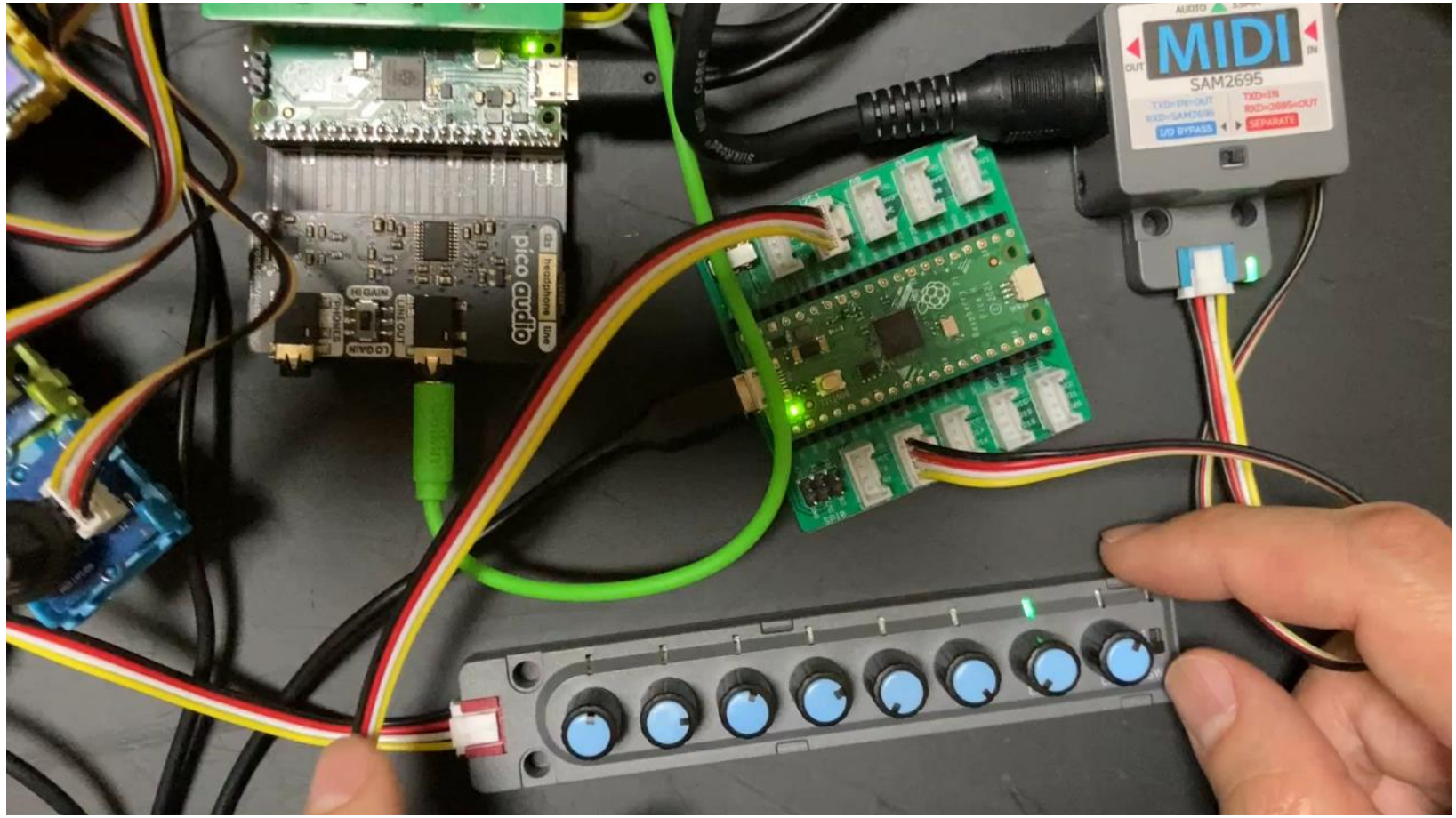
Making a MIDI controller device with PicoRuby/R2P2

