# SmartRA 学習キット
# チュートリアル 9

ルネサス エレクトロニクス社　RA マイコン搭載
HSB シリーズマイコンボード　評価キット

-本書を必ずよく読み、ご理解された上でご利用ください

株式会社北斗電子

REV.1.0.0.0

ー目　次ー

# 注意事項

本書を必ずよく読み、ご理解された上でご利用ください

## 【ご利用にあたって】

1. 本製品をご利用になる前には必ず取扱説明書をよく読んで下さい。また、本書は必ず保管し、使用上不明な点がある場合は再読し、よく理解して使用して下さい。
2. 本書は株式会社北斗電子製マイコンボードの使用方法について説明するものであり、ユーザシステムは対象ではありません。
3. 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
4. 弊社のマイコンボードの仕様は全て使用しているマイコンの仕様に準じております。マイコンの仕様に関しましては製造元にお問い合わせ下さい。弊社製品のデザイン・機能・仕様は性能や安全性の向上を目的に、予告無しに変更することがあります。また価格を変更する場合や本書の図は実物と異なる場合もありますので、御了承下さい。
5. 本製品のご使用にあたっては、十分に評価の上ご使用下さい。
6. 未実装の部品に関してはサポート対象外です。お客様の責任においてご使用下さい。

## 【限定保証】

1. 弊社は本製品が頒布されているご利用条件に従って製造されたもので、本書に記載された動作を保証致します。
2. 本製品の保証期間は購入戴いた日から1年間です。

## 【保証規定】

**保証期間内でも次のような場合は保証対象外となり有料修理となります**

1. 火災・地震・第三者による行為その他の事故により本製品に不具合が生じた場合
2. お客様の故意・過失・誤用・異常な条件でのご利用で本製品に不具合が生じた場合
3. 本製品及び付属品のご利用方法に起因した損害が発生した場合
4. お客様によって本製品及び付属品へ改造・修理がなされた場合

## 【免責事項】

弊社は特定の目的･用途に関する保証や特許権侵害に対する保証等、本保証条件以外のものは明示・黙示に拘わらず一切の保証は致し兼ねます。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任を負いません。損害の発生についてあらかじめ知らされていた場合でも保証は致し兼ねます。
ただし、明示的に保証責任または担保責任を負う場合でも、その理由のいかんを問わず、累積的な損害賠償責任は、弊社が受領した対価を上限とします。本製品は「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。
保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。
本製品を使った二次製品の保証は致し兼ねます。

# 安全上のご注意

製品を安全にお使いいただくための項目を次のように記載しています。絵表示の意味をよく理解した上でお読み下さい。

**表記の意味**

**警告** 取扱を誤った場合、人が死亡または重傷を負う危険が切迫して生じる可能性がある事が想定される

**注意** 取扱を誤った場合、人が軽傷を負う可能性又は、物的損害のみを引き起こすが可能性がある事が想定される

## 絵記号の意味

| | | | |
|---|---|---|---|
|  | **一般指示**<br>使用者に対して指示に基づく行為を強制するものを示します |  | **一般禁止**<br>一般的な禁止事項を示します |
|  | **電源プラグを抜く**<br>使用者に対して電源プラグをコンセントから抜くように指示します |  | **一般注意**<br>一般的な注意を示しています |

## 警告

**以下の警告に反する操作をされた場合、本製品及びユーザシステムの破壊・発煙・発火の危険があります。マイコン内蔵プログラムを破壊する場合もあります。**

1. 本製品及びユーザシステムに電源が入ったままケーブルの抜き差しを行わないでください。
2. 本製品及びユーザシステムに電源が入ったままで、ユーザシステム上に実装されたマイコンまたはIC等の抜き差しを行わないでください。
3. 本製品及びユーザシステムは規定の電圧範囲でご利用ください。
4. 本製品及びユーザシステムは、コネクタのピン番号及びユーザシステム上のマイコンとの接続を確認の上正しく扱ってください。

**発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。**

電源がある場合は電源を切って、コンセントから電源プラグを抜いてください。そのままご使用すると火災や感電の原因になります。

# ⚠注意

**以下のことをされると故障の原因となる場合があります。**

1. 静電気が流れ、部品が破壊される恐れがありますので、ボード製品のコネクタ部分や部品面には直接手を触れないでください。
2. 次の様な場所での使用、保管をしないでください。

   ホコリが多い場所、長時間直射日光があたる場所、不安定な場所、衝撃や振動が加わる場所、落下の可能性がある場所、水分や湿気の多い場所、磁気を発するものの近く
3. 落としたり、衝撃を与えたり、重いものを乗せないでください。
4. 製品の上に水などの液体や、クリップなどの金属を置かないでください。
5. 製品の傍で飲食や喫煙をしないでください。

**ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。**

取り付け、取り外しの際は製品の両端を持ってください。裏面のハンダ付け跡で、誤って手など怪我をする場合があります。

**CD メディア、フロッピーディスク付属の製品では、故障に備えてバックアップ(複製)をお取りください。**

製品をご使用中にデータなどが消失した場合、データなどの保証は一切致しかねます。

**アクセスランプがある製品では、アクセスランプの点灯中に電源を切ったり、パソコンをリセットをしないでください。**

製品の故障や、データ消失の原因となります。

**本製品は、医療、航空宇宙、原子力、輸送などの人命に関わる機器やシステム及び高度な信頼性を必要とする設備や機器などに用いられる事を目的として、設計及び製造されておりません。**

医療、航空宇宙、原子力、輸送などの設備や機器、システムなどに本製品を使用され、本製品の故障により、人身や火災事故、社会的な損害などが生じても、弊社では責任を負いかねます。お客様ご自身にて対策を期されるようご注意ください。

# 1. RA2L1_FULL_COLOR_LED

フルカラーLED の制御を行うチュートリアルです。

## 1.1. フルカラーLED の概要

市販のフルカラーLED は、R(Red), G(Green), B(Blue)の 3 本の端子に流す電流で任意の色を出せるタイプのものと、マイコンが内蔵されておりデジタル信号で制御できるタイプのものがあります。

