mruby/c

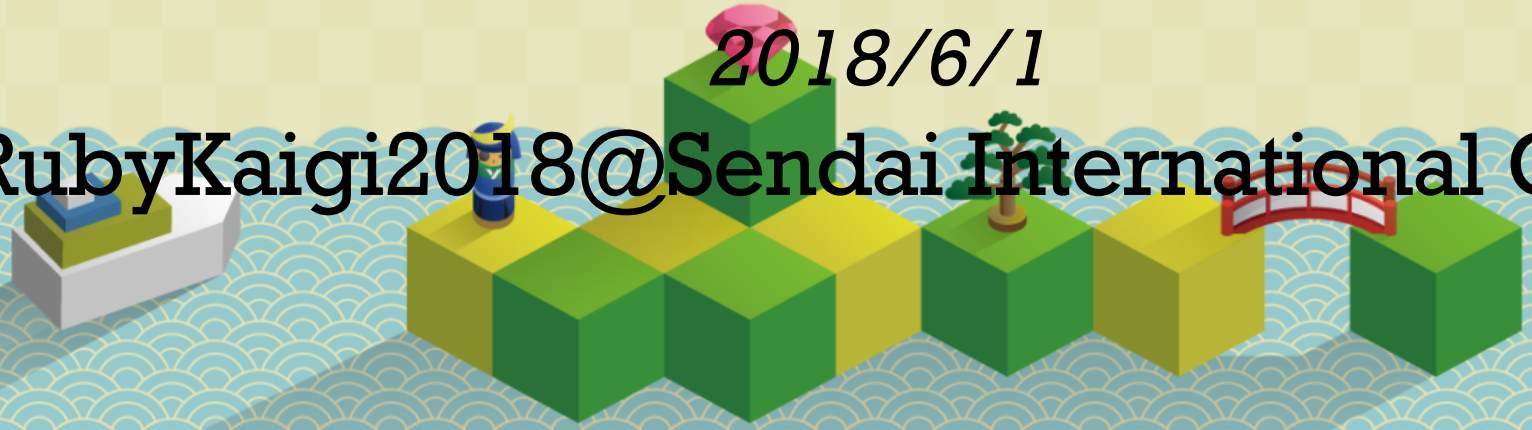
Firmware Programming with mruby/c

HASUMI Hitoshi

Monstar Lab, Matsue office

2018/6/1

RubyKaigi2018@Sendai International Center



before starting,

- assumed audience
  - people who want to make IoT with Ruby
  - people who have never touched mruby/c
  - people who did only tutorials of mruby/c
  - **people who love Sake**
- what this talk doesn't mention
  - microcontroller itself and peripheral concepts like GPIO.

recommended book about microcontroller

recommended tutorial

http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_tutorial/

agenda

- terminology
- about my IoT project
- mruby and mruby/c
- how does mruby/c work
- debugging
- actual source code of my project
- development environment

terminology

terminology

- mruby/c
  - I say エムルビーシー[emurubi:ʃí:] in this talk
- microcontroller = マイコン[maikon]
  - small computer contains CPU, memory and programmable I/O peripherals
  - in this talk, microcontroller is distinguished from single board computer like Raspberry Pi. RasPi is rich, microcontroller is poor

terminology

- RTOS
  - Real-time OS. usually used for microcontroller. but we won't use it
- task
  - almost equivalent to `thread` in linux. we say `task` in microcontroller world

terminology

- 旭日酒造(asahi-shuzo)
  - one of the best Japanese Sake brewery
  - I also call the brewery **asahi-san**
  - asahi-san and I make IoT system using mruby/c

terminology (IMO)

- 旭酒造(asahi-shuzo), yamaguchi: 獺祭(dassai). fruity, aromatic and sweet
- 朝日酒造(asahi-shuzo), niigata: 久保田(kubota). clear, dry and sharp
- 朝日酒造(asahi-shuzo), fukui: I don't know at all...
- 旭日酒造(asahi-shuzo), shimane: **十旭日(juji-asahi)**. tasteful, mature with years and good for お燗(warmed style)

why microcontroller?

(rather than single board computer)

why microcontroller?

- it starts immediately right after plugged in
  - end users, brewery workers in my case, can use simply
- you can narrow security issue list
  - many a malware aims at linux or windows platform as a target
  - you don't need to consider unnecessary deamon
  - you don't need to do `apt upgrade`

why microcontroller?

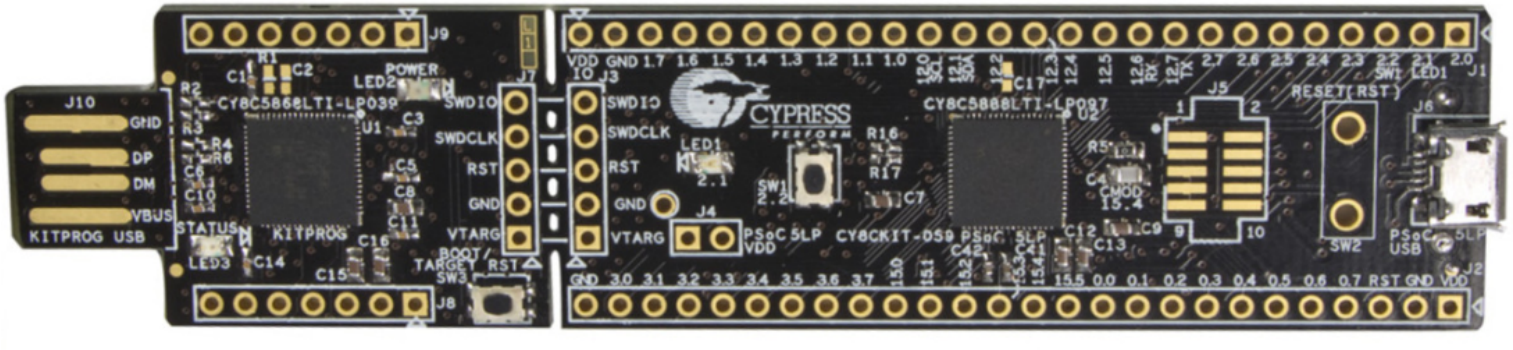
- low energy
  - rarely overheated
  - many choices of power supply
- mass production
  - you can choose appropriate chipset (number of GPIOs, memory size, etc.) for your application
  - cost advantage for parts supply and subcontractor manufacturing

which microcontroller?

which microcontroller?

- CYPRESS PSoC5LP
  - 32-bit Arm Cortex-M3 CPU
  - Flash/SRAM: 256KB/64KB
  - package: 68-pin QFN, 99-pin WLCSP, 100-pin TQFP

which microcontroller?



- PSoC5LP Prototyping Kit
  - for prototyping and small production
  - you can purchase with 秋月電子通商 (akizukidenshi.com)

mruby/c works on only PSoC5LP?

mruby/c works on only PSoC5LP?