RubyKaigi 2025 LT

Ryo Ishigaki

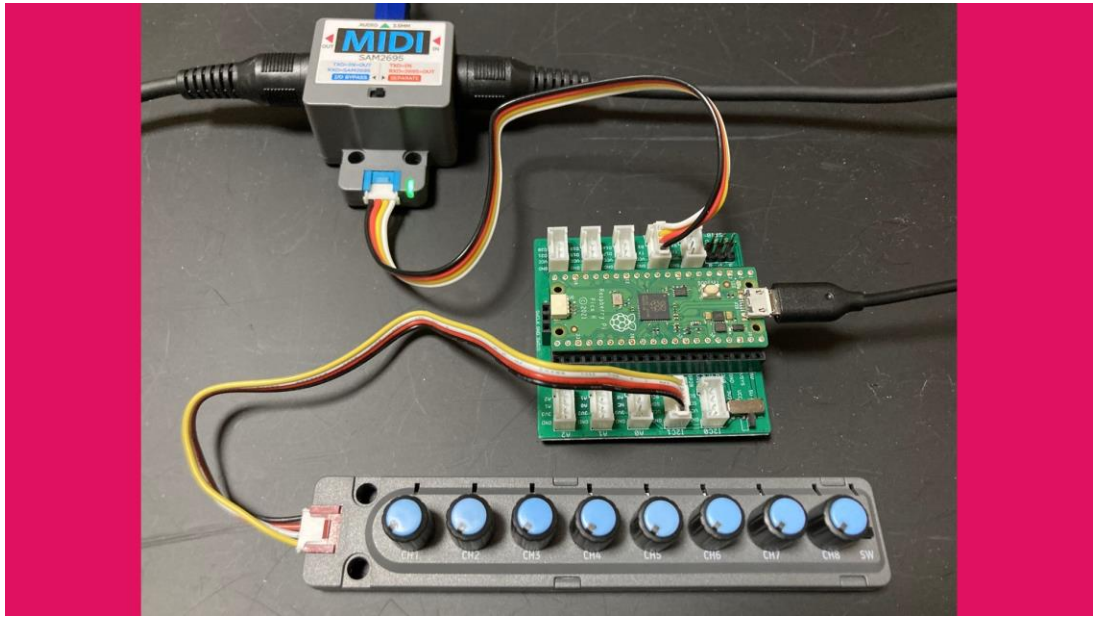
MIDI Controller PRMC-1 (type-0) Demo



https://youtu.be/Eq25Ze3kTH4

Thank you for listening!

MIDI controller PRMC-1 (type-0)



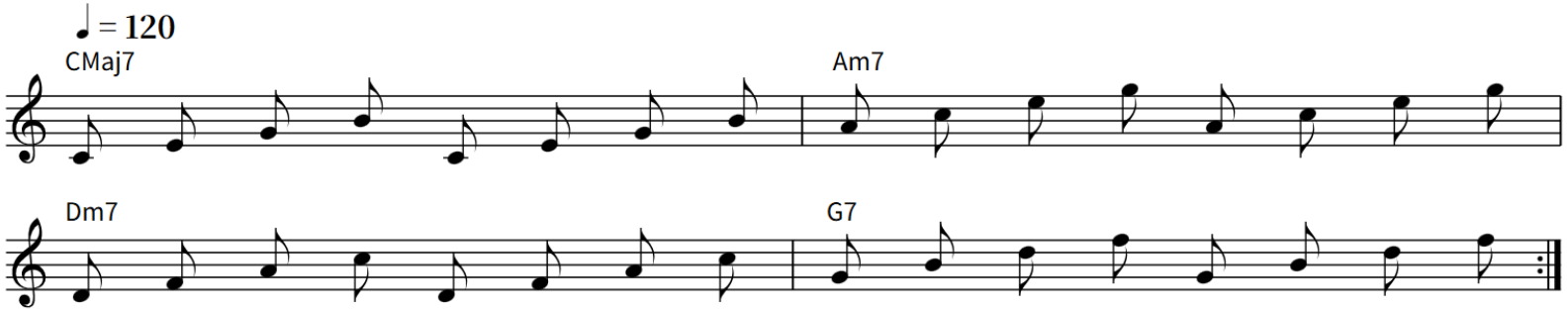
- **PRMC-1** is a **MIDI** Controller using **PicoRuby/R2P2**
  - https://github.com/risgk/midi-controller-prmc-1
- Made to control synthesizers in electronic music performances
  - PRMC-1 transmits MIDI messages
- Hardware (no soldering)
  - **Raspberry Pi Pico**
  - Grove Shield for Pi Pico
  - M5Stack Unit 8Angle
  - M5Stack Unit MIDI
- All code is written in **Ruby**!

