K4ZUKI /KY技研

まえがき

このドキュメントは何

この本は、Arduino UNO R4 minima (以下 “Arduino R4”) の低レベル(ベアメタル)プログラミング研究のため、手始めに PWMをいじろうとして色々うんざりした経緯を語るドキュメントです。R4 Wifiは実験していません。

この本で解説する範囲

Arduino R4の2ピンを使って相補型PWMを1チャンネル出すまでを試します。 Arduino R4のMCUには、全部で8チャネルの汎用PWMタイマ（GPT: General PWM Timer）が内蔵されています。カウンタのビット幅以外にチャネル間の機能の差はありません。16ビットのタイマでも32ビットレジスタを持っているふりをします。 Arduino R4の右側、D0-D13ピンは全部いずれかのチャンネルにつながっていて、タイマごとに2相の信号を出力できます。

ところで、ネットを探してみると、少ないながらも[[1]](#footnote-1)ArduinoR4でベアメタルプログラミングをしているブログ記事が見つかります。いまやろうとしている相補PWMも、まさにそれを取り扱った記事がすでにあったりします。Arduinoライブラリの利用をあきらめて、ハードウェアマニュアルを片手に頑張って実装したそうです。 なので、やや読みづらいソースコード[[2]](#footnote-2)になっています。つまり、どこのレジスタのどのビットをどのように操作するのか、というゴールはもうわかっています。この本では、ライブラリを使うことで、可読性を上げたプログラムにすることを目標にします。

おことわり

以下のマイコンボードで実験しました。リストにないものは触っていません。

* Arduino UNO R4 minima <https://docs.arduino.cc/hardware/uno-r4-minima/>
* FLINT ProMicro R4 <https://flint.works/p/flint-promicro-r4/>

コンパイラやライブラリの環境は以下のとおりです。

* Arduino IDE 2.3.6

筆者はMac版・Windows版両方を使っていますが、このドキュメント内ではWindows版のキーアサインで説明します。なお、スクリーンショットはMac版になることがあります。

* arduino-cli 1.2.2

IDEも内部的に使っていますが、別個にインストールしています。設定ファイル類は共用するようです。

* arduino:renesas\_uno 1.5.0

ボードマネージャからArduino UNO R4 Boardsをインストールします。

実装コードの引用はGitHubリポジトリ<https://github.com/arduino/ArduinoCore-renesas>の1.5.0タグをダウンロードして直接参照しています。

* Renesas FSP 4.0.0

FSPライブラリの実装を確かめるためにGitHubリポジトリ<https://github.com/renesas/fsp>からソースコード一式をダウンロードしました。上記ボードと同じv4.0.0を参照しています。

RA4M1のデータシート「ハードウェアマニュアル」(たぶん日立語)は **Rev.1.10 Sep 29, 2023**を参照しました。

Table of Contents

[まえがき 1](#_Toc206086771)

[このドキュメントは何 1](#_Toc206086772)

[この本で解説する範囲 1](#_Toc206086773)

[おことわり 1](#_Toc206086774)

[1 先人の成果を調べてみる 5](#_Toc206086775)

[1.1 お手本実装 5](#_Toc206086776)

[1.2 今回の計画 6](#_Toc206086777)

[2 PWM.h を読み解いてみる 7](#_Toc206086778)

[2.1 PWM.h → pwm.h 7](#_Toc206086779)

[2.2 PwmOut クラス定義 7](#_Toc206086780)

[2.3 PwmOut::begin() 8](#_Toc206086781)

[2.3.1 周波数とデューティー比を設定するモード 8](#_Toc206086782)

[2.4 PwmOut::cfg\_pin() (プライベート関数) 9](#_Toc206086783)

[2.4.1 R\_IOPORT\_PinCfg() 9](#_Toc206086784)

[3 FspTimer.h（PWMモード関連部分）を使った最終的な実装（例） 11](#_Toc206086785)

