mruby/c 評価ボード 02
利用説明書

2020/10/20 rev 1.0.0
しまねソフト研究開発センター

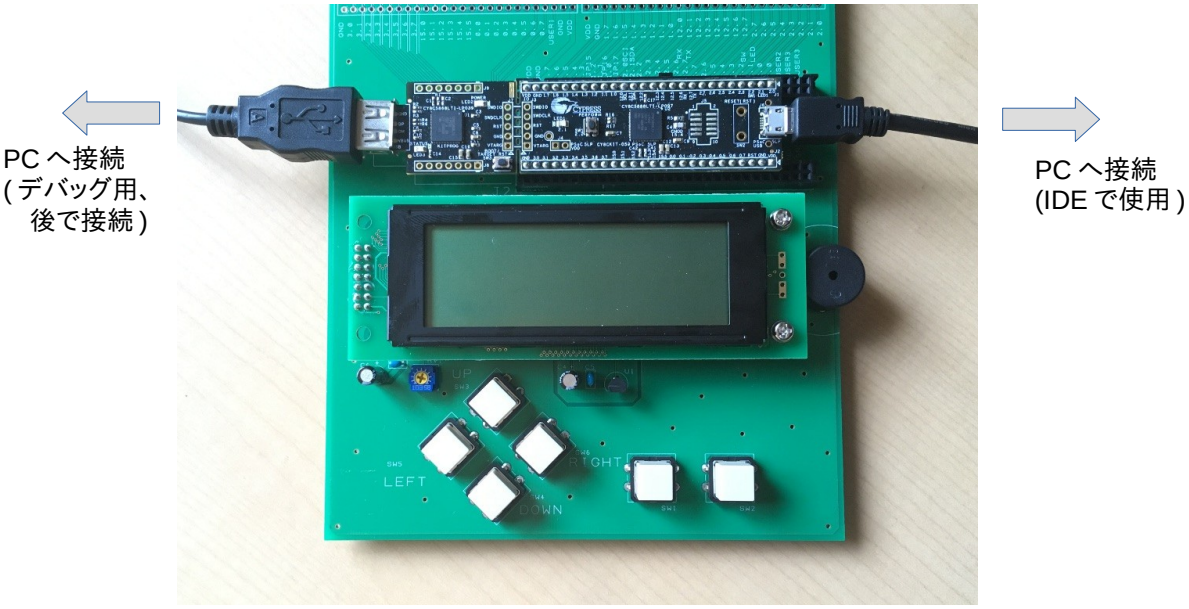
ピンアサイン

ポート名	タイプ	デバイス割り当て	（参考） CY8CKIT-059 上のデバイス PSoC5 内蔵デバイス
P0-0	D-in	SW1	OpAmp2.vout
P0-1	D-in	SW2	OPAmp0.vout
P0-2	D-in	SW3 UP	C12 1u OpAmp0.v+, SAR1.ext
P0-3	D-in	SW4 DOWN	C13 1u OpAmp0.v-, DSM0.ext1
P0-4	D-in	SW5 LEFT	C9 1u OpAmp2.v+, SAR0.ext
P0-5	D-in	SW6 RIGHT	OpAmp2.v-
P0-6	D-Out	Grove GPIO D1 / PWM1	VIDAC0.iout
P0-7	D-I/O	Grove GPIO D2	VIDAC2.iout
P1-0		(N/C)	PROG_SWDIO
P1-1		(N/C)	PROG_SWDCLK
P1-2		SPI MOSI	
P1-3		SPI MISO	P_SWO
P1-4		SPI SCLK	P_TDI JTAG[0] tdi
P1-5		SPI SS	JTAG[0] ntrst
P1-6			
P1-7			
P2-0			
P2-1	D-out	(on-board LED1)	LED1
P2-2	D-in	(on-board SW1)	SW1
P2-3	D-I/O	Grove GPIO D3	TRACE.CLK
P2-4	D-I/O	Grove GPIO D4	TRACE0.D0
P2-5	D-I/O	Grove GPIO D5	TRACE0.D1
P2-6			TRACE0.D2
P2-7			TRACE0.D3
P3-0	A-in	U1 ThermoSensor TI LM60BIZ	VIDAC1.iout
P3-1	A-in	Grove Analog A1	VIDAC3.iout
P3-2			C7 1u OpAmp3.v-, DSM0.ext2
P3-3	A-in	Grove Analog A2	OpAmp3.v+
P3-4			OpAmp1.v-
P3-5	A-in	Grove Analog A3	OpAmp1.v+
P3-6		Piezoelectric Speaker (PWM0)	OpAmp1.vout
P3-7		(SP1-)	OpAmp3.vout
P12-0	D-I/O	Grove I2C SCL	I2C1.scl
P12-1	D-I/O	Grove I2C SDA	I2C1.sda
P12-2	D-in	Grove UART1 RxD	SIO
P12-3	D-out	Grove UART1 TxD	SIO
P12-4	D-in	Grove UART2 RxD	I2C0.scl
P12-5	D-out	Grove UART2 TxD	I2C0.sda
P12-6	D-in	--------	SIO RX (プログラマ経由ホストシリアル)
P12-7	D-out	--------	SIO TX (プログラマ経由ホストシリアル)
P15-0	D-out	LCD DB4 (SC2004CS)	XTAL.xo
P15-1	D-out	LCD DB5	XTAL.xi
P15-2	D-out	LCD DB6	C42 22p XTAL32k.xo
P15-3	D-out	LCD DB7	C41 22p XTAL32k.xi
P15-4	D-out	LCD E	C4 2200p
P15-5	D-out	LCD RS	
P15-6	USB	--------	USBIO dp (オンボード USB マイクロコネクタ用)
P15-7	USB	--------	USBIO dm (オンボード USB マイクロコネクタ用)