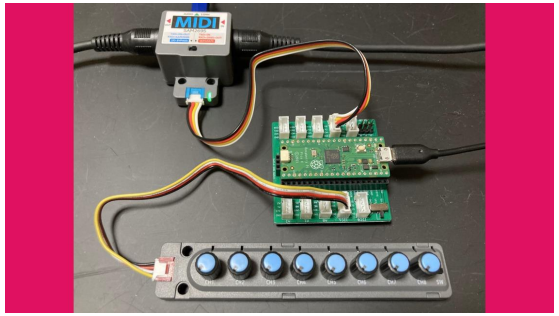
MIDI (Musical Instrument Digital Interface)

- Examples of MIDI messages
  - **Note On/Off**: Turning on/off a note
  - **Control Change**: Modifying sounds (or adjusting parameters)
  - **Program Change**: Changing tones
  - **Clock**: Synchronizing the tempo of devices (24 clocks per quarter note)
  - **Start/Stop**: Controlling playback
- MIDI receivers and transmitters use **UART** (Universal Asynchronous Receiver/Transmitter) at 31.25 kbaud
- For details, see the specifications
  - MIDI 1.0 Detailed Specification https://midi.org/midi-1-0-detailed-specification
  - MIDI1.0規格書 https://amei.or.jp/midistandardcommittee/MIDIspcj.html

Specifications of PRMC-1 (type-0)

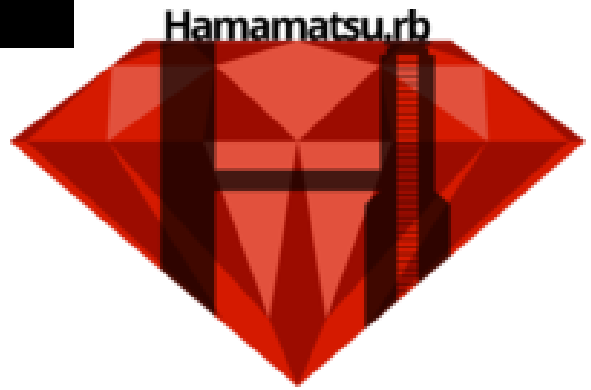
- Simple functions
  - 4-step **chord sequencer** for playing chords as arpeggios
  - Adjusting Brightness (Cutoff) and Harmonic Content (Resonance)
  - Unit MIDI has a built-in synthesizer chip (SAM2695), so PRMC-1 can also produce sound directly
- Usage
  - CH1 – CH4 Knob: Root of Step 1 - 4 Chord
  - CH5 Knob: Arpeggio Pattern, 1 - 6
  - CH6 Knob: Brightness
  - CH7 Knob: Harmonic Content
  - CH8 Knob: BPM
  - SW Switch: Start/Stop Sequencer