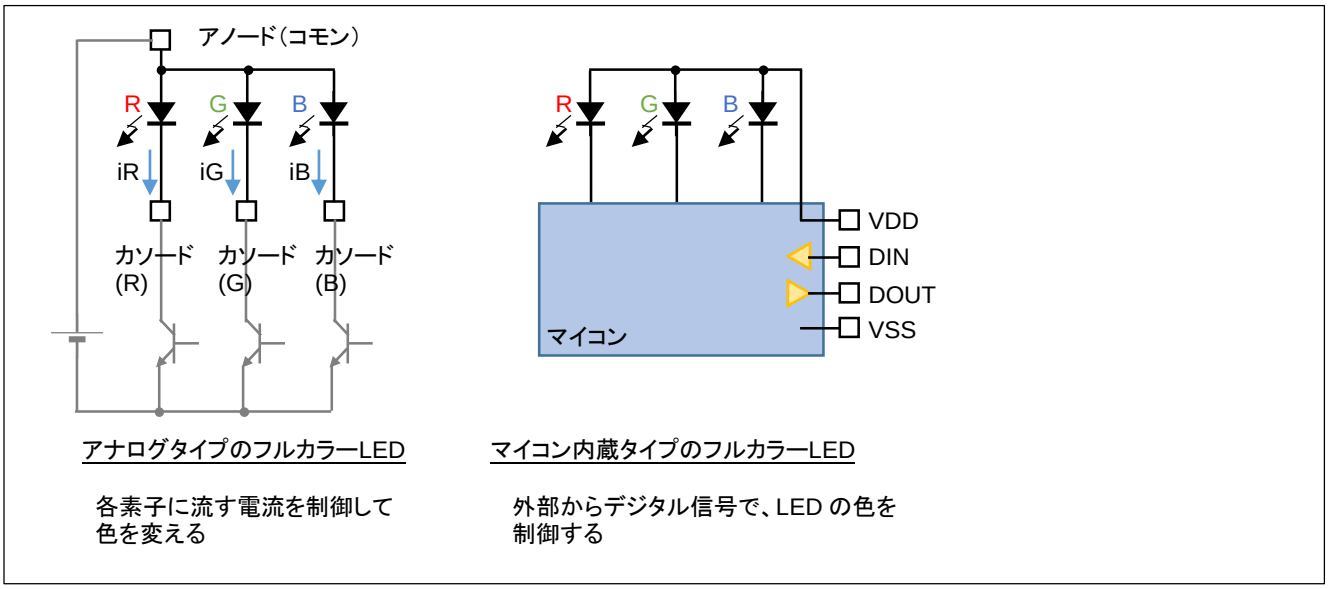



**図 1-1 フルカラーLED の種別**

本チュートリアルでは、マイコン内蔵タイプのフルカラーLED を制御する事を考えます。

フルカラーLED は、制御するマイコン（ここでは、RA2L1 マイコンボード）からデジタルで信号を送る事により、色や明るさを変えられます。

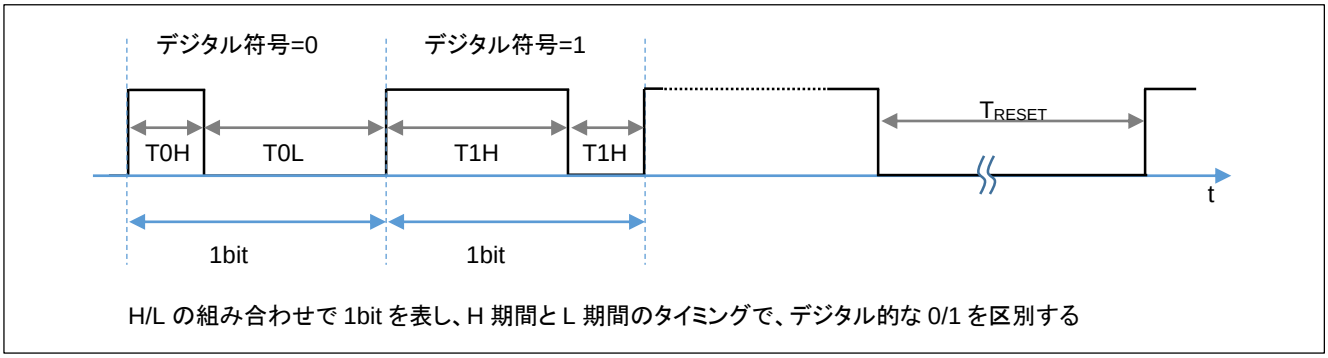



**図 1-2 フルカラーLED の制御波形**

送信する信号は、H/L が繰り返される波形となりますが、デジタルで 0 を表す場合と 1 を表す場合で、H 期間と L 期間のタイミングが変わります。

LED に、R（赤）G（緑）B（青）を、それぞれ 8bit 合計 24bit のデータを送って色調を制御します。

代表的なフルカラーLED のタイミングパラメータを示します。

**表 1-1 フルカラーLED のタイミングパラメータ**

| | PL9823 | WS2812B | WS2813<br>OSTW3535C1A | SK6812 | OST45050 | 設計<br>ターゲット |
|---|---|---|---|---|---|---|
| T0H | 0.35±0.15 | 0.4±0.15 | 0.22~0.38 | 0.3±0.15 | 0.3±0.15 | 0.3 |
| T0L | 1.36±0.15 | 0.85±0.15 | 0.58~1.6 | 0.9±0.15 | 0.9±0.15 | 0.9 |
| T1H | 1.36±0.15 | 0.8±0.15 | 0.58~1.6 | 0.6±0.15 | 0.6±0.15 | 0.8 |
| T1L | 0.35±0.15 | 0.45±0.15 | 0.22~0.42 | 0.6±0.15 | 0.6±0.15 | 0.4 |
| $T_{RESET}$ | 50 | 50 | >280 | 80 | 80 | 300 |

単位[us]

※WS2822S は、アドレス、データの 2 線での制御方式なので、本チュートリアルでは扱わない
※OST4ML8132A, OST4ML5B32A の、VDD, 1/2・VDD, 0 の 3 値の電圧で制御する方式のタイプは、2 章(RA2L1_FULL_COLOR_LED2)で扱います

シリアルデータで制御するタイプの LED のデータシートを見ると、タイミングのパラメータに振れがあるので、最大公約数的な値を設計ターゲットとします。赤字の部分はスペックから外れますが、それ以外はどのタイプのフルカラーLED でも制御可能かと思います。（実際にプログラムを作成して動かしてみると、表 1-1 の設計ターゲットで作成したプログラムで表に記載のどの LED も動作する様です）