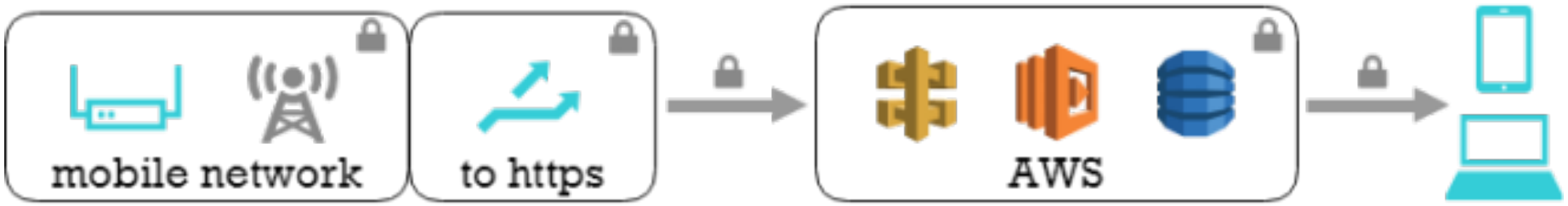
- yes and no
  - the number of microcontroller on which mruby/c runs is increasing
  - PSoC5LP is the most recommended one. because it has a good example, my project
- PSoC5LP is very resonable to develop IoT product especially for mass production
  - IMO, the mruby/c core team made the right choice

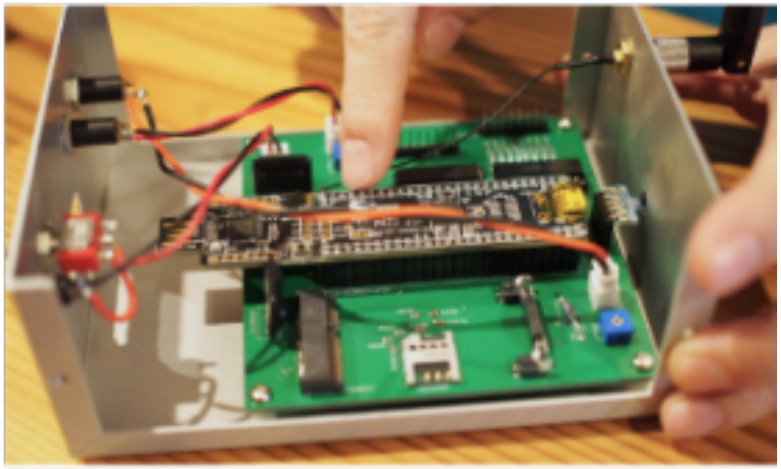
about my IoT project

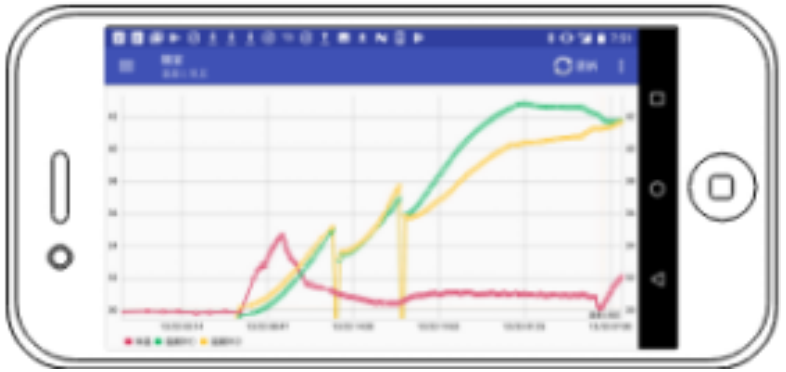
about my IoT project

- IoT system for asahi-san
- delivered to actual brew work in January 2018
- devices post temperature of Sake materials in brewing, surrounding temperature and humidity to server
- data is displayed on smartphone app

about my IoT project







about my IoT project

what were difficult about mruby/c?

- we can neither do step execution nor look into appropriate memory address of mruby/c's variables
- so many factors to be trouble in IoT
  - hard to find why the application doesn't work well
- mruby/c is growing
  - bugs, lack of docs and examples

about my IoT project

so, is mruby/c bad?