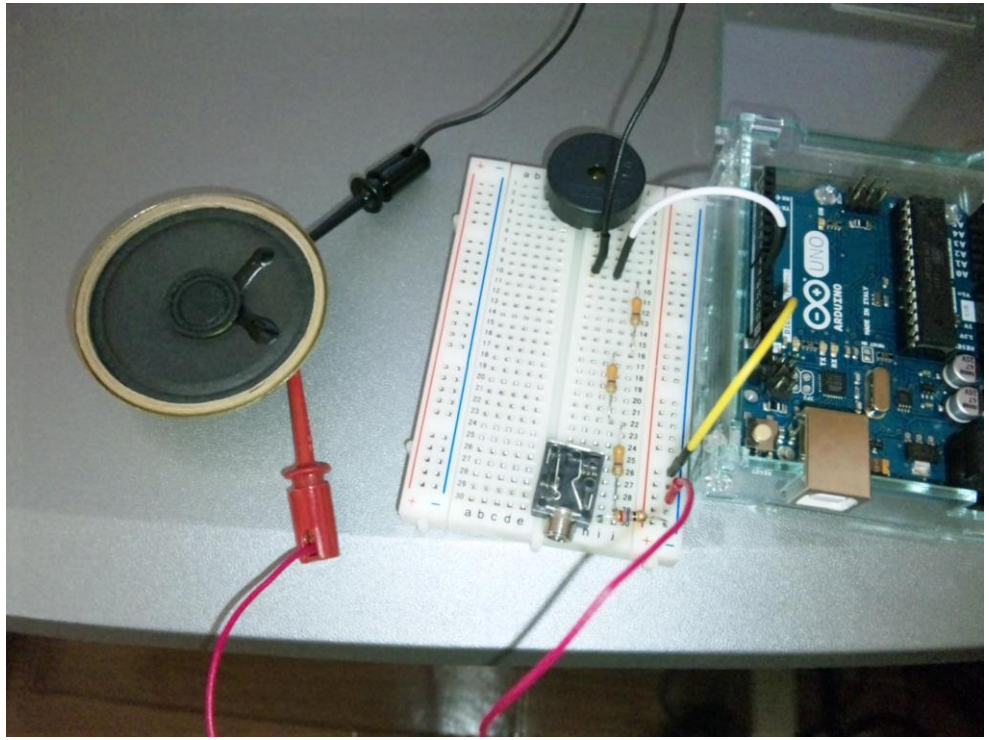
About me

- **Ryo Ishigaki** (石垣 良) @risgk
- Member of **Hamamatsu.rb** (called “Hamamatsu Ruby”)
  - Hamamatsu is the physical birthplace of Ruby
- Embedded software engineer
  - Experience: projector, milling machine, audio product, etc.
- Making synthesizers for microcontrollers using C/C++ as a hobby