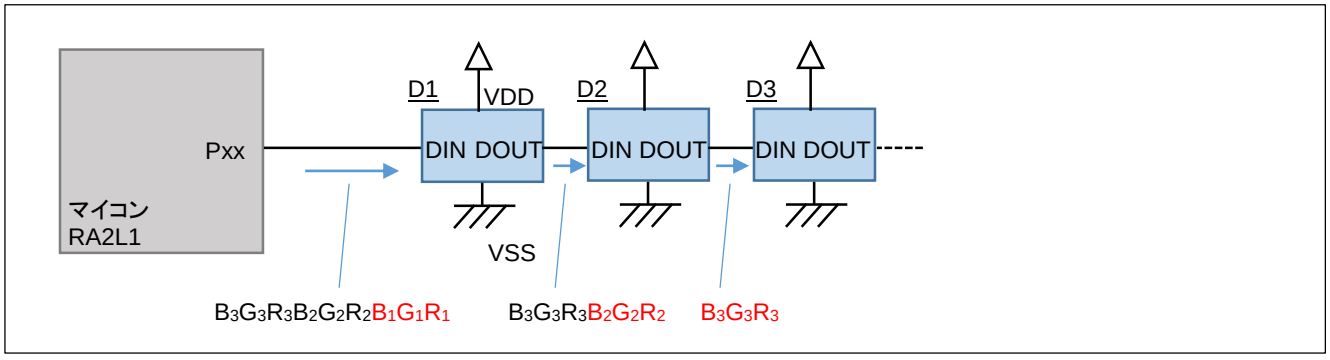



**図 1-3 フルカラーLED の接続**

フルカラーLED は、ディジーチェーン（数珠繋ぎ）で、複数のデバイスが接続される形となります。
1 つ目のデバイスが制御用マイコンと接続され、2 つ目のデバイスの入力には 1 つ目のデバイスの出力を接続します。
マイコンの I/O ポート(PA6)から出力されたデータは、先頭の 3 バイト($R_1$,$G_1$,$B_1$)が D1 のデバイスに対するデータとなり、次のデバイス(D2)には、4 バイト目以降のデータが伝播されます。以下同様の繰り返しとなります。（フルカラー

LED デバイスは、データの先頭の 3 バイトを自分自身に対するデータと認識し、4 バイト目以降は、次デバイスに対するデータと認識して、次のデバイスに渡します）

よって、マイコンから 3 つ先のデバイス(D3)に対する制御データは、マイコンから送出するデータの 7~9 バイト目にセットすれば良い事となります。

色データ（RGB コード）送信後、リセット信号（長めの L データ）を送信すると、フルカラーLED デバイスは、リセット信号を受信したタイミングで、LED の色が変更されます。

フルカラーLED デバイスは、R,G,B をそれぞれ 8bit（256 階調）の明るさで制御でき、コードと色は以下の様になります。

・3 原色 ON/OFF の組み合わせ

| R | G | B | 色 |
|---|---|---|---|
| 255 | 255 | 255 | 白 |
| 255 | 0 | 0 | 赤 |
| 0 | 255 | 0 | 緑 |
| 0 | 0 | 255 | 青 |
| 255 | 255 | 0 | 黄 |
| 255 | 0 | 255 | ピンク |
| 0 | 255 | 255 | アイスブルー |
| 0 | 0 | 0 | 消灯 |

・虹の 7 色（一例）

| R | G | B | 色 |
|---|---|---|---|
| 255 | 0 | 0 | 赤 |
| 241 | 137 | 0 | 橙 |
| 255 | 255 | 0 | 黄 |
| 0 | 255 | 0 | 緑 |
| 0 | 0 | 255 | 青 |
| 28 | 44 | 82 | 藍 |
| 92 | 42 | 134 | 紫 |

明るさを変える場合は、以下の様に

RGB=(255,255,255)　白（フルパワー）
RGB=(100,100,100)　白（中間）
RGB=(10,10,10)　白（暗い）

RGB コードの比率を変えずに、全体的に数値を小さくする事により、色調は保ったまま暗く制御できます。

フルカラーLED デバイスにより、RGB コードのデータ順が異なりますので、デバイスに応じてデータを送信する順番の変更が必要です。

表 1-2 フルカラーLED のデータ順

| | PL9823 | WS2812B<br>WS2813<br>SK6812<br>OSTW3535C1A | OST45050 |
|---|---|---|---|
| データ順 | RGB | GRB | ~~RGBW~~<br>GRBW |
| bit 数 | 24 | 24 | 32 |

データシートでは、RGBW となっていますが、実際の動作は GRBW の順です

OST45050 は、4 つの LED 素子が内蔵されており、RGB 以外に W（白色 LED）の LED を RGB とは別に設定できます。（サンプルプログラムは、24bit の LED 向けに作成されています。32bit のデータを送る場合は、定義値を変えて再コンパイルする必要があります。）