[3.1 PWMモードのおまじないadd\_pwm\_extended\_cfg() 12](#_Toc206086786)

[3.2 各種コンフィグを操作して準備 12](#_Toc206086787)

[3.2.1 timer\_cfg\_t\* FspTimer::get\_cfg() 12](#_Toc206086788)

[3.2.2 gpt\_extended\_cfg\_t \*p\_extend 12](#_Toc206086789)

[3.3 open()：設定情報がペリフェラルに渡る 12](#_Toc206086790)

[3.4 start()：タイマーのカウントを開始 12](#_Toc206086791)

[3.5 結局FSPライブラリのみを用いてお手本コードを再現するのは無理だった 12](#_Toc206086792)

[3.5.1 gpt\_extended\_pwm\_cfg\_t \*p\_pwm\_cfg 12](#_Toc206086793)

[3.5.2 gpt\_gtior\_setting\_t gtior\_setting 12](#_Toc206086794)

[あとがき 13](#_Toc206086795)

# 先人の成果を調べてみる

## お手本実装

*小倉 キャッスル 一馬* 氏のブログに筆者がやろうとしていることがほぼ全部書かれています。以下の記事と続編をたどると、 ArduinoのAPIや関数をほぼ使わず、レジスタの直接操作をして相補PWMにしています。

* [「任意のポートでPWM出力させたい」 (2023-11-14)](https://ameblo.jp/ogura-castle/entry-12828562536.html)
* [「任意のポートでPWM出力させるためのレジスタ制御」 (2023-11-15)](https://ameblo.jp/ogura-castle/entry-12828565434.html)

２つ目の記事に載っているコードを下に示します。1kHz、デューティー比25%の矩形波がD0/D1から出力されます。

▼ List 1.1: 参考にしたコード

/\*\* As-is copy from https://ameblo.jp/ogura-castle/entry-12828565434.html   
\*/   
   
#include <FspTimer.h>   
static FspTimer fsp\_timer;   
   
void setup() {   
 fsp\_timer.begin(TIMER\_MODE\_PWM, 0, 4, 1000.0, 25.0);   
 fsp\_timer.open();   
   
 R\_PFS->PORT[3].PIN[1].PmnPFS\_b.PMR = 0; //D0を汎用入出力に設定   
 R\_PFS->PORT[3].PIN[2].PmnPFS\_b.PMR = 0; //D1を汎用入出力に設定   
 R\_PFS->PORT[3].PIN[1].PmnPFS\_b.PSEL = 0x03; //D0をGPT端子として使用   
 R\_PFS->PORT[3].PIN[2].PmnPFS\_b.PSEL = 0x03; //D1をGPT端子として使用   
 R\_PFS->PORT[3].PIN[1].PmnPFS\_b.PMR = 1; //D0を周辺機能用の端子に設定   
 R\_PFS->PORT[3].PIN[2].PmnPFS\_b.PMR = 1; //D1を周辺機能用の端子に設定   
   
 R\_GPT4->GTIOR\_b.GTIOA = 0x06; //GTIOCA端子機能選択   
 R\_GPT4->GTIOR\_b.GTIOB = 0x19; //GTIOCB端子機能選択   
 R\_GPT4->GTIOR\_b.OAE = 1; //GTIOCA端子出力許可   
 R\_GPT4->GTIOR\_b.OBE = 1; //GTIOCB端子出力許可   
   
 fsp\_timer.start();   
}   
   
void loop() {   
   
}

キャッスル氏のコードをだいたい要約すると、

* FspTimerライブラリを使ってタイマーペリフェラルを起動 (begin())
* タイマーの諸設定を行う (open())
* IOピンのレジスタでタイマー機能ににコントロールを渡す (R\_PFS->...)
* タイマーのレジスタでピン出力変化タイミングを決める (R\_GPT4->...)
* タイマー起動（カウント開始、信号出力開始）(start())

という手順を踏んでいます。注意点として、このコード例では、AB相間のデッドタイムは設定されません。また、デューティー比を変えるときに両方ともHになる期間が発生する可能性があります。