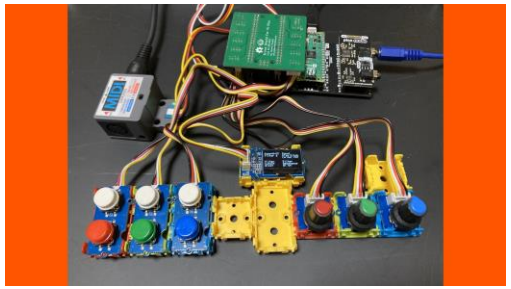


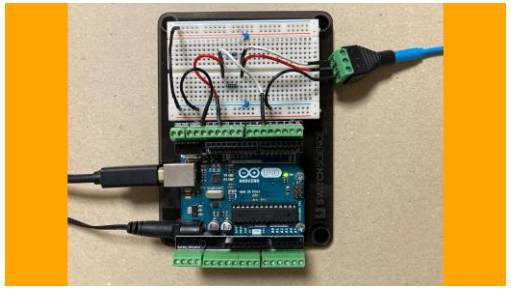
Making synthesizers

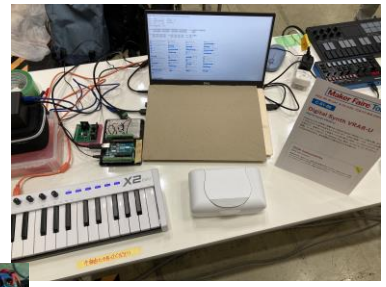
- Hamamatsu is famous for its musical instrument industry
- One of the reasons I started making synthesizers is that I moved to Hamamatsu
- About 11 years ago, my first synthesizer **VRA8** was prototyped with Ruby and ported it to **Arduino Uno**
  - Details: Ruby x Arduinoでシンセサイザーを作ってみた (浜松Ruby会議01) https://gist.github.com/risgk/0db52ea683530652d933