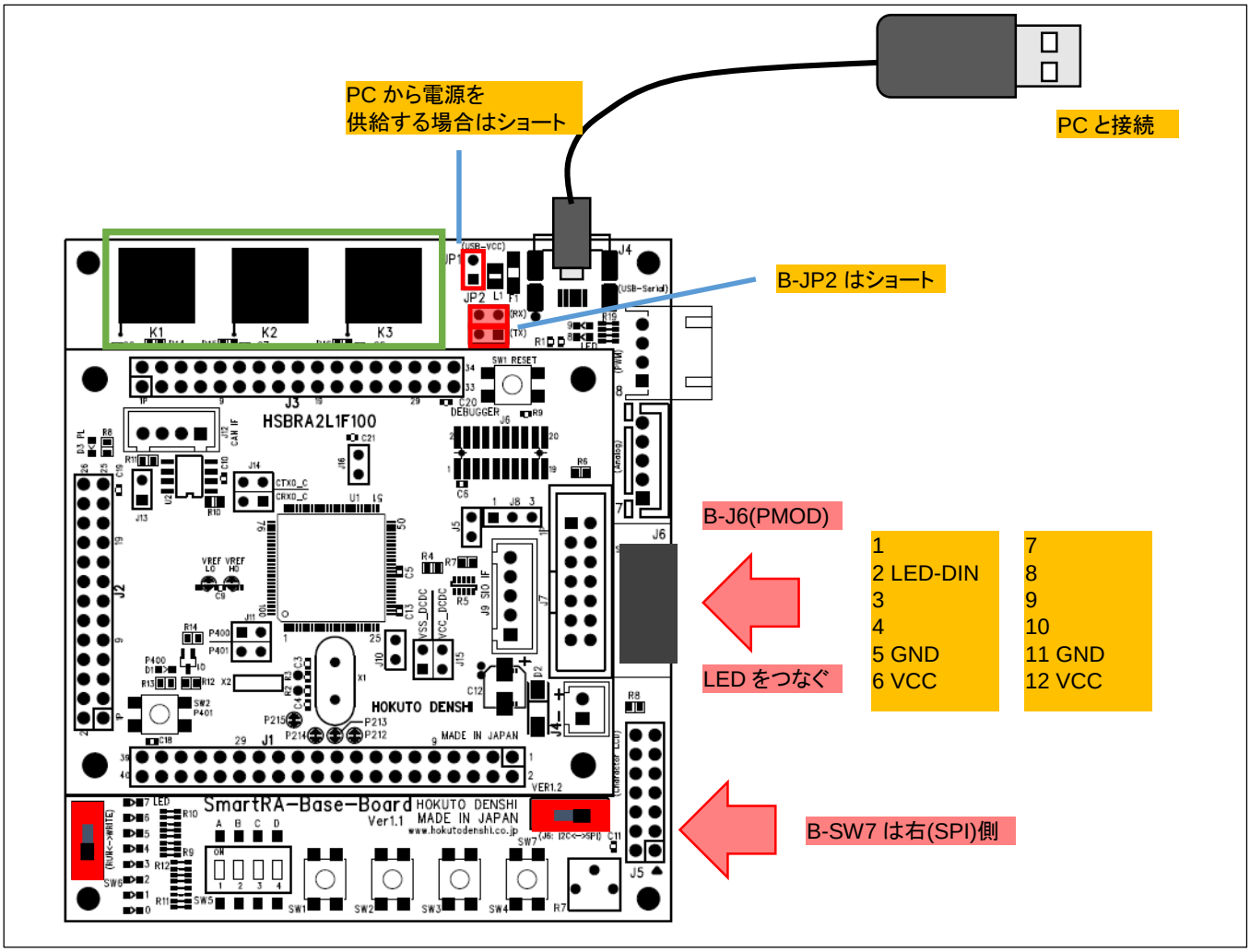
どのタイプのデバイスでも、データは MSB ファースト

MSB LSB

0x82 = 0b 1000 0010

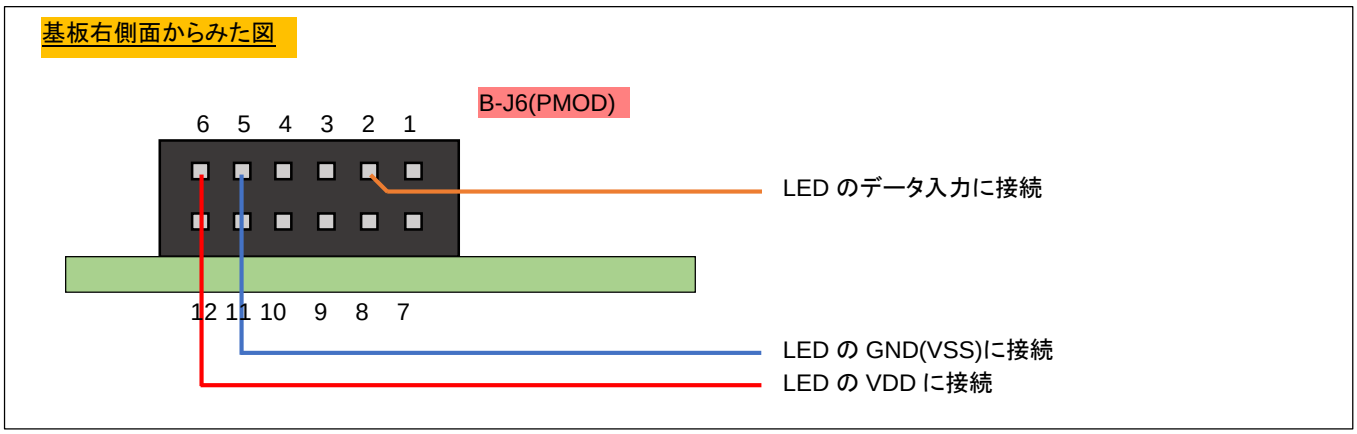
データ送信は 1 0 0 0 0 0 1 0 の順番となります。

## 1.2. ボードの設定

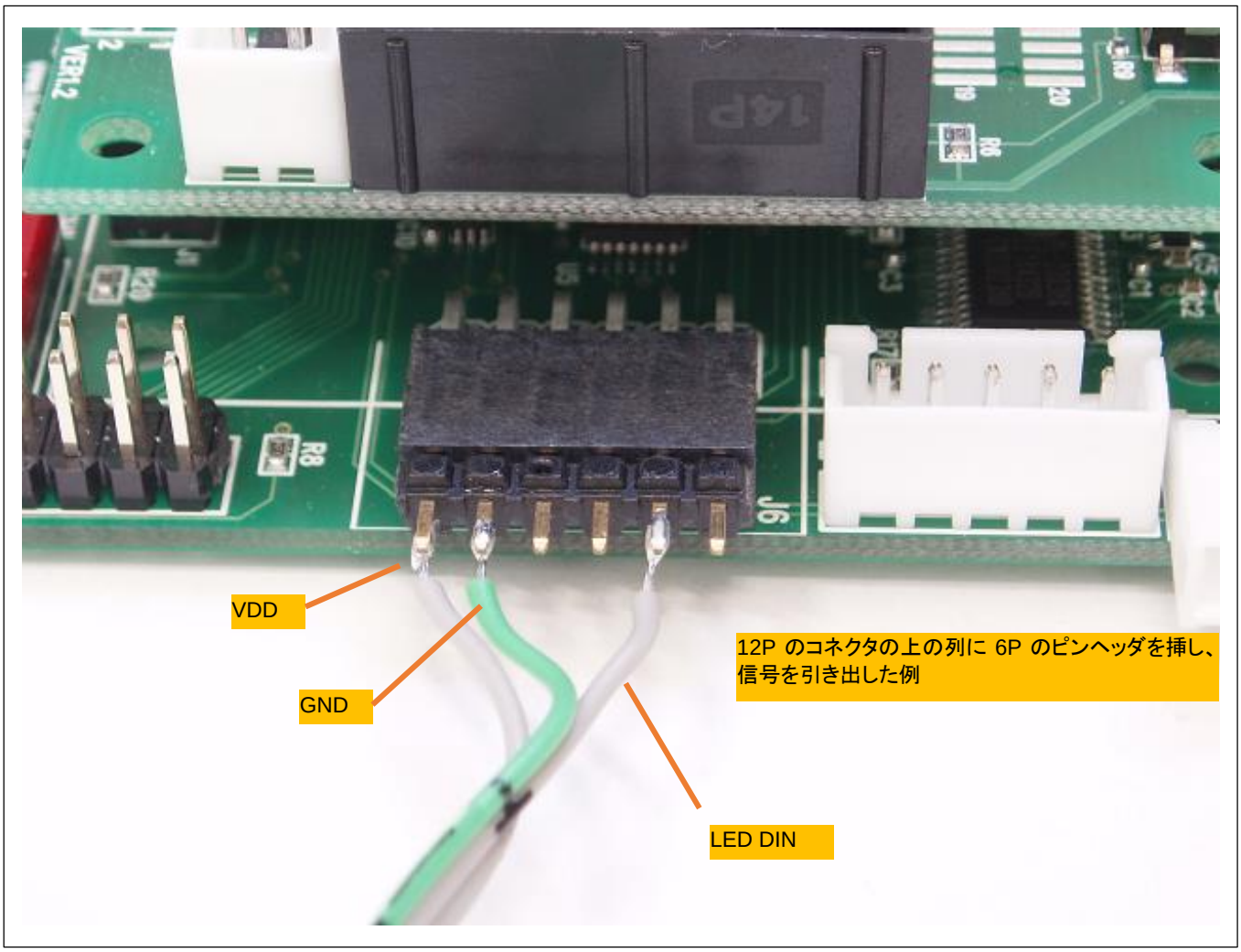


LED の電源に関しては、B-J6 から取る場合は、5-6（もしくは 11-12）から LED に供給してください。（5V を直接入力できないタイプの LED の場合は、電流制限抵抗を経由して接続してください）