mruby/c IDE の利用

提供された機能のみを使ってアプリケーションを構築する方法です。
統合開発環境、IDE を利用します。
あらかじめファームが書き込まれた本体を使用します。

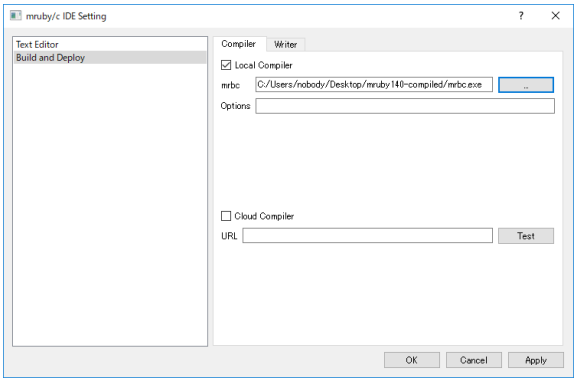
接続

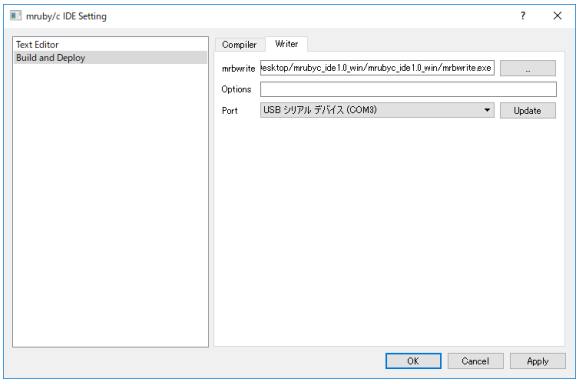


IDE 起動と設定

Windows の例で説明しますが、Mac や UNIX も同等です。

1. mrubyc-ide.exe を実行します。
2. File > Settings を選び、設定ダイアログを表示します。
3. 左ペインの Build and Deploy をクリックします。
4. 右ペインの Compiler タブで、LocalCompiler にチェックが入っていることを確認します
5. mrbc 欄の[...]ボタンをクリックし、mrbc.exe のパスを指定します。
6. 右ペインの Writer タブに移り、同様に mrbwrite.exe のパスを指定します。
7. Port 欄に、基板右サイドのマイクロ USB 端子側の認識ポートを指定します。