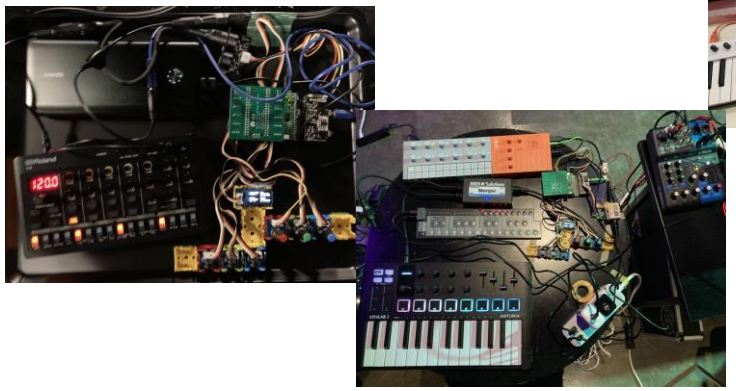


Making and playing synthesizers





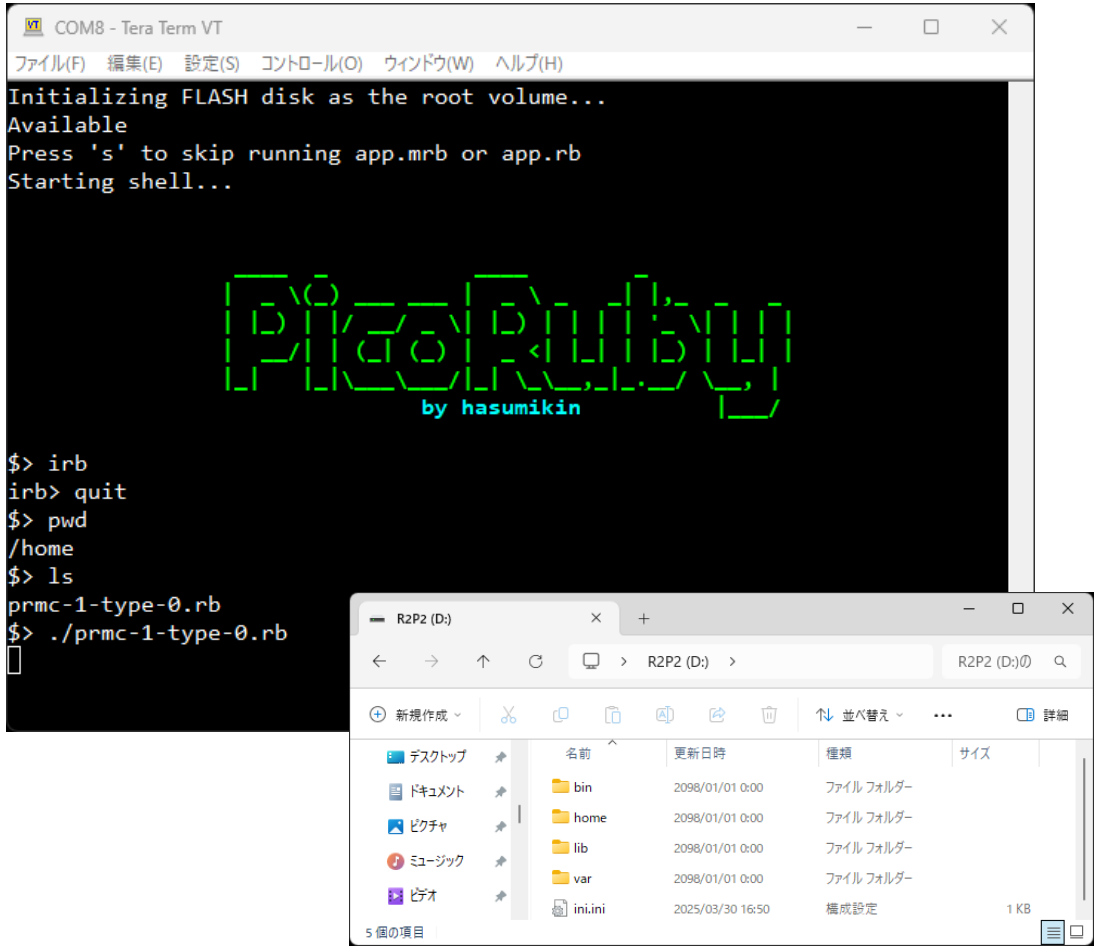




- As **ISGK Instruments**, I make about one synthesizer a year and exhibit it at Maker Faire and other events
  - Latest work: **PRA32-U2** for Raspberry Pi Pico 2 https://github.com/risgk/digital-synth-pra32-u2
- I wrote articles on synthesizer making for a magazine and a book
  - Book: ラズベリー・パイPico/Pico W攻略本 https://interface.cqpub.co.jp/magazine/2023pico/
- I am challenging electronic music performance using hardware synthesizer, called “machine live”
  - I will be at シンセカイリアル (Synthesizer Meeting Real) on April 28th https://x.com/Yasushi_K/status/1910670512419738018