about my IoT project

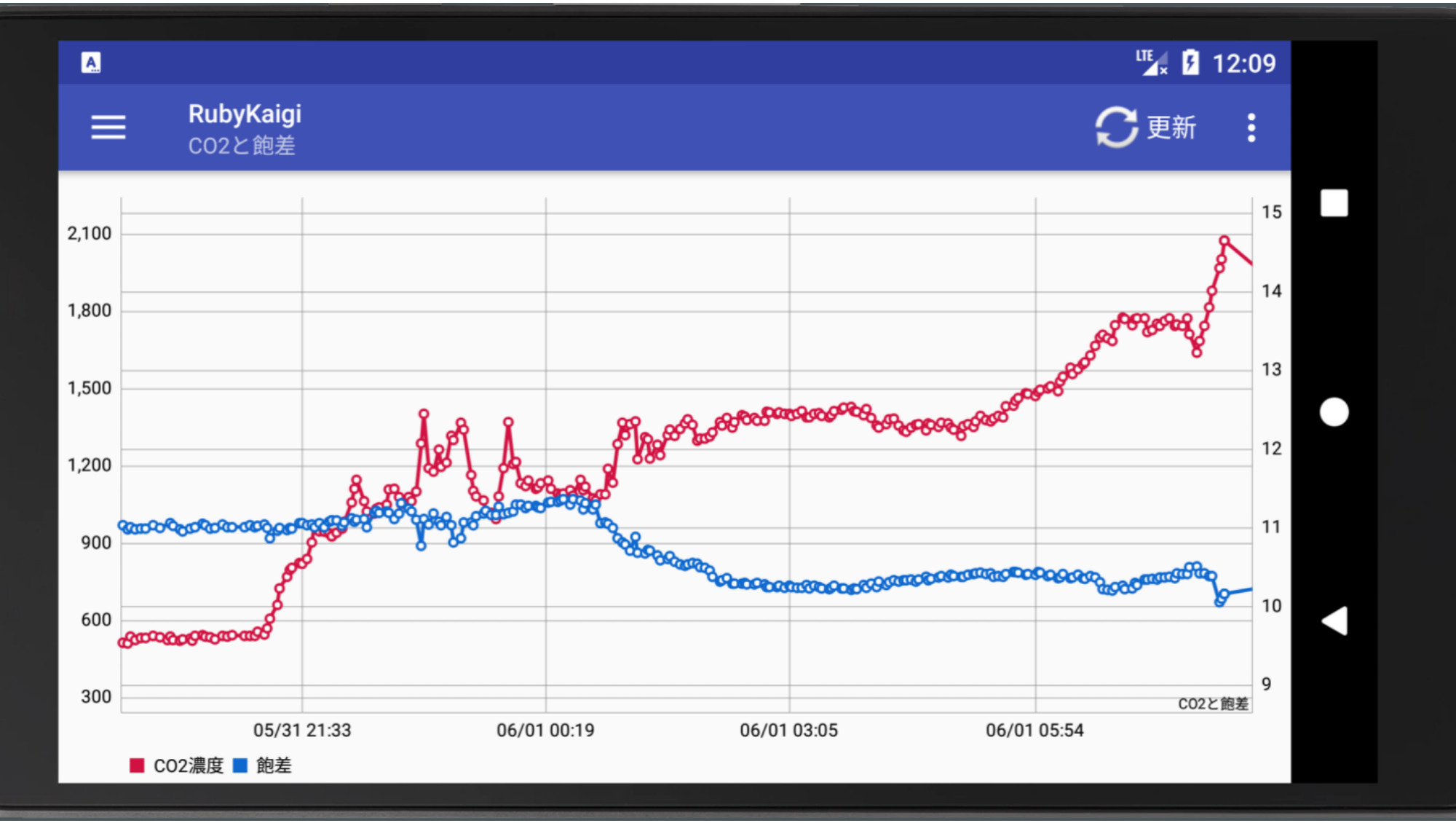
so, is mruby/c bad? - NO

- IoT at work makes you hurry
  - you have to go alternately to dark 10℃ storage cellar and humid 35℃ manufacturing room
  - brewery workers run around
  - you have to amend your firmware with your small laptop in 10 minutes
  - you will thank Ruby's descriptiveness and agility

today's demo

- CO_2 concentration
  - 400ppm : atmospheric
  - 1000ppm : your programming speed decreases
  - 1500ppm : tomatoes may grow well
  - > 2000ppm : sleepy, headache
  - > 40000ppm : 💀
- HD = humidity deficit = 飽差
  - 3~6g : tomatoes grow well

today's demo


12:09
RubyKaigi
CO2と飽差
更新
2,100
1,800
1,500
1,200
900
600
300
15
14
13
12
11
10
9
CO2と飽差
05/31 21:33
06/01 00:19
06/01 03:05
06/01 05:54
CO2濃度
飽差

today's demo

- prev page's graph shows CO2 of room I stayed last night
- CO2 concentration went up though 24 hour ventilation is mandatory
- it is terrible that CO2 reached 2000ppm when I should have wake up, isn't it?!
- the device will measure while I talk today. so I prove that it is due to CO2 if someone slept while I speaking

so many factors to be trouble in IoT

so many factors to be trouble in IoT

- peripheral equipments (☆)
- circuit and wiring design
- printed circuit board = PCB
- soldering (☆)
- C, mruby and mruby/c (☆)
- communication timing control (☆)
- network

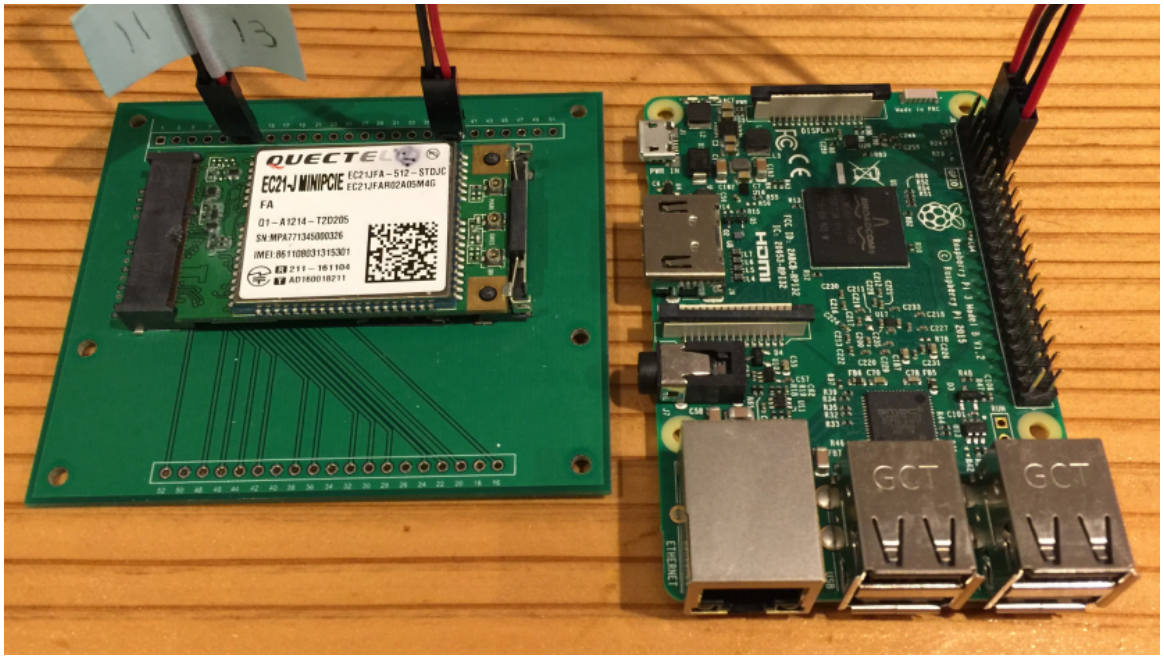
☆...I will explain these topics

peripheral equipments

peripheral equipments

- it is very important to check these things before writing application code
  - equipment like sensor or communication module works as its spec sheet
  - whether the equipment is not broken (sometimes broken by soldering 😭)
- unless you will regret
  - so…

peripheral equipments



- Raspberry Pi & CRuby are great for pre-prototyping
  - use breadboard or make PCB for test like this photo

peripheral equipments

ex) CRuby for serial communication test

```ruby
# notice this is CRuby for RasPi
require 'rubyserial'
require 'timeout'
BAUDRATE = 9600 # match with your instrument
sp = Serial.new '/dev/serial0', BAUDRATE, 8
loop do
  puts '[command]'
  command = gets
  sp.write command.sub("\n", "\r") # replace LF if needed
  sleep 0.1
  result = ''
  begin
    Timeout.timeout(10) do
      loop do
        line = sp.read(128)
        break if line == '' && result != ''
        result << line
        sleep 0.1
      puts '=> ' + result
  rescue Timeout::Error
    puts 'timeout!'
ennnnd
```