2 番ピンを、LED の DIN（データ入力）に接続します。LED は、市販のデイジーチェーン（数珠つなぎ）になっているタイプの LED が使用できます。




－接続例－



－接続例－


※LED は製品には付属致しませんので、市販品を用意してください

## 1.3. サンプルプログラムの動作

起動すると、シリアル端末(115,200bps)に以下の表示が出ます。

```
Copyright (C) 2021 HokutoDenshi. All Rights Reserved.

SmartRA KIT SPI FULL COLOR LED TUTORIAL

>
```

このチュートリアルのサンプルプログラムは、端末からのコマンド入力で動作します。どのようなコマンドが使えるかは、ヘルプ(h コマンド)を入力してください。

```
>h
FullColor LED data send buffer operation usage:
  r###: r code set (###:0-255)
  g###: g code set (###:0-255)
  b###: b code set (###:0-255)
  i##: intensity code set (##:1-10, default=10)
  n##: data send num set(##:1-30, default=1)
  t##: target set(##:0-29)
  c: clear buffer
  p: print buffer
  s: send data to LED(update LED color)
>
```

LED を赤く点灯させる場合は、

```
>r255
>s
Send data to LED.
>
```

r255

s

と入力してください。

LED を青く点灯させる場合は、

```
>r0g0b255
>s
Send data to LED.
>
```

r0g0b255

s

と入力してください。

デイジーチェーン接続されている 2 つ目の LED を緑に点灯させる場合は、

```
>n2
>t1
>r0g255b0
>s
Send data to LED.
```

n2

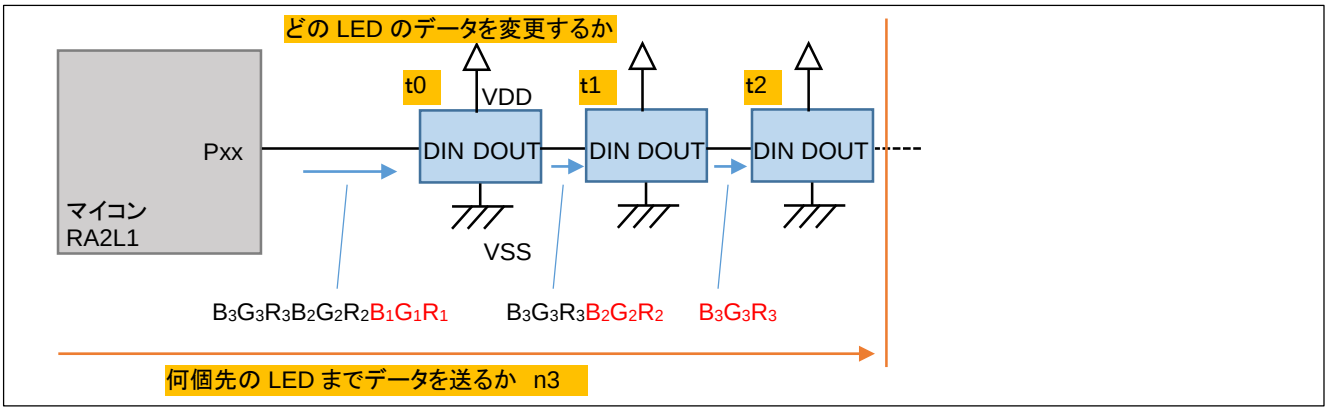
t1

r0g255b0

s

と入力してください。

基本的にコマンドは、送信バッファを編集して、s コマンドで送信するという流れとなります。



## 1.4. サンプルプログラムで使用できるコマンドの説明

・n コマンド

トータルでいくつの LED を制御するかを指定します。初期状態では、1~30 の数値を指定できます。

※30 以上の LED を制御する場合には、バッファメモリを増やして再コンパイルが必要です

[サンプル]
n5

5 個の LED（24bit×5）のデータを送信

・rgb(w)コマンド

LED の R,G,B,(W)値を変更します。

※w は、RGBW のデータを持つ LED の場合に使用します。オプションを変更して再コンパイルが必要です。

[サンプル]
r255g100b50

R,G,B = (255, 100, 50)に設定します。各数値は、0-255 までの値を指定可能です。

r255g100 とすると、b の値は以前の値から変更を行いません。RGB 全てのデータを与える必要はありません。

・t コマンド

どの LED のデータを変更するかを選びます。マイコンに近い側から、t0, t1, t2, ....です。初期状態では、0-29 の数値を指定できます。

[サンプル]
t0r255g0b0
t1r0g0b255

LED0 R,G,B = (255, 0, 0)　赤
LED1 R,G,B = (0, 0, 255)　青
に設定します。

・s コマンド

LED に対しデータの送信を行います。

このコマンドを入力した時点で、LED が変化します。

・c コマンド

バッファをオールクリアします。

```
>c
Clear buffer.
```

ターゲット LED の個数は 1 個となり、RGB=(0,0,0)設定となります。

※LED が 10 個接続されており全て消灯したい場合は、

```
>c
Clear buffer.
>n10
>s
Send data to LED.
```

c コマンドの後で、n[LED の個数]、s を入力する事で、10 個の LED に RGB=(0,0,0)（消灯）のデータを送信できます。

・i コマンド

LED の明るさを一括で制御するコマンドです。

[サンプル]

i1

s

で、全体的に、LED の明るさを 10%にする事ができます。1~10 の値を設定可能で、10 は 100%。1 は 10%です。

（r250i1 と設定するのと、r25i10 と設定するのは等価です、一括で全 LED の明るさを変えられるコマンドです。）

・p コマンド

現状のバッファの設定値を表示します。

```
>p
FullColor LED data send buffer information.
--
intensity = 10/10
data send led num = 2/30
target no : (R,G,B[,W]) (*:modify target no)
  t0 : (0,0,255)
* t1 : (0,255,0)
--
--command format--
i10n2t0r0g0b255t1r0g255b0
--
>
```