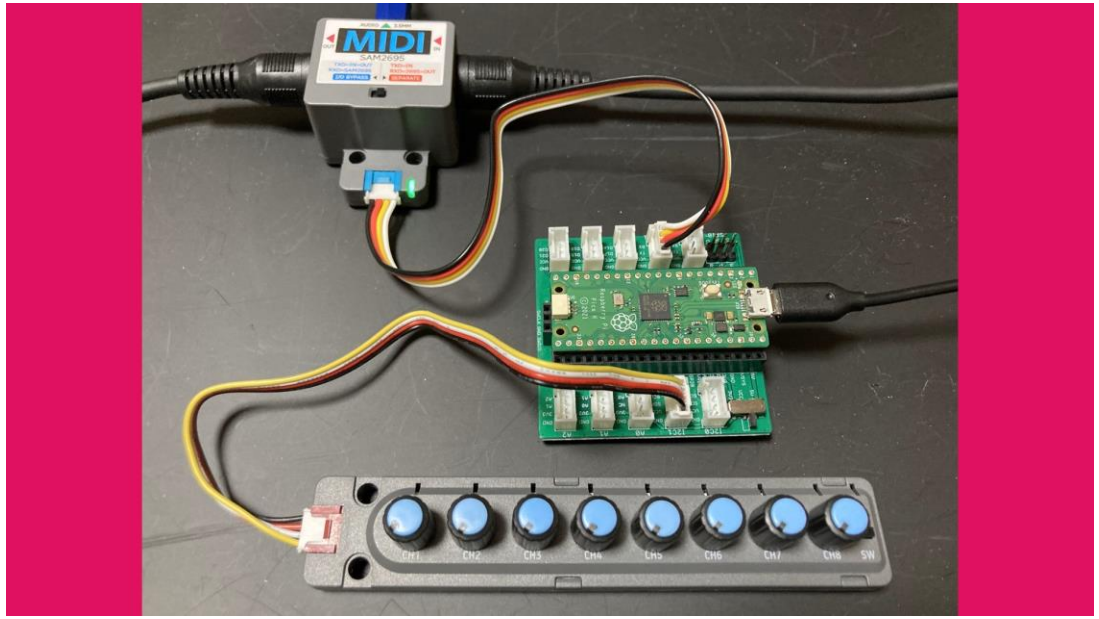
PicoRuby and R2P2

- **PicoRuby** is a Ruby implementation for microcontrollers
  - https://github.com/picoruby/picoruby
- PicoRuby's VM is **mruby/c**
- **R2P2** (Ruby Rapid Portable Platform) is a PicoRuby shell for Raspberry Pi Pico and others
  - https://github.com/picoruby/R2P2
- Usage of R2P2
  1. Drag and drop a release binary into the RPI-RP2 drive of Pi Pico
  2. Connect the shell through a serial port with a terminal emulator
  3. /home/app.rb is automatically run at startup



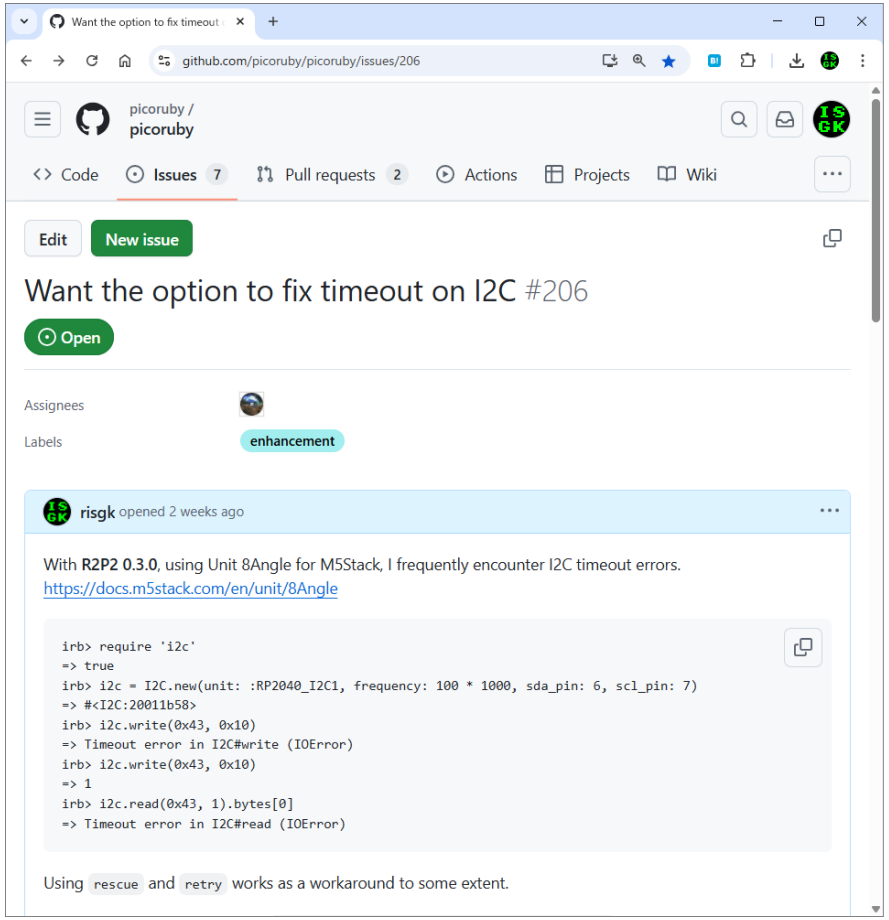
Functional Units of M5Stack

- Functional Units of M5Stack are compatible with **Grove** interface
- So, most of these should work with Raspberry Pi Pico
  - However, as far as I have briefly tested, M5Stack Unit **8Encoder** did not work well for some reason...