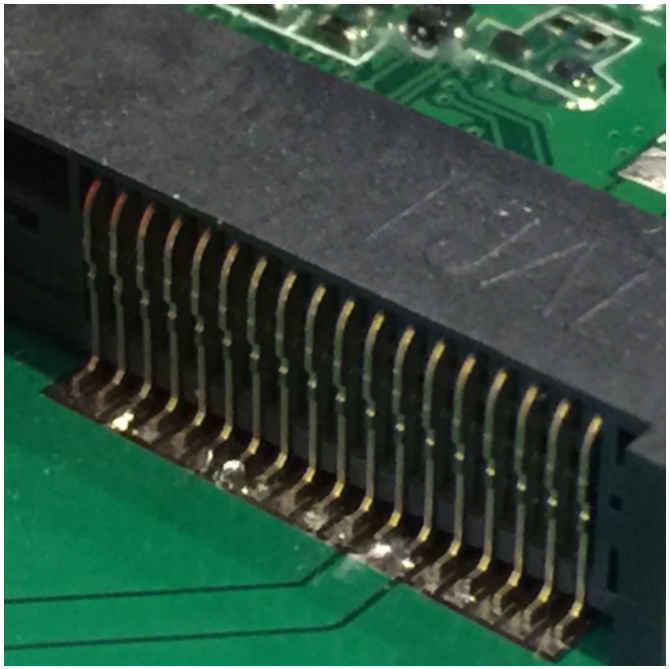
peripheral equipments

ex) CRuby for serial communication test

```
$ serial_communication_test.rb
[command]
AT                  # command
=> OK               # response
[command]
AT+CIMI             # command
=> 123456789012     # response
[command]
AT+XXX              # command
=> error            # response
```

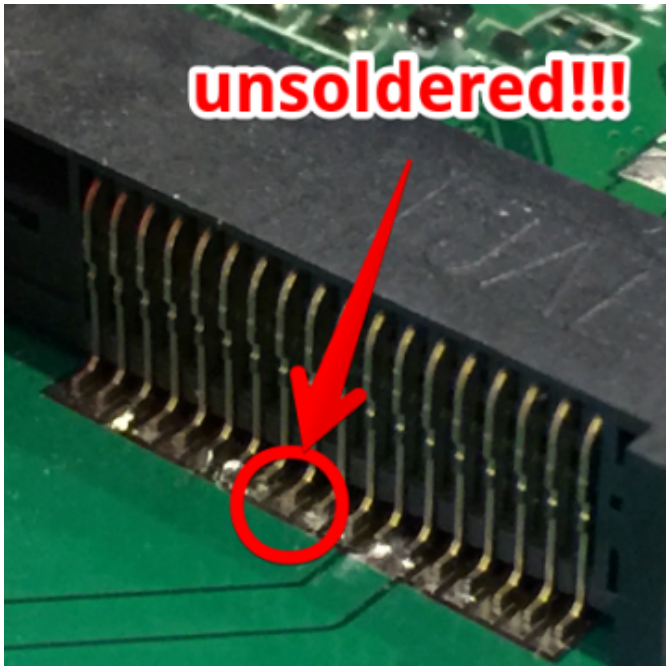
soldering

soldering



- it may work even if you leave a pin unsoldered on surface mounting
  - because the pin touches circuit
  - then, it will not work one day

soldering



- all the cause of failure, it is **impatience**
  - すべての失敗の原因、それは**焦り**

soldering

mruby and mruby/c

mruby and mruby/c

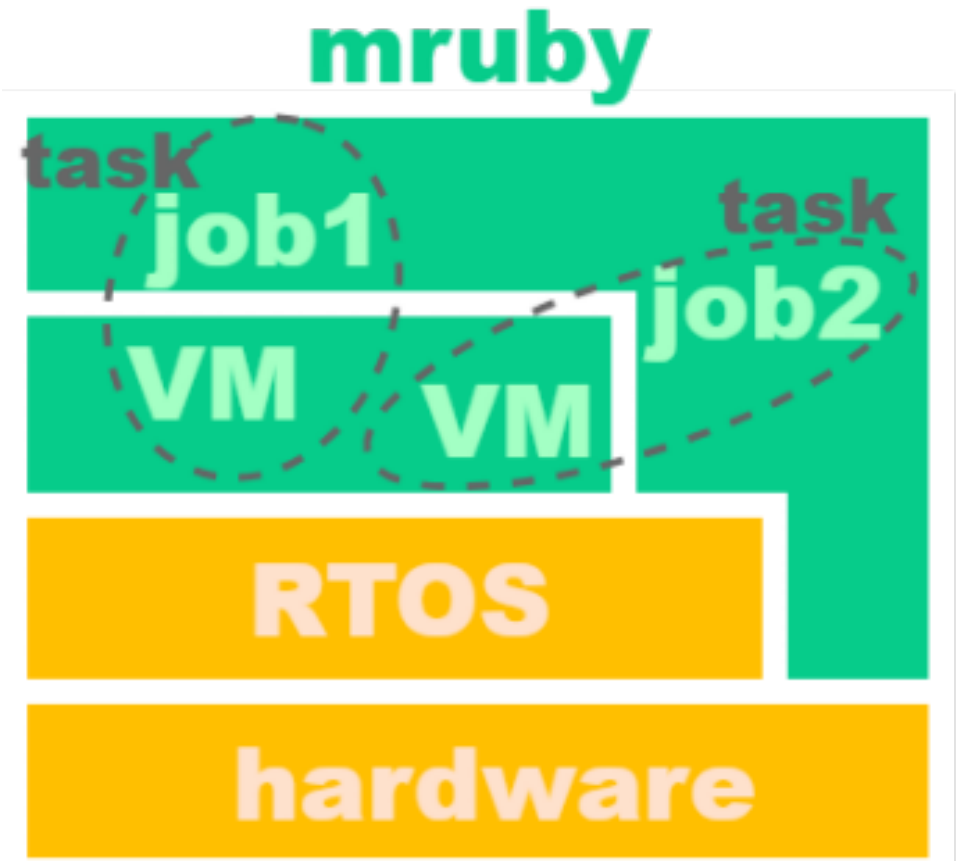
mruby

- an ecosystem of interpreter, compiler (**mrbc**), shell(mirb) and virtual machine and mrbgems