上記は、n2（2 個の LED にデータ送信。

LED0 R,G,B = (0, 0, 255)　青

LED1 R,G,B = (0, 255, 0)　緑

現在の変更対象は t1（t1 の前に*が付いている）。

事を示しています。（r,g,b コマンドは*の付いている t1 が数値変更ターゲットとなります）

表示には、--command format--の行があります。

i10n2t0r0g0b255t1r0g255b0

が、設定のサマリ（全情報）となりますので、この行をどこかにコピーしておけば、後でコマンド入力時、コマンドプロンプト

>

にペーストすることで、一括で設定の変更が可能となります。

・虹の 7 色（一例）

| t<br>ターゲット番号 | R | G | B | 色 |
|---|---|---|---|---|
| t0 | 255 | 0 | 0 | 赤 |
| t1 | 241 | 137 | 0 | 橙 |
| t2 | 255 | 255 | 0 | 黄 |
| t3 | 0 | 255 | 0 | 緑 |
| t4 | 0 | 0 | 255 | 青 |
| t5 | 28 | 44 | 82 | 藍 |
| t6 | 92 | 42 | 134 | 紫 |

上記、7 個の LED を設定、（明るさは 30%）として、p コマンドを実行すると、

i3n7t0r255g0b0t1r241g137b0t2r255g255b0t3r0g255b0t4r0g0b255t5r28g44b82t6r92g42b134s

上記文字列が得られます（端末に表示されるのは末尾の s コマンドなし）。とりあえず、LED をつなげてみて動作が見たいという時は、上記をコピー&ペーストで入力してみてください。

## 1.5. LED 制御の通信プロトコル

本チュートリアルでは、SPI(RSPI)の機能を使って、LED 制御波形を作ります。

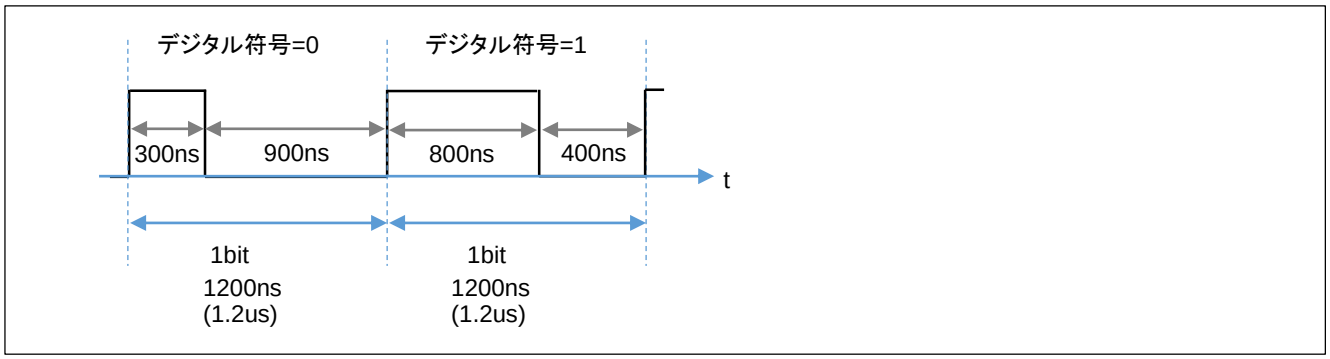



**図 1-3 作成する制御波形ターゲット**

SPI で、16Mbps に設定して例えば、0xff00(16bit)の信号を送出すると、イメージとしては、以下の様な波形となります。

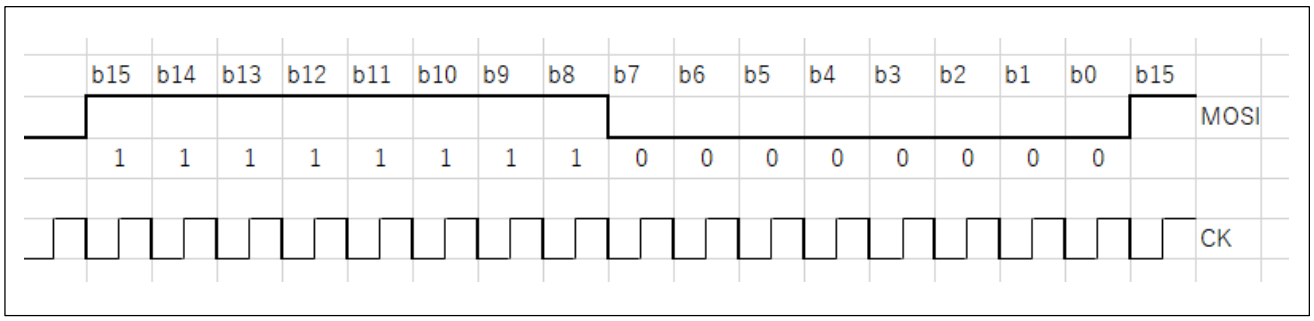



**図 1-4 SPI 出力波形イメージ**

本来 SPI は、クロック(CK)とデータ出力(MOSI)の組み合わせでデータを送信するインタフェースですが、本チュートリアルでは、クロックは使用せず、データ出力の方のみ使用する作戦です。

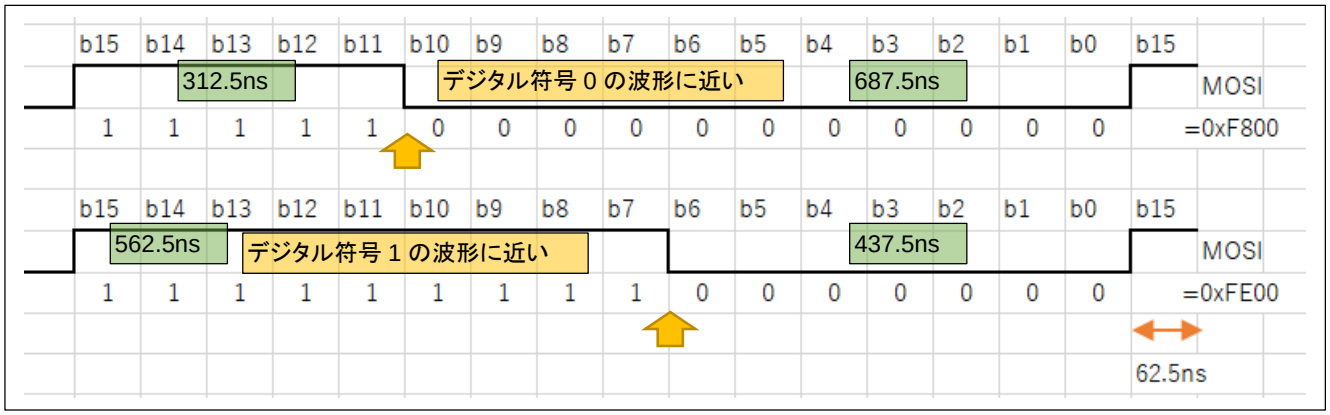



**図 1-5 SPI を使用した制御波形出力イメージ**

例えば、16bit データで、先頭 5bit を 1 後半 11bit を 0 とする。前半 9bit を 1 にして、後半 7bit を 0 にするといった様なデータを送ると、図 1-3 で想定した 2 種類の波形を作り出すことができます。