## 今回の計画

* キャッスル式をお手本にして、同じ結果を目指します。D0/D1ピンに、互いに逆相のノコギリ波PWM信号を出すことがゴールです。 D0/D1にはGPT4が割り当てられています。[tbl. 1.1](#tbl:arduino-r4-pin-assignments)を参考にしてください。
* 次に、デッドタイムつきで三角波PWMを同じく逆相で出します。A相のパルス幅を更新すると、 B相はハードウェアが計算して自動的に更新してくれます。

また、FspTimerの使い方は以下のブログページを参考にしました。

* [「Arduino UNO R4のFspTimerライブラリの使い方」 （2023-08-11 ）](https://qiita.com/yasuhiro-k/items/93efb640aa12f3db9086)
* [「Arduino UNO R4 でタイマー割り込みを使う」 （2024.10.21）](https://workshop.aaa-plaza.net/archives/1658)

▼ Table 1.1: Arduino UNO R4 Minima ピンアサイン一覧

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| GPT channel | RA4M1 | Arduino |  | Arduino | RA4M1 | GPT channel |
| - | - | - |  | SCL | **P100** | **5B** |
| - | - | - |  | SDA | **P101** | **5A** |
| - | - | - |  | AREF | AREF | - |
| - | - | - |  | GND | GND | - |
| - | MD | NC |  | D13/SCK | P111 | 3A |
| - | 5V | IOREF |  | D12/MISO | P110 | 1B |
| - | /RESET | RESET |  | D11/MOSI | P109 | 1A |
| - | - | 3V3 |  | D10/CS~ | P112 | 3B |
| - | 5V | 5V |  | D9~ | P303 | 7B |
| - | GND | GND |  | D8~ | P304 | 7A |
| - | GND | GND |  | — | — | — |
| - | VIN | VIN |  | D7 | P107 | 0A |
| — | — | — |  | D6 | P106 | 0B |
| - | P014 | A0 |  | D5~ | P102 | 2B |
| - | P000 | A1 |  | D4~ | P103 | 2A |
| - | P001 | A2 |  | D3 | P104 | 1B |
| - | P002 | A3 |  | D2~ | P105 | 1A |
| **5A** | **P101** | A4 |  | D1/TX | P302 | 4A |
| **5B** | **P100** | A5 |  | D0/RX | P301 | 4B |

# PWM.h を読み解いてみる

## PWM.h → pwm.h

まずPWM.hをインクルードしてみます。多くの作例では、ここで定義されているPwmOutオブジェクトを使っています。 IDE上でCtrl+LMBを使いヘッダに飛ぶと、  
.../Arduino15/packages/arduino/hardware/renesas\_uno/1.5.0/cores/arduino/pwm.h となっていました。この部分はターゲットボードごとに変わると思います。実装ファイルはヘッダと同じディレクトリにある .../pwm.cppです。

ヘッダの中身を見ると、はじめの方で早速Arduino.h と FspTimer.h をインクルードしています。

▼ List 2.1: PWM.h (先頭部分抜粋)

#include "Arduino.h"  
#include "FspTimer.h"  
  
#ifndef \_\_ARDUINO\_PWM\_H\_\_  
#define \_\_ARDUINO\_PWM\_H\_\_  
  
  
class PwmOut {

インスタンス宣言してコンストラクタを呼び出した時点では、ピン番号を内部に保持するだけで何もしません。 begin関数を呼び出すと、内部で初期設定が行われます。

## PwmOut クラス定義

▼ List 2.2: PwmOut クラス定義(ヘッダ)

class PwmOut {  
 public:  
 PwmOut(int pinNumber);  
 ~PwmOut();  
  