Timeout error in I2C

- When accessing M5Stack Unit **8Angle** in R2P2 environment, **I2C** (Inter-Integrated Circuit) timeout errors occurred frequently
- Workaround
  - Use only 1-byte registers
  - Reduce communication speed to **20 kbps** (usually 100 or 400 kbps)
  - If an error occurred, **retry**
- I would like PicoRuby to have an option to adjust the timeout (like Arduino)
  - Want the option to fix timeout on I2C #206 https://github.com/picoruby/picoruby/issues/206



prmc-1-type-0.rb (about 400 lines)

```
require 'i2c'
require 'uart'

# options
MIDI_CHANNEL = 1
…
class M5UnitAngle8
…
end

class MIDI
…
end
```

```
class PRMC1Core
…
end

# setup
…

# loop
loop do
…
end
```

class M5UnitAngle8

```
def prepare_to_get_analog_input(ch)
  @i2c.write(ANGLE8_I2C_ADDR, ANGLE8_ANALOG_INPUT_8B_REG + ch)
rescue
  retry
end

def get_analog_input
  @i2c.read(ANGLE8_I2C_ADDR, 1).bytes[0]
rescue
  retry
end
```

class MIDI

```
  def send_note_on(note_number, velocity, channel)
    @uart.write((0x90 + channel - 1).chr + note_number.chr + velocity.chr)
  end

  def send_note_off(note_number, velocity, channel)
    @uart.write((0x80 + channel - 1).chr + note_number.chr + velocity.chr)
  end
...
  def send_clock
    @uart.write(0xF8.chr)
  end
```

class PRMC1Core (1)

```
  def change_parameter(key, value)
    case key
...
    when 4
      arpeggio_pattern = (value * (6 - 1) * 2 + 127) / 254 + 1

      case arpeggio_pattern
      when 1
        @arpeggio_intervals_candidate = [1, 3, 5, 7, 1, 3, 5, 7]
        @step_division_candidate = 8
      when 2
        @arpeggio_intervals_candidate = [1, 3, 5, 7, 5, 3, 1, 3]
        @step_division_candidate = 8
```

class PRMC1Core (2)

```
  def process_sequencer
    if @playing
      usec = Time.now.usec
…
        receive_midi_clock
…
    end
  end
…
  def receive_midi_clock
    @midi.send_clock
…
      @midi.send_note_on(@playing_note, NOTE_ON_VELOCITY, @midi_channel) if @playing_note != -1
…
      @midi.send_note_off(playing_note_old, NOTE_OFF_VELOCITY, @midi_channel) if playing_note_old != -1
```

setup

```
led_builtin = GPIO.new(25, GPIO::OUT)
led_builtin.write(1)
i2c1 = I2C.new(unit: :RP2040_I2C1, frequency: 20_000, sda_pin: 6, scl_pin: 7)
angle8 = M5UnitAngle8.new(i2c: i2c1)
uart1 = UART.new(unit: :RP2040_UART1, txd_pin: 4, rxd_pin: 5, baudrate: 31_250)
midi = MIDI.new(uart: uart1)
prmc_1_core = PRMC1Core.new(midi: midi, midi_channel: MIDI_CHANNEL)
current_inputs = []
```

loop

```
(0..7).each do |ch|
  prmc_1_core.process_sequencer
  angle8.prepare_to_get_analog_input(ch)
  prmc_1_core.process_sequencer
  analog_input = angle8.get_analog_input

  if current_inputs[ch].nil? ||
     analog_input > current_inputs[ch] + 1 ||
     analog_input < current_inputs[ch] - 1
    current_inputs[ch] = analog_input
    prmc_1_core.change_parameter(ch, 127 - current_inputs[ch] / 2)
  end
end
```

After this

- I will continue to evolve **PRMC-1**
- **Ruby** code is easily modified, so **PRMC-1 (type-N)** may be made for each performance

Would you like to create your own musical device?

Enjoy electronics!

Finally... Live performance!