送信する 16bit のデータを変えると、図 1-5 の波形の立下り(↑)の位置を変えられるイメージです。

実際の SPI の波形では、
・クロックの立ち上がり、もしくは立下りに同期してデータ送信を行う
・16bit と次の 16bit データの間にスペースが空く
等の要素があるので、図 1-5 の通りの波形とはなりません。

SPI の送信の設定を、
・クロック立下りに合わせてデータ送信
・SPI 次アクセス遅延は 1RSPCK+2PCLKB(最小設定)
として、実際にデータを送信してタイミングをみてみます。

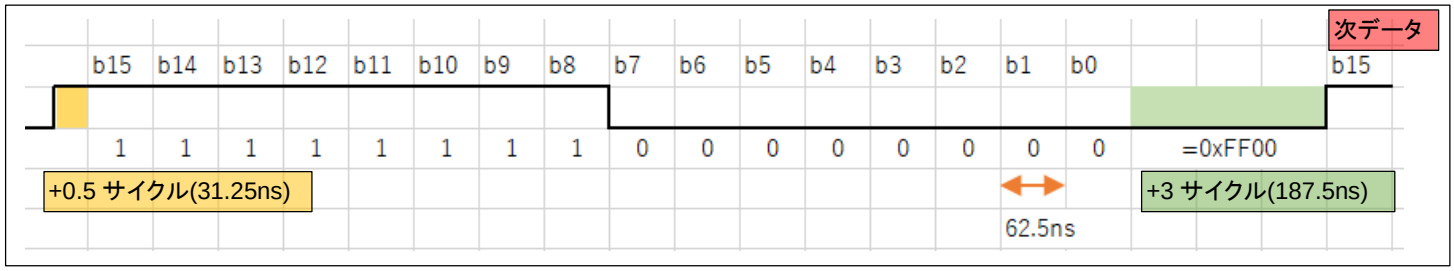



**図 1-5 SPI を使用した実際の波形出力イメージ**

実際の波形としては、前半(H データ)は+0.5 サイクル、後半(L データ)は+3 サイクル追加となります。追加分を考慮すると、以下の様になります。

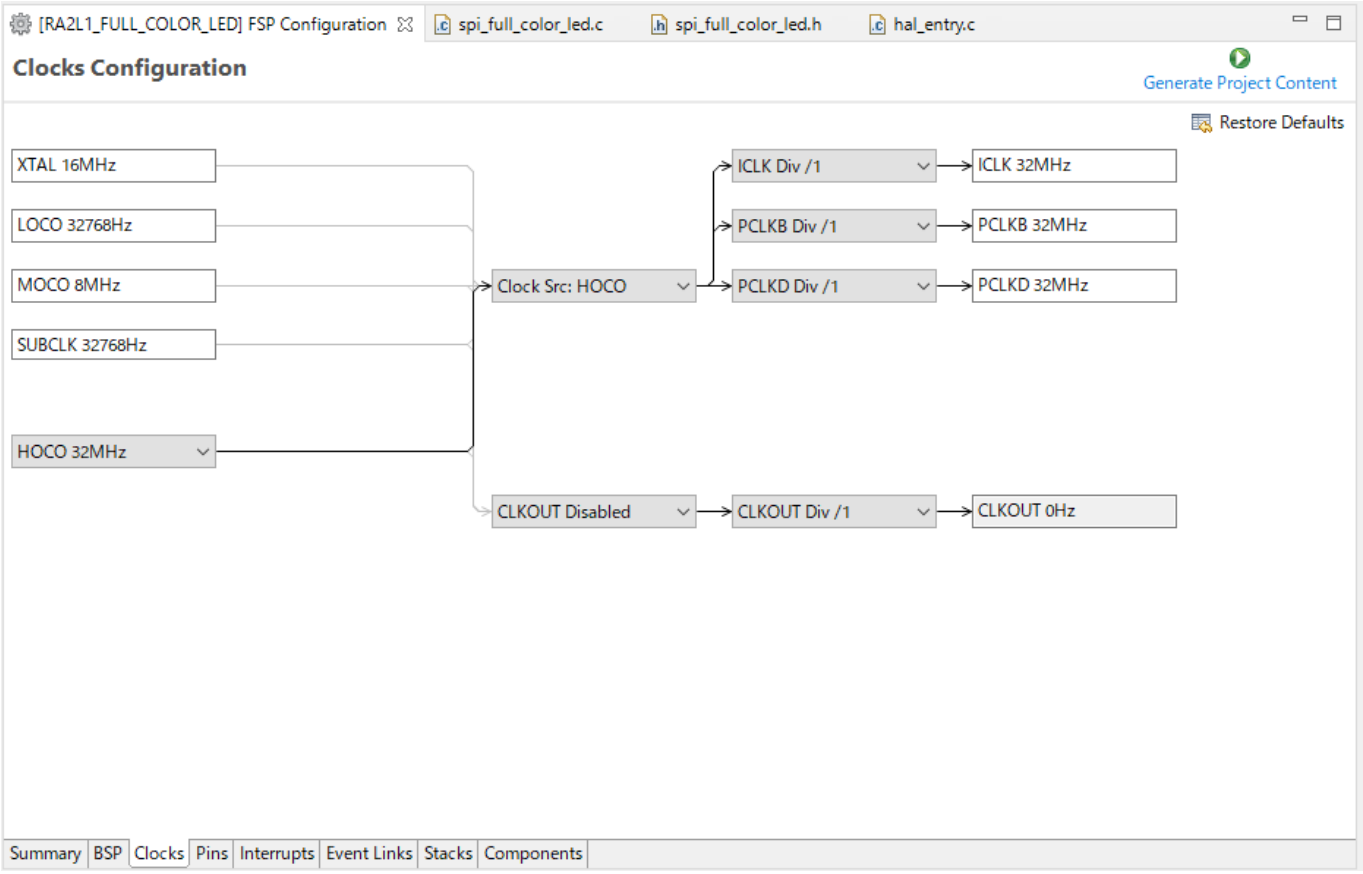
**表 1-2 フルカラーLED 制御の設定サイクル数**

| | | 目標ターゲット[ns] | サイクル数 | 設定時間[ns] | 送信データ |
|---|---|---|---|---|---|
| 0 データ | H 期間 | 300 | 4 サイクル(+0.5) | 281.25 | 0xF000 (0b1111 000000000000) |
| | L 期間 | 900 | 12 サイクル(+3) | 937.5 | |
| 1 データ | H 期間 | 800 | 13 サイクル(+0.5) | 843.75 | 0xFFF8 (0b1111111111111 000) |
| | L 期間 | 400 | 3 サイクル(+3) | 375.0 | |