 /\* that is "default" begin that set up a PWM with 490Hz of frequency and 50% of duty cycle \*/  
 bool begin();  
 /\* with raw = false then period and pulse are supposed to be in micro second  
 with raw = true then period\_usec and pulse\_usec will be user as raw value to be used directly in micro registers   
 Pay attention that in case of raw use the possible divider are  
 TIMER\_SOURCE\_DIV\_1  
 TIMER\_SOURCE\_DIV\_4   
 TIMER\_SOURCE\_DIV\_16  
 TIMER\_SOURCE\_DIV\_64  
 TIMER\_SOURCE\_DIV\_256  
 TIMER\_SOURCE\_DIV\_1024 \*/  
 bool begin(uint32\_t period\_usec, uint32\_t pulse\_usec, bool raw = false, timer\_source\_div\_t sd = TIMER\_SOURCE\_DIV\_1);  
 bool begin(float freq\_hz, float duty\_perc);  
 void end();  
 bool period(int ms);  
 bool pulseWidth(int ms);  
 bool period\_us(int us);  
 bool pulseWidth\_us(int us);  
 bool period\_raw(int period);  
 bool pulseWidth\_raw(int pulse);  
 bool pulse\_perc(float duty);  
 void suspend();  
 void resume();  
  
 FspTimer \*get\_timer() {return &timer;}  
  
 private:  
 bool cfg\_pin(int max\_index);  
 int \_pin;  
 bool \_enabled;  
 bool \_is\_agt;  
 TimerPWMChannel\_t \_pwm\_channel;  
   
 uint8\_t timer\_channel;   
 FspTimer timer;  
};

privateメンバとしてFspTimerオブジェクトtimerが使われています。timerへのポインタを渡すget\_timer()関数を通じてアクセスできます。用意されている関数でカバーされていない設定を行うときは、get\_timer()経由でオブジェクトを取得してFspTimerクラスの関数やメンバオブジェクトを操作します。

FspTimer \*get\_timer() {return &timer;}

## PwmOut::begin()

IOとペリフェラルの確保と初期設定を行い、PWM信号の出力を即時開始する関数です。 3種類の実装が用意されています。どの実装でも、ピンの制御をGPTのタイマブロックに割り当て[[3]](#footnote-3)、対応するタイマーをノコギリ波PWMモードに初期化します。

タイマーブロックの種類はICレベルでピンごとに固有の割り当てが定義されています。アナログ用のピン(A0~A3)には割り当てがありません。デジタルピン(D0~D13)とI2C兼用ピン(A4,A5)だけが対象です。

### 周波数とデューティー比を設定するモード

bool begin(float freq\_hz, float duty\_perc);

スイッチング周波数とデューティー比を指定するモードです。周波数はメインクロックまで指定できると思われますが、筆者は実験していません。現実的には480kHzが上限と思われます[[4]](#footnote-4)。

▼ List 2.3: PwmOut::begin() (周波数・デューティー指定モード・pwm.cpp抜粋)

/\* -------------------------------------------------------------------------- \*/  
bool PwmOut::begin(float freq\_hz, float duty\_perc) {  
/\* -------------------------------------------------------------------------- \*/  
 \_enabled = true;  
 int max\_index = PINS\_COUNT;  
 \_enabled &= cfg\_pin(max\_index);  
  
 if(\_enabled) {  
 \_enabled &= timer.begin(TIMER\_MODE\_PWM, (\_is\_agt) ? AGT\_TIMER : GPT\_TIMER, timer\_channel, freq\_hz, duty\_perc);  
 }  
  
 if(\_enabled) {  
 timer.add\_pwm\_extended\_cfg();  
 timer.enable\_pwm\_channel(\_pwm\_channel);  
  
 \_enabled &= timer.open();  
 \_enabled &= timer.start();  
 }  
  
 return \_enabled;  
}

## PwmOut::cfg\_pin() (プライベート関数)

IOピンがPWMに使えるかを検査し、~~謎の~~FSPライブラリ関数R\_IOPORT\_PinCfg()を使って設定を行います。 PWMライブラリを使わずとも、この関数に適切な引数を渡せばIOピンは設定できそうです[[5]](#footnote-5)。

