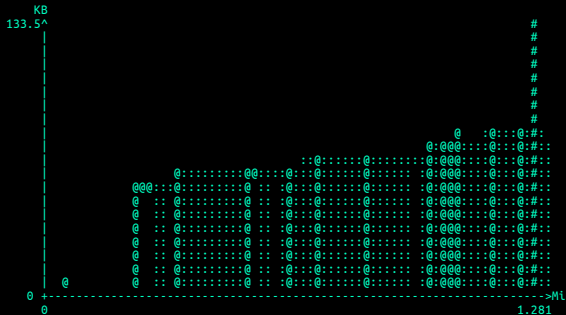
```
$ valgrind \
  --tool=massif \
  --stacks=yes \
  path/to/(mruby|picoruby) \
  -e 'puts "Hello World!";'
```

massif.out.[pid] file will be created. Then,

```
$ ms_print massif.out.1234 | less
```

# Small foot print

```
-----  
Command:      mruby -e 'puts "Hello World!'"  
Massif arguments:  --stacks=yes  
ms_print arguments: massif.out.18391  
-----
```

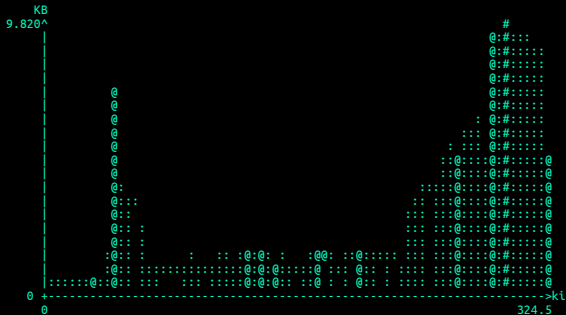


Note: Measured in 64 bit Ubuntu



# Small foot print

```
-----  
Command:      picoruby -e 'puts "Hello World!"  
Massif arguments:  --stacks=yes  
ms_print arguments: massif.out.21752  
-----
```



Note: Measured in 64 bit Ubuntu

# Small foot print

- RAM consumption of puts "Hello World!"
  - mruby: 133.5 KB (on 64 bit)
  - PicoRuby: 9.82 KB (on 64 bit)
- RP2040 (32 bit) has 264 KB RAM
  - Only small applications written in mruby should work
  - R2P2 and PRK Firmware should be written in PicoRuby

# PRK Firmware: DIY keyboard firmware



# PRK Firmware on Meishi2 (4-keys macro pad)



# PRK Firmware on Meishi2 (4-keys macro pad)

```
require "consumer_key"
kbd = Keyboard.new
kbd.init_pins(
  [ 6, 7 ],    # row0, row1
  [ 28, 27 ]   # col0, col1
)
kbd.add_layer :default, %i[ RAISE  KC_2  KC_A  KC_4 ]
kbd.add_layer :raise,   %i[ RAISE
                           KC_AUDIO_VOL_UP
                           KC_AUDIO_VOL_DOWN
                           KC_AUDIO_MUTE ]
kbd.define_mode_key :RAISE, [ :KC_SPACE, :raise, 200, 200 ]
kbd.start!
```

# PicoRuby ecosystem

- Picogems

- PRK Firmware is also a Picogem

- Peripheral gems

- picoruby-gpio, picoruby-adc, picoruby-i2c, picoruby-spi, picoruby-uart, picoruby-pwm

- Peripheral interface guide

- <https://github.com/mruby/microcontroller-peripheral-interface-guide>



# PicoRuby ecosystem

---

- Build system forked from mruby
  - You can build your application in a similar way to mruby
  - You can also write your gem and host it on your GitHub
- RP2040 is the only target as of now though,
  - Carefully designed to keep portability

# Conclusion

- PicoRuby is a Ruby implementation targeting on one-chip microcontroller
- You can prototype your microcontroller application step-by-step using R2P2 and IRB
- You don't need any compiler or linker
- Educational and fun to learn about microcontroller programming



# Stargaze at



[github.com/picoruby/picoruby](https://github.com/picoruby/picoruby)