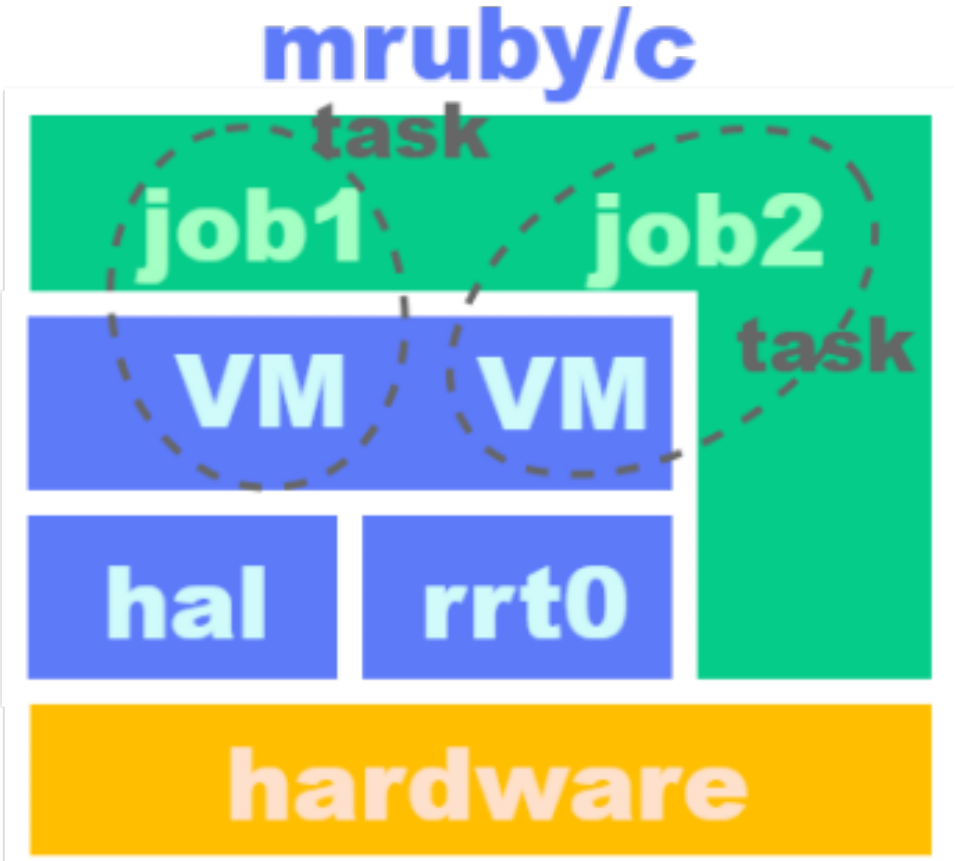
mruby/c

- VM(smaller than mruby), **rrt0** and hal (hardware abstraction layer)

mruby and mruby/c



RTOS manages VMs. you should learn about RTOS to use its features like concurrency



rrt0 manages VMs. so you can run multiple tasks without OS.
note: jobs are compiled with mrbc

mruby and mruby/c

mruby	mruby/c
v1.0.0 in Jan 2014	v1.0 in Jan 2017
mrbgems	no package manager
RAM < N*100KB	RAM < 64KB

coming up features to mruby/c

- task priority
  - at this point, tasks run as simple round robbin schedule, although we can use Mutex
- instance variable
  - at this point, you can use constants and globals

in fact, you can use these features above.
just not announced

coming up features to mruby/c

- compile option of including <math.h>
  - to reduce memory usage
- Array#each and Hash#each
  - and Hash#to_json will come
  - note that now you can compile `#each` with mrbc but it will not run on mruby/c's VM
    - in short, mruby/c VM has less methods than mruby VM

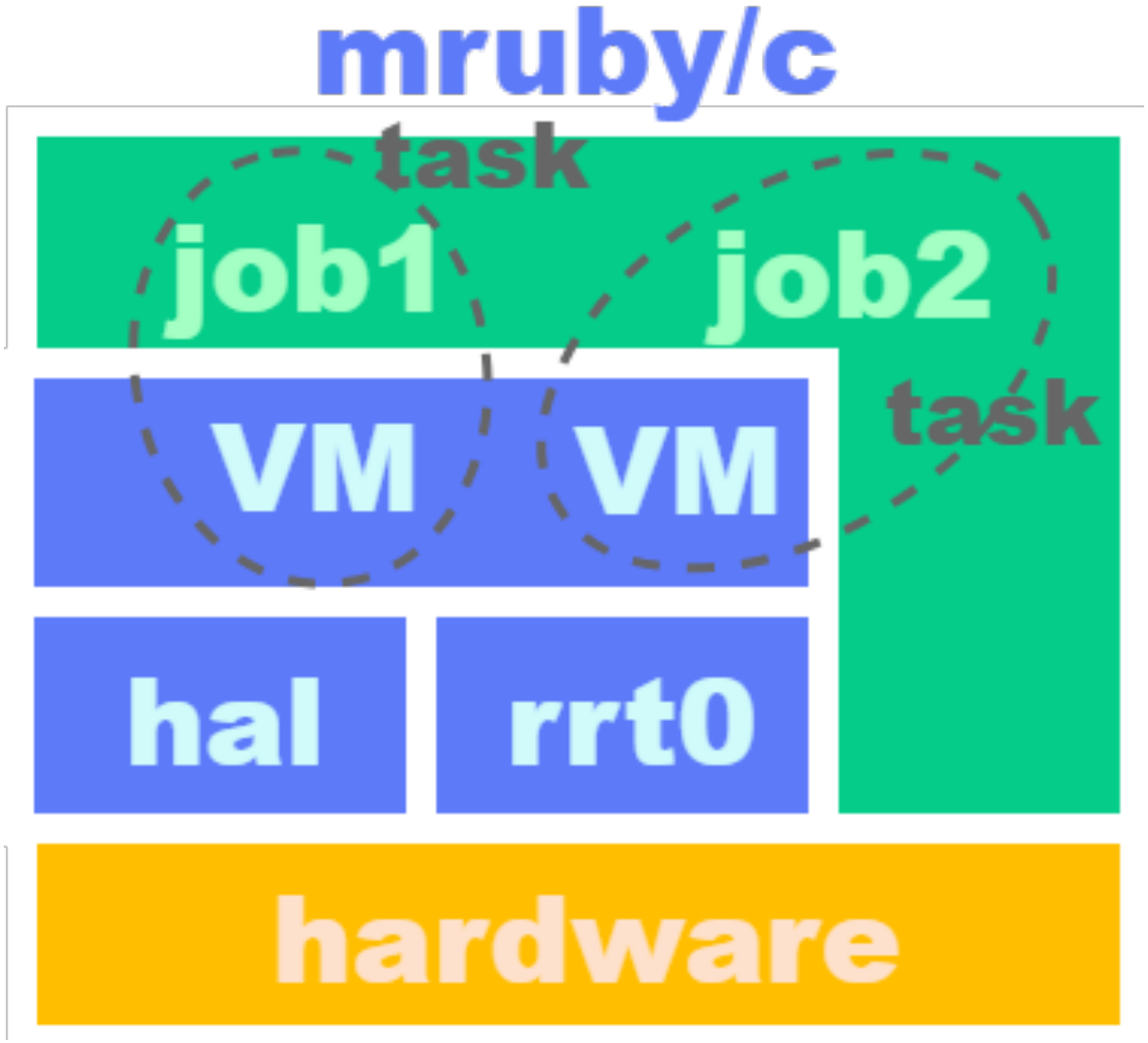
how does mruby/c work

how does mruby/c work

```
~/sample_project
├──── main.c
├──── mrblib
│       ├──── job_1.rb
│       └──── job_2.rb
└──── src
        ├──── job_1.c
        └──── job_2.c
```