▼ List 2.4: PwmOut::cfg\_pin() (pwm.cpp抜粋)

bool PwmOut::cfg\_pin(int max\_index) {  
 /\* verify index are good \*/  
 if(\_pin < 0 || \_pin >= max\_index) {  
 return false;  
 }  
 /\* getting configuration from table \*/  
 auto pin\_cgf = getPinCfgs(\_pin, PIN\_CFG\_REQ\_PWM);  
  
 /\* verify configuration are good \*/  
 if(pin\_cgf[0] == 0) {  
 return false;  
 }  
  
 timer\_channel = GET\_CHANNEL(pin\_cgf[0]);  
  
 \_is\_agt = IS\_PIN\_AGT\_PWM(pin\_cgf[0]);  
  
 \_pwm\_channel = IS\_PWM\_ON\_A(pin\_cgf[0]) ? CHANNEL\_A : CHANNEL\_B;  
  
 /\* actually configuring PIN function \*/  
 R\_IOPORT\_PinCfg(&g\_ioport\_ctrl, g\_pin\_cfg[\_pin].pin, (uint32\_t) (IOPORT\_CFG\_PERIPHERAL\_PIN | (\_is\_agt ? IOPORT\_PERIPHERAL\_AGT : IOPORT\_PERIPHERAL\_GPT1)));  
 return true;  
  
}

### R\_IOPORT\_PinCfg()

ルネサスの”FSPライブラリ”内にある関数です。お手本コードで直接レジスタ操作していた部分の置き換えに使えます。[ヘルプページも存在します](https://renesas.github.io/fsp/group___i_o_p_o_r_t.html#gab518fc544fe2b59722e30bd0a28ef430)が、Arduinoが使っているものとはバージョンが異なっている[[6]](#footnote-6)ので、残念ながら助けにはなりません。 FSPのソースコードをリポジトリから入手して内容を見るのがベストです。

▼ List 2.5: FSPバージョン情報

/\*\* Public FSP version name. \*/  
 #define FSP\_VERSION\_STRING ("4.0.0")

この関数の引数一覧を[tbl. 2.1](#tbl:r-ioport-pincfg-param-list)に示します。

どうやら最初の引数p\_ctrlは互換性のためにあるらしく、cfg\_pin()では共用の変数を使用しています。次のpinは専用の数値型ですが、Arduino形式から変換したものを渡せばOKです。最後のcfgは、渡す値がそのままPmnPFSレジスタの構造に当てはまります。 cfg\_pin()ではAGT・GPTの切り替え判定をするためややこしく見えますが、IOPORT\_CFG\_PERIPHERAL\_PINと IOPORT\_PERIPHERAL\_GPT1の論理和をuint32\_tにキャストして渡せば大丈夫です。

▼ Table 2.1: R\_IOPORT\_PinCfg() 引数一覧

| Parameter | Type | Purpose | Note |
| --- | --- | --- | --- |
| p\_ctrl | ioport\_ctrl\_t \* const | Unused | 共用の固定値を使用 |
| pin | bsp\_io\_port\_pin\_t | Pin identifier | g\_pin\_cfg[\_pin].pin; \_pinにはD13などArduino形式のピン番号を指定できる |
| cfg | uint32\_t | Directly update PmnPFS register | (uint32\_t) (IOPORT\_CFG\_PERIPHERAL\_PIN｜IOPORT\_PERIPHERAL\_GPT1) |

# FspTimer.h（PWMモード関連部分）を使った最終的な実装（例）

お手本コードに使われいることから薄々わかっていたことですが、結局FspTimerが相補PWMのゴールには最短のようです。

PwmOutと同様に、コンストラクタ呼び出し時点では大したことはしません。begin()で各種オブジェクトの構築が始まります(信号はまだ出ません)。こちらの実装も互換モード用のbegin\_pwm()を含め3つあります。ここでは適度に簡単な実装（[lst. 3.1](#lst:fsptimer-pwm-begin)）を採用します。