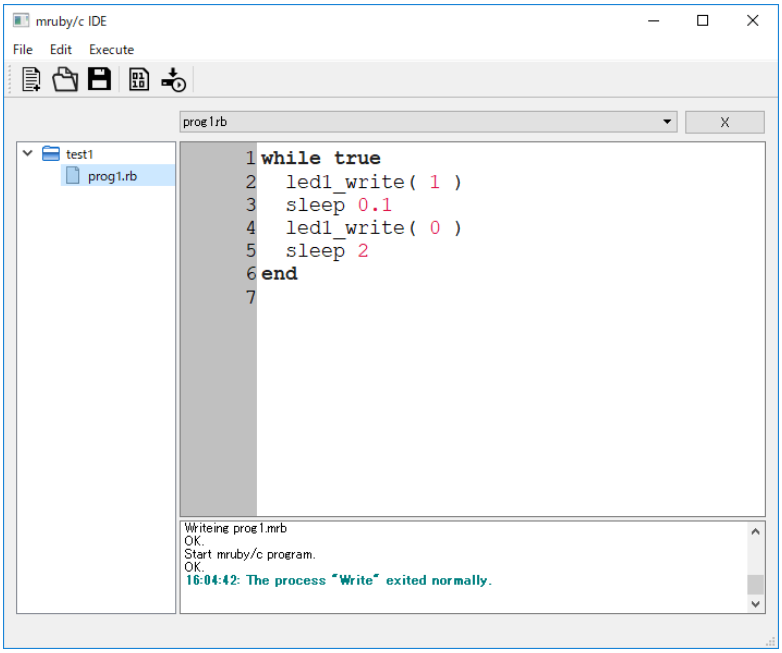


デバグ用コンソールの起動（任意）

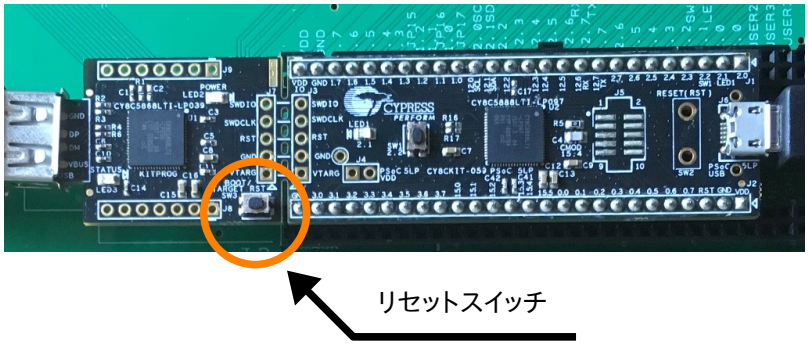
基板左サイドの USB 端子から、デバッグ用メッセージが表示されますので、開発時は接続しておくと便利です。
任意のターミナルソフトを利用して、ボーレート 57600bps で接続します。

プログラムの編集と書き込み

1. File > New > Project を選び、Setup New Project ダイアログを表示します。
2. 任意のプロジェクト名と、保存場所を指定します。
3. File > New > File を選び、Setup Add New File ダイアログを表示します。
4. 任意のファイル名を指定します。
5. 右上のプログラムペインで、Ruby プログラムを編集します。



6. 編集が終わったら、Execute > Write を選びます。
7. プログラムの書き込み待ち状態になるので、基板上のリセットスイッチを押下します。
8. プログラムが書き込まれ、動作開始します。



利用できる機能

オンボードスイッチ

マイコンドータ基板上のプッシュスイッチです。SW1 のシルクがあります。
負論理で、0（押された）、1（押されていない）が、返ります。

```
sw1 = sw1_read()
```

LED

マイコンドータ基板上の青色 LED です。LED1 のシルクがあります。
正論理で、1（点灯）、0（消灯）です。

```
led1_write(1)
```

キャラクタ LCD 表示器

4 行×20 文字の LCD パネルです。
アルファベットと数字、記号などが表示できます。詳しくは、HD44780 のデータシート等を参照してください。

表示位置の指定

```
lcd_location( row, column )
```

1 文字表示

```
lcd_putc( char_code )
```

文字列の表示

```
lcd_puts( "String" )
```

全画面消去

```
lcd_clear()
```

キーパッド

メイン基板上の 6 個のタクトスイッチです。
ビットマップで全てのスイッチの押下状態を同時に確認できます。
負論理で、ビットは基板上シルクの SWx を参照してください。

```
key = keypad_read()
```

温度センサー

メイン基板上に、温度センサー LM60 が搭載されており、温度の測定ができます。
詳細は、ADC の項を参照してください。

GPIO

汎用のデジタル入出力です。Grove コネクタに D1 から D5 としてアサインしています。
D1 は出力専用で、PWM1 と共用です。

コンストラクタ

GPIO.new(n)

例

```
gpio1 = GPIO.new( 1 )    # param: grove pin number. 1 to 5.
```

入出力方向設定

setmode(mode)

コンストラクタで指定したピンを、入力とするか出力とするかを指定します。
例

```
gpio1.setmode( GPIO::IN )      # GPIO::IN or GPIO::OUT
```

出力

write(value)

例

```
gpio1.write( 1 )
```

入力

read() -> Integer

例

```
val = gpio1.read()             # val = 0 or 1
```

ADC

汎用の電圧入力です。
A/D コンバータ仕様

- 12 ビット
- 0V - 2.048V
- 100000sps

コンストラクタ

ADC.new(n)

例

```
adc1 = ADC.new( 1 )      # param: grove pin number.
```

入力

read() -> Float

例

```
val = adc1.read()        # val = 0.0 - 2.048
```

温度センサー

メイン基板上に、温度センサー LM60 が搭載されており、温度の測定ができます。
LM60 は、アナログ値を出力します。本ボードでは、ADC のチャンネル 0 へ接続されています。

例

```
adc0 = ADC.new( 0 )
v = adc0.read()
t = (v - 424e-3) / 6.25e-3
```

シリアル通信　UART

UART シリアルインターフェースを扱います。
仕様

- ボーレート UART1 19200bps，UART2 9600bps
- 8bit，パリティーなし，スタートストップビット 1bit
- 5V

コンストラクタ

UART.new(n)