- job_*.c are compliled code from job_*.rb

how does mruby/c work



how does mruby/c work

```c
/* main.c */
#include "src/job_1.c"
#include "src/job_2.c"
// use 30KB RAM for VMs in this case
#define MEMORY_SIZE (1024*30)
static uint8_t memory_pool[MEMORY_SIZE];
int main(void) {
  mrbc_init(memory_pool, MEMORY_SIZE);
  mrbc_create_task(job_1, 0);
  mrbc_create_task(job_2, 0);
  mrbc_run(); // 2 tasks run concurrently!
  return 0;
  // we will not write `main loop` in main.c
}
```

how does mruby/c work

- we can run easily multiple VMs with **concurrency** due to **rrt0**
- you might be disappointed to know you have to write C
  - yes, we have to write **main.c**
  - don't worry, it's almost boilerplate code

how does mruby/c work

```
~/mrubyc $ tree src -P *.h
src
├── alloc.h
├── c_array.h
...
├── console.h
├── errorcode.h
├── global.h
├── hal_posix
│   └── hal.h
├── hal_psoc5lp
│   └── hal.h
├── load.h
├── mrubyc.h
├── opcode.h
├── rrt0.h
├── static.h
├── symbol.h
├── value.h
├── vm.h
└── vm_config.h
```

how does mruby/c work

```
~/mrubyc $ tree src -P *.h
src
├── alloc.h
├── c_array.h
...
├── console.h
├── errorcode.h
├── global.h
├── hal_posix
│   └── hal.h # hal for posix
├── hal_psoc5lp
│   └── hal.h # hal for PSoC5LP
├── load.h
├── mrubyc.h
├── opcode.h
├── rrt0.h
├── static.h
├── symbol.h
├── value.h
├── vm.h
└── vm_config.h
```

how does mruby/c work

```
~/mrubyc $ tree src -P *.h
src
├── alloc.h
├── c_array.h
...
├── console.h
├── errorcode.h
├── global.h
├── hal_posix
│   └── hal.h
├── hal_psoc5lp
│   └── hal.h
├── load.h
├── mrubyc.h
├── opcode.h
├── rrt0.h      # runtime scheduler
├── static.h
├── symbol.h
├── value.h
├── vm.h
└── vm_config.h
```

how does mruby/c work

```
~/mrubyc $ tree src -P *.h
src
├── alloc.h
├── c_array.h
...
├── console.h
├── errorcode.h
├── global.h
├── hal_posix
│   └── hal.h
├── hal_psoc5lp
│   └── hal.h
├── load.h
├── mrubyc.h
├── opcode.h
├── rrt0.h
├── static.h
├── symbol.h
├── value.h # this gives you hints about variable
├── vm.h
└── vm_config.h
```

how does mruby/c work

```
~/mrubyc $ tree src -P *.h
src
├── alloc.h
├── c_array.h
...
├── console.h
├── errorcode.h
├── global.h
├── hal_posix
│   └── hal.h
├── hal_psoc5lp
│   └── hal.h
├── load.h
├── mrubyc.h
├── opcode.h
├── rrt0.h
├── static.h
├── symbol.h
├── value.h
├── vm.h
└── vm_config.h # edit this if needed
```

debugging

- we can neither do step execution nor look into memory to see mruby/c variables
- before writing app code, we should prepare way of debug
- let's go with **old-fashioned 'print debug'**. it'll be almost enough

debugging

```c
/* add this snippet into main.c */
// create serial console with UART for debug print
// http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_tutorial/737
static void c_debugprint(mrb_vm *vm, mrb_value *v, int argc){
  int total, used, free, fragment;
  mrbc_alloc_statistics(&total, &used, &free, &fragment);
  console_printf(
    "Memory total:%d, used:%d, free:%d, fragment:%d\n",
    total, used, free, fragment);
  unsigned char *key = GET_STRING_ARG(1);
  unsigned char *value = GET_STRING_ARG(2);
  console_printf("%s:%s\n", key, value );
}
int hal_write(int fd, const void *buf, int nbytes){
  UART_DEBUG_PutArray(buf, nbytes);
  return nbytes;
}
int hal_flush(int fd){
  return 0;
}
int main(void) {
  mrbc_define_method(0, mrbc_class_object, "debugprint", c_debugprint);
}
```

debugging

```
# mruby
pi = 3.14
debugprint('Pi', pi)

=> # print in serial console like 'PuTTY' connecting USB
   Memory total:30000, used:20000, free:10000, fragment:3
   Pi:3.14
```

actual source code

http://github.com/
hasumikin/rubykaigi2018_demo.cydsn

actual source code

```
~/rubykaigi2018_demo.cydsn
├──── main.c
├──── mrblib
│       ├──── job_3gmodule.rb  # transmit JSON data to cloud
│       ├──── job_breath.rb    # hit 3gmodule periodically to send data
│       ├──── job_co2.rb       # measure CO2 concentration
│       ├──── job_heartbeat.rb # show values on LED
│       └──── job_humidity_deficit.rb # measure humidity deficit(HD)
└──── src
        ├──── job_3gmodule.c
        ├──── job_breath.c
        ├──── job_co2.c
        ├──── job_heartbeat.c
        └──── job_humidity_deficit.c
```

actual source code

```
# job_heartbeat.rb
$mutex = Mutex.new # rrt0 has Mutex functionality
while true # this is a `main loop`
  r_LED_Driver_ClearDisplayAll # wrapper method of C func
  $co2 = measure_co2 # defined in job_co2.rb
  $hd = measure_humidity_deficit # job_humidity_deficit.rb
  $mutex.lock() # to prevent other jobs from overwriting LED
    r_LED_Driver_WriteString7Seg(sprintf('%4d', $co2), 0)
    r_LED_Driver_WriteString7Seg(sprintf('%3.0f', $hd * 10), 4)
    r_LED_Driver_PutDecimalPoint(1, 5)
    r_LED_Driver_WriteString7Seg('g', 7)
  $mutex.unlock()
  sleep 3
end
```

how does
r_LED_Driver_WriteString7Seg() work?