▼ List 3.1: FspTimer::begin() (FspTimer.cpp抜粋)

/\* -------------------------------------------------------------------------- \*/  
bool FspTimer::begin(timer\_mode\_t mode, uint8\_t tp, uint8\_t channel, float freq\_hz, float duty\_perc, GPTimerCbk\_f cbk /\*= nullptr\*/ , void \*ctx /\*= nullptr\*/ ) {  
/\* -------------------------------------------------------------------------- \*/   
   
 init\_ok = true;  
 /\* AGT timer is always 16 bit \*/  
 if(channel < TIMER\_16\_BIT\_OFFSET && tp == GPT\_TIMER) {  
 /\* timer a 32 BIT \*/  
 set\_period\_counts(tp, 1.0 / freq\_hz, CH32BIT\_MAX);  
 }  
 else {  
 /\* timer a 16 BIT \*/  
 set\_period\_counts(tp, 1.0 / freq\_hz, CH16BIT\_MAX);  
 }  
  
 if(duty\_perc >= 0 && duty\_perc <= 100) {  
 \_duty\_cycle\_counts = (uint32\_t)(((float)\_period\_counts \* duty\_perc) / 100.0);  
 }  
 else {  
 init\_ok = false;  
 }  
   
 if(init\_ok) {  
 init\_ok = begin(mode, tp, channel, \_period\_counts, \_duty\_cycle\_counts, \_sd, cbk, ctx);  
 }  
 return init\_ok;  
}

ここでmodeにはTIMER\_MODE\_PWM（ノコギリ波PWM）またはTIMER\_MODE\_TRIANGLE\_WAVE\_SYMMETRIC\_PWM（三角波PWM・モード1）を渡します。tpはタイマーの種類（GPTまたはAGT）、channelはGPTのチャネル（今回は4）、残りはスイッチング周波数とデューティー比です。最後のパラメータ2つは省略できます。

周波数の設定には少し注意が必要です。三角波PWMのときは設定したい周波数の**2倍**を与える必要があります。 PwmOutのときと同様に周波数の現実的な上限は480kHzです。

以上から、スケッチの冒頭はだいたい以下のような感じになります。

#include "Arduino.h"  
#include "FspTimer.h"  
  
#define \_USE\_GPT4 (4)  
#define \_PWM\_FREQ (4.0e3)  
#define \_PWM\_DUTY (50.0f)  
  
FspTimer gpt4;  
  
void setup() {  
 // Initialise GPT4 timer with saw-tooth PWM  
 gpt4.begin(TIMER\_MODE\_PWM, GPT\_TIMER, \_USE\_GPT4, \_PWM\_FREQ, \_PWM\_DUTY);  
   
 // Initialise GPT4 timer with triangle PWM mode 1  
 gpt4.begin(TIMER\_MODE\_TRIANGLE\_WAVE\_SYMMETRIC\_PWM, GPT\_TIMER, \_USE\_GPT4, (2 \* \_PWM\_FREQ), \_PWM\_DUTY);   
}

## PWMモードのおまじないadd\_pwm\_extended\_cfg()

より詳細に初期化する方のbegin()の中でtimer\_cfgという設定リストが初期化されます。FspTimerをPWMモードで使うときは、このリストを更に拡張するadd\_pwm\_extended\_cfg()を実行する必要があります。timer\_cfgの拡張部分がポインタ渡しなので、これを忘れるとわかりにくいバグり方をします（何回かブートローダを入れ直しました）。ヘルパー関数get\_cfg()がtimer\_cfgへのポインタを返します。

## 各種コンフィグを操作して準備

### timer\_cfg\_t\* FspTimer::get\_cfg()

設定リストtimer\_cfgへのポインタを返すヘルパー関数です。とくにp\_extendと、その中のPWMモード固有設定リストp\_pwm\_cfg にアクセスするときに使います。