例

```
uart1 = UART.new( 1 )    # param: grove pin number.
```

出力

write(string)

指定された文字列を出力します。

例

```
uart1.write("Output string¥r¥n")
```

入力

read(n_bytes) -> String, Nil

指定されたバイト数のデータを読み込みます。指定されたバイト数のデータが到着していない場合、nil を返します。

例

```
val = uart1.read( 10 )
```

（参考）通常、データが到着していなければブロックする方式が一般的ですが、mruby/c で作るアプリケーションの場合、あえてブロックしない方式としたほうがプログラミングしやすかったので、そのような仕様になっています。

read_nonblock(maxlen) -> String

指定されたバイト数のデータを読み込みます。指定されたバイト数のデータが到着していない場合、到着している分のデータを返します。

例

```
val = uart1.read_nonblock( 1024 )
```

gets()

文字列を一行読み込みます。実際には受信キュー内の "\n" までのバイト列を返します。受信キューに "\n" が無い場合、nil を返します。

例

```
val = uart1.gets()
```

（参考）受信キューより長い文字列を受信した場合、gets()では処理できません。その場合、受信キューを大きくしたファームウェアを用意する必要があります。

その他

clear_tx_buffer()

読み込みバッファをクリアします。

例

```
uart1.clear_tx_buffer()
```

clear_rx_buffer()

書き込みバッファをクリアします。

例

```
uart1.clear_rx_buffer()
```

シリアル通信　I2C

I2C シリアルインターフェースを扱います。
仕様

- 5V
- 100kbps
- マスタモードのみ

コンストラクタ

I2C.new()

例

```
i2c = I2C.new()
```

出力

write(i2c_adrs_7, data1, data2,...)

指定されたバイトを順次出力します。

例

```
i2c.write( ADRS, 0x02, 0x16, 0x00 )
```

write(i2c_adrs_7, "string")

指定された文字列を出力します。

例

```
i2c.write( ADRS, "string" )
```

入力

read(i2c_adrs_7, read_bytes, *params) -> String

指定されたバイト数のデータを読み込みます。指定されたバイト数のデータが到着していない場合、ブロックします。
params を指定することで、read 前に params を出力します。すなわち、以下のシーケンスとなります。

(S) - ADRS7 (W)(A) - [params ...] - (Sr) - ADRS7 (R)(A) - data_1 (A)... data_n (A|N) - (P)
S : Start condition　　P : Stop condition
Sr: Repeated start condition
A : Ack　N : Nack

例

```
s = i2c.read( ADRS, 2, 0xfe )　　# 0xfe レジスタから 2 バイト取得
```

シリアル通信　SPI

SPI シリアルインターフェースを扱います。

コンストラクタ

SPI.new()

例

```
spi = SPI.new()
```

出力

write(data1, data2,...)

指定されたバイトを順次出力します。

例

```
spi.write( 0x02, 0x16, 0x00 )
```

write("string")

指定された文字列を出力します。

例

```
spi.write( "string" )
```

入力

read(read_bytes) -> String

指定されたバイト数のデータを読み込みます。

例

```
s = spi.read( 2 )
```

汎用転送

transfer([d1, d2,...], recv_size) -> String

d1,d2...を送信し、その後 recv_size 分の 0x00 を送信します。
戻り値は、受信した長さ recv_size バイトの文字列となります。

パルス幅変調　PWM

パルス幅変調(Pulse Width Modulation)を扱います。周波数を指定する方法と、周期を指定する方法があります。
本ボードでは、0 番に圧電スピーカーを接続しています。1番は、GPIO1 と出力ピンを共有しています。

コンストラクタ

PWM.new(n)

例

```
pwm = PWM.new(0)
```

周波数を指定する方法

frequency(n)

指定した周波数の信号を出力します。最大値は利用する機器に依存します。0 を指定すると出力を停止します。

duty(n)

デューティー比を、0（全オフ）から 1023（全オン）までで指定します。

例

```
pwm.frequency( 440 )   # 440Hz
pwm.duty( 512 )        # 50%
```

周期を指定する方法

period_us(n)

周期を指定して信号を出力します。単位はマイクロ秒 (uS) です。最大値は利用する機器に依存します。0 を指定すると出力を停止します。

例

```
pwm.period_us( 2273 )   # 440Hz
pwm.duty( 512 )         # 50%
```

Mutex

複数プログラムの同期用に、Mutex を使う事ができます。

```
$mutex1 = Mutex.new
$mutex1.lock()
$mutex1.unlock()
$mutex1.try_lock()
```

その他のメソッド

mruby/c ランタイムスケジューラには、以下のメソッドを用意しています。

sleep(n)

実行一時停止。秒単位。

sleep_ms(n)

実行一時停止。ミリ秒単位。

relinquish()

他のタスクに実行を譲ります。