actual source code

```
/* add into main.c */
#include "src/job_heartbeat.c"
static void c_LED_Driver_WriteString7Seg(
    mrb_vm *vm, mrb_value *v, int argc){
  char *string = GET_STRING_ARG(1); // see value.h
  int position = GET_INT_ARG(2);
  // this is defined in IDE's framework
  LED_Driver_WriteString7Seg(string, position);
}
int main(void){
  ...
  mrbc_define_method(0, mrbc_class_object,
    "r_LED_Driver_WriteString7Seg",
    c_LED_Driver_WriteString7Seg);
  ...
}
```

actual source code

```
# job_breath.rb
$state = 'initializing'
while !$mutex # wait until main loop runs
  relinquish()
end
sleep 10 # wait for 3G module starts
$state = 'waiting' if check_3gmodule
while true
  if $state != 'sending'
    $state = 'ready_to_send'
  end
  sleep 300 # send data every 5 mins
end
```

actual source code

```
# job_3gmodule.rb(excerpt)
while true
  if $state == 'ready_to_send'
    $state = 'sending'
    i = 0
    while true
      flag = send_data(json($co2, $humidity_deficit))
      if flag
        $mutex.lock()
        r_LED_Driver_WriteString7Seg('sendgood', 0)
        sleep 1
        $mutex.unlock()
        break
      else
        $mutex.lock()
        r_LED_Driver_WriteString7Seg('sendfail', 0)
        sleep 1
        r_LED_Driver_WriteString7Seg('retry', 0)
        sleep 1
        $mutex.unlock()
      end
      if i > 2
        j = 0
        $mutex.lock()
        while j < 20
          j += 1
          r_LED_Driver_WriteString7Seg('fatalerr', 0)
          sleep 0.2
          r_LED_Driver_ClearDisplayAll
          sleep 0.05
          break
        end
        $mutex.unlock()
      end
      i += 1
    end
    debugprint('memory', 'check')
    $state = 'waiting'
ennd
```

actual source code

```
# job_co2.rb does not have loop
def measure_co2
  r_UART_CO2_ClearTxBuffer
  r_UART_CO2_ClearRxBuffer
  ary = [0xff,0x01,0x86,0x00,0x00,0x00,0x00,0x00,0x79]
  r_UART_CO2_PutArray(ary, ary.size) # this will be explained
  res = []
  i = 0
  # can't write `for i in 0..3`, equiv. of #each
  while i < 4
    res[i] = r_UART_CO2_GetByte
    i += 1
  end
  if res[0] == 255 && res[1] == 134
    return res[2] * 256 + res[3]
  else
    return false
ennd
```

actual source code

```
/* add into main.c */
// mruby array should be converted into C array
static void c_UART_CO2_PutArray(
    mrb_vm *vm, mrb_value *v, int argc){
  mrb_value mrbc_array = GET_ARY_ARG(1);
  uint8 array[GET_INT_ARG(2)];
  for( int i = 0; i < GET_INT_ARG(2); i++ ) {
    mrb_value value = mrbc_array_get(&mrbc_array, i);
    array[i] = value.i;
  }
  UART_CO2_PutArray(array, GET_INT_ARG(2));

  uint8 tmpStat;
  do { // will be explained later
    tmpStat = UART_CO2_ReadTxStatus();
  } while (~tmpStat & UART_CO2_TX_STS_COMPLETE);
}
```

actual source code

- you can pass string instead of mruby/c array
- it was a sample to handle mrb_value
  - I also wanted argument class(Array) to correspond with method name (xxx_PutArray)
- use string instead of array if memory becomes short

actual source code

```
/* add into main.c */
// mruby array should be converted into C array
static void c_UART_CO2_PutArray(
    mrb_vm *vm, mrb_value *v, int argc){
  mrb_value mrbc_array = GET_ARY_ARG(1);
  uint8 array[GET_INT_ARG(2)];
  for( int i = 0; i < GET_INT_ARG(2); i++ ) {
    mrb_value value = mrbc_array_get(&mrbc_array, i);
    array[i] = value.i;
  }
  UART_CO2_PutArray(array, GET_INT_ARG(2));

  uint8 tmpStat;
  do { // communication timing control here
    tmpStat = UART_CO2_ReadTxStatus();
  } while (~tmpStat & UART_CO2_TX_STS_COMPLETE);
}
```

actual source code

- communication of microcontroller takes time
- we will not get notification something like callback from peripheral
- we have to wait until it's ready
- we can write another wrapper for waiting
- but we want to reduce memory usage

development environment

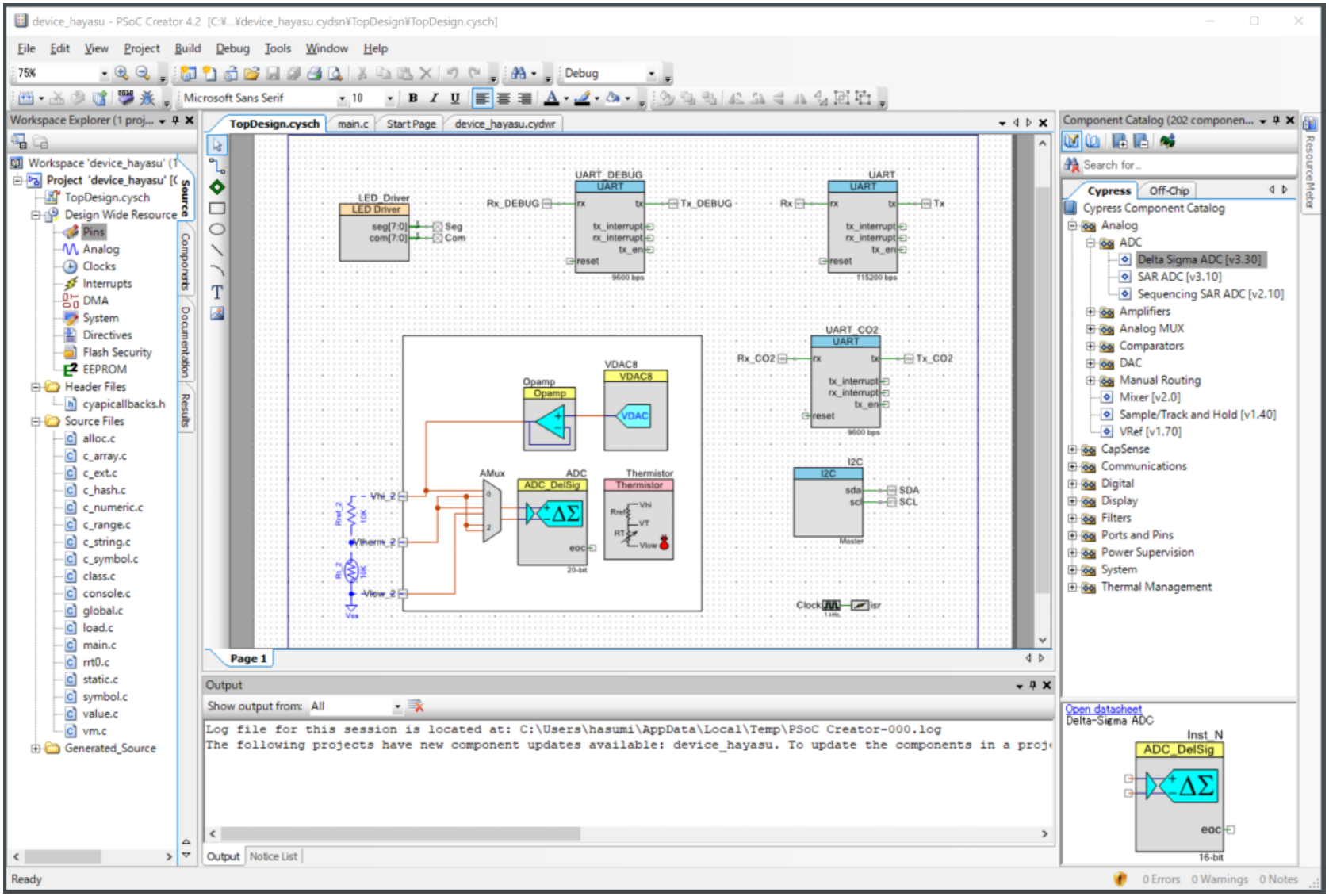
development environment

- IDE `PSoC Creator`
  - you need to use it for periferal arrangement and pin assignment