/\*\* prepare variables to update config \*/  
 timer\_cfg\_t\* gpt4\_cfg = gpt4.get\_cfg();  
 gpt4\_ext\_cfg = (gpt\_extended\_cfg\_t\*)gpt4\_cfg->p\_extend;  
 gpt4\_ext\_pwm\_cfg = (gpt\_extended\_pwm\_cfg\_t\*)gpt4\_ext\_cfg->p\_pwm\_cfg;

### gpt\_extended\_cfg\_t \*p\_extend

## open()：設定情報がペリフェラルに渡る

## start()：タイマーのカウントを開始

## 結局FSPライブラリのみを用いてお手本コードを再現するのは無理だった

### gpt\_extended\_pwm\_cfg\_t \*p\_pwm\_cfg

### gpt\_gtior\_setting\_t gtior\_setting

あとがき

今回も前日印刷TMです[[7]](#footnote-7)。主な遅延要因はバトルフィールド６のオープンベータとタルコフのワイプです。 BF3リメイクと言った感じでとても楽しいです。2週連続でベータ期間で、2回目はコミケと重なってしまうのであまりできないかもしれないですが、日本時間の木曜日夕方開始だし、タイムゾーンの差を活かして月曜日まで楽しもうと思います。

タルコフの進捗は、グランドゼロマップが嫌すぎて、プラパーの最初のタスクで詰まってます。メカニックとスキヤーも詰まってます。かろうじてセラピは進んだけど、イエガー未開放奴です。本編がつらいからってアリーナでしばらく遊んでたら勝手にレベル14まで進んでしまった（アリーナはまあまあ面白いって感じです）。タスクは進んでないけど取引額の条件は満たしているのでセラピッピとピスキはLL2になっているでしょう。

ハードコアワイプ初日のタスクなしモード、悪くないと思ったんですが、コミュニティは好きじゃないみたいですね。

Arduino R4のデバッガにTsuboLinkII（現在は入手不可）を使おうとして一時沼りましたが、過去の自分がDropboxに残していたファームウェアのバックアップを書き込んで事なきを得ました。やっててよかったｴｪｪｪｪﾝﾍﾞｯﾄﾞ! まあMacとの相性が悪かったのかなー

*  ← 原稿はこちらから

|  |
| --- |
| そういうとこやぞ® シリーズ / Arduino R4のPWMライブラリ実装を解析した話 |
| K4ZUKI / KY技研 |
| Comiket 106 東 Ｓ-44b (東７ホール) / 2025 |
| Markdown x Pandoc Version 3.5 (DOCX writer) with docx-coreprop-writer program |
| キンコーズオンラインで印刷製本 |

1. 記事の少なさはR4が登場して2年？くらいしか経っていないからでしょう。ライブラリの互換性が良くて、ベアメタルに手を出す必要がないんだと思います。 [↑](#footnote-ref-1)
2. 読みづらいのは、ブログ主さんの問題ではなくて、マイコンのレジスタフィールドの略称が一見では意味不明なところから来ていると思います。 [↑](#footnote-ref-2)
3. ソースを見るとAGTとの切り分けが各所に入っていますが、少なくともUNO R4では、AGTが割り当てられる可能性はなさそうです。 [↑](#footnote-ref-3)
4. メインが48MHzなので、スイッチングが480kHzのときデューティー1％が1クロックになります。スイッチングを1MHzにすると1周期が48クロック（デューティー1％あたり0.5クロックくらい）しかありません。 [↑](#footnote-ref-4)
5. PWMライブラリに極小デューティーを設定してから停止させて諸設定を乗っ取る方法も試しましたが、よくわからないバグが増えただけでした。素直にFspTimerライブラリを使ったほうがいいです。 [↑](#footnote-ref-5)
6. ヘッダ情報によると、FSPバージョンv4.0.0を使っています。 [↑](#footnote-ref-6)
7. このあとがきは木曜日朝7時頃に書かれました。本編はまだ書き終わっていません。 [↑](#footnote-ref-7)