development environment

- IDE `PSoC Creator`
  - you need to use it for periferal arrangement and pin assignment
  - runs **only on Windows**

development environment



development environment

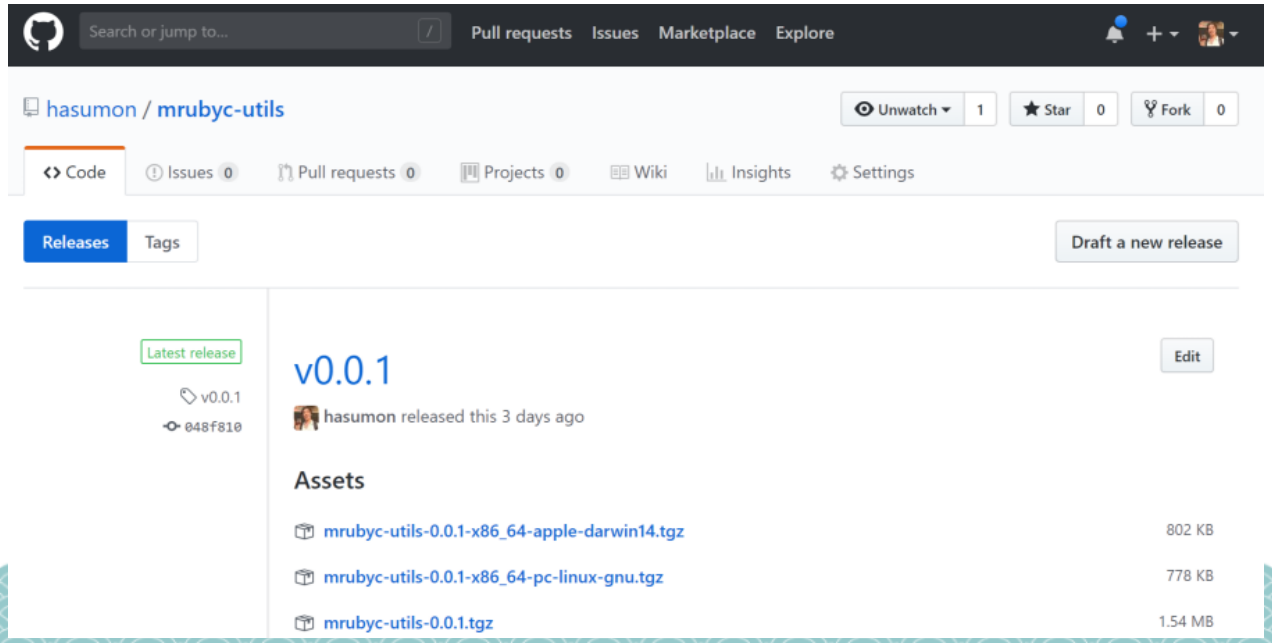
- do you hate IDE?

development environment

- do you hate IDE?
  - **#MeeToo**

development environment

- do you hate IDE?
  - **#MeeToo**
- I made a tool for Linux and macOS
  - github.com/hasumikin/**mrubyc-utils**



development environment

```
your_project $ mrubyc-utils --help
Usage: mrubyc-utils COMMAND [ARGS]

 install         Install mruby/c repo into your local and setup templates.
                 Please run this command at the top directory
                 of your project (normally it should have 'main.c').
 update          Update mruby/c repo to the newest master branch.
 checkout        Checkout specified tag or commit of mruby/c repo.
  -t | --tag       [required] You can specify anything that
                   `git checkout` will accept.
 tag             Show all tags of mruby/c repogitory that you installed.
 classes         Show all the classes that are defined in
                 mruby/c's virtual machine.
 methods         Show all the methods that are available
                 in specified class of mruby/c.
  -c | --class     [required] You have to specify class name
 compile         Compile your mruby source into C byte code.
  -w | --watch     [optional] Monitoring loop runs and it will
                   compile mruby source every time you save.
```

development environment

```
your_project $ mrubyc-utils install
(...install mruby/c and template files)
your_project $ mrubyc-utils compile
(...you can specify --watch option)
your_project $ tree
 ├── .gitignore
 ├── .mrubyc/
 ├── .mrubycconfig
 ├── main.c
 ├── mrblib
 │       ├── job_main_loop.rb
 │       ├── job_operations.rb
 │       └── job_sub_loop.rb
 ├── mrubyc_src
(...)
 │       └── vm_config.h
 └── src
         ├── job_main_loop.c
         ├── job_operations.c
         └── job_sub_loop.c
```

development environment

```
your_project $ mrubyc-utils classes
- Array
- False
- Fixnum
- Float
- Hash
- Mutex
- Nil
- Object
- Proc
- Range
- String
- Symbol
- True
```

development environment

```
your_project $ mrubyc-utils methods --class=array
Array
- +
- <<
- []
- []=
- at
- clear
- count
- delete_at
- dup
- empty?
- first
- index
- last
- length
- pop
- push
- shift
- size
- unshift
  < Object
  - !
  - !=
  - <=>
  - attr_accessor
  - attr_reader
  - change_priority
  - class
  - get_tcb
  - instance_methods
  - new
  - p
  - puts
  - relinquish
  - resume_task
  - sleep
  - sleep_ms
  - sprintf
  - suspend_task
```

development environment

- using mrubyc-utils, you can minimize uses of IDE to these:
  - build setting
  - peripheral arrangement and pin assignment
  - build
- then you can write app code with vim, emacs or textbringer

what you can do for mruby/c

what you can do for mruby/c

- write another hal
  - so that mruby/c can be suitable for more microcontrollers
- write documentation, test and real applications
- publish that you use mruby/c
- posix...

demo (this was added after Kaigi)

demo (this was added after Kaigi)

- CO2 kept around 700ppm while I was talking!
- ventilating facilities of Sendai International Center are so goooood!

about me

- HASUMI Hitoshi（羽角 均）
- Monstar Lab（モンスター･ラボ）
  - Matsue office（島根開発拠点）

conclusion

conclusion

- you should refresh air
  - 換気大事

thank you!


23BY
加水火入
12
%
生酛純米
旭日
じゅうじあさひ
純米吟醸酒
旭日
じゅうじあさひ
生酛純米
旭日
旭日酒造有限会社
純米酒
旭日