例えば、フルカラーLED に、0b0110....と送信したい場合は、SPI から、0xF000 0xFFF8 0xFFF8 0xF000....の様に(16bit×送信ビット数)を送信します。(1bit のデータが 16bit に膨らむイメージです)

フルカラーLED を 1 つ制御するのに、24bit のデータが必要なので 16x24=384bit, 48byte のデータ送信を行う事となります。扱うデータ量が増えてしまうのは欠点ですが、波形切り替わりのタイミングに関しては、完全にハードウェア(マイコン)任せというのは制御する側は楽で良いです。

## 1.6. SPI を使用するための FSP の設定

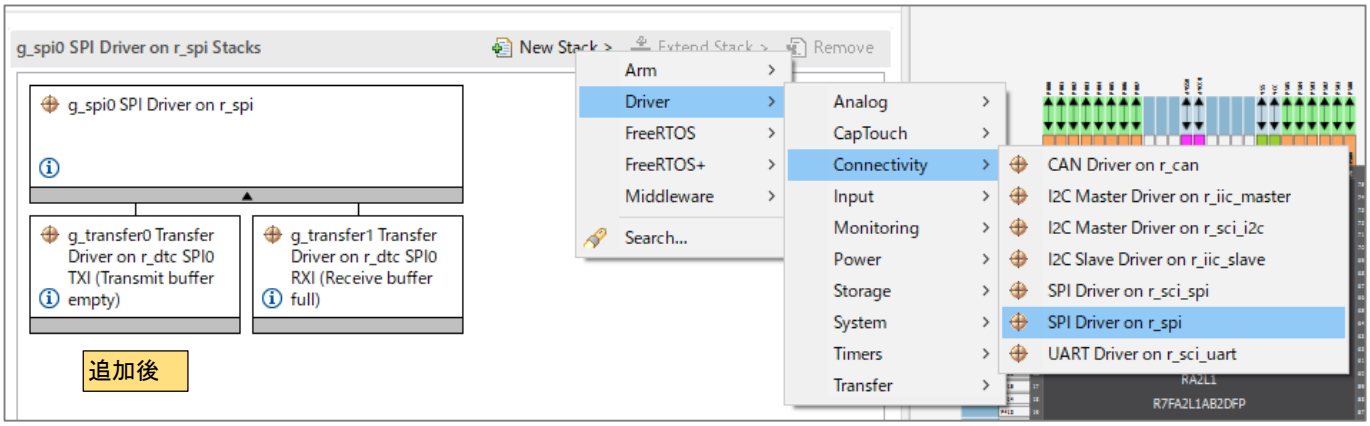



Clock タブのクロック設定は、上記としています。SPI は、PCLKB をベースに動作（一番速い設定で、PCLKB/2 がベースクロックとなる）します。16Mbpsで動かしたいので、PCLKB は 32MHz に設定しています。この場合、ICLK（CPU クロック）は、32MHz となります。RA2L1 は、48MHz で動作させることができるマイコンですが、SPI の速度設定を優先すると、CPU コアの動作速度が犠牲となります。

※ICLK=48MHz, PCLKB=24MHz として、SPI の速度を 12Mbps とし、データ長を 12bit に設定すれば、1/12MHz=83.3ns, 83.3ns×(12+3.5)=1.29us となり、

| | | 目標ターゲット[ns] | サイクル数 | 設定時間[ns] | 送信データ |
|---|---|---|---|---|---|
| 0 データ | H 期間 | 300 | 3 サイクル(+0.5) | 291.6 | 0b111 000000000 |
| | L 期間 | 900 | 9 サイクル(+3) | 999.6 | |
| 1 データ | H 期間 | 800 | 10 サイクル(+0.5) | 874.7 | 0b1111111111 00 |
| | L 期間 | 400 | 2 サイクル(+3) | 416.5 | |

上表の様に送信データを調整する事で、大体目標に近いタイミングに設定する事は可能です。CPU の速度を上げたい時はこのような手段もあります。但し、現行バージョンの FSP では、GUI の設定ではデータ長 12bit に設定できない様ですので、本チュートリアルでは、ICLK=PCLKB=32MHz に設定してプログラムを作成します。




Stacks で、

Driver – Connectivity – SPI Driver r_spi

を追加します。SPI は、r_sci_spi と r_spi があります。今回使用するのは、r_spi の方です。
（r_sci_spi は、SCI の機能を使って SPI を実現するもので、簡易的な SPI です。）

g_spi0 SPI Driver on r_spi

Settings | API Info

| プロパティ | 値 |
|---|---|
| Common | |
| Parameter Checking | Default (BSP) |
| Enable Support for using DTC | Enabled |
| Enable Transmitting from RXI Interrupt | Disabled |
| Module g_spi0 SPI Driver on r_spi | |
| Name | g_spi0 |
| Channel | 0 |
| Receive Interrupt Priority | Priority 2 |
| Transmit Buffer Empty Interrupt Priority | Priority 0 (highest) |
| Transfer Complete Interrupt Priority | Priority 2 |
| Error Interrupt Priority | Priority 2 |
| Operating Mode | Master |
| Clock Phase | Data sampling on odd edge, data variation on even edge |
| Clock Polarity | Low when idle |
| Mode Fault Error | Disable |
| Bit Order | MSB First |
| Callback | spi_callback |
| SPI Mode | Clock Synchronous Operation |
| Full or Transmit Only Mode | Transmit Only |
| Slave Select Polarity | Active Low |
| Select SSL(Slave Select) | SSL0 |
| MOSI Idle State | MOSI Idle Value Fixing Low |
| Parity Mode | Disabled |
| Byte Swapping | Disable |
| Bitrate | 16000000 |
| Clock Delay | SPI_DELAY_COUNT_1 |
| SSL Negation Delay | SPI_DELAY_COUNT_1 |
| Next Access Delay | SPI_DELAY_COUNT_1 |
| Pins | |
| MISO | None |
| MOSI | P101 |
| RSPCK | P102 |
| SSL0 | None |
| SSL1 | None |
| SSL2 | None |
| SSL3 | None |

デフォルトから変更

設定項目（主要なもの）

| | | |
|---|---|---|
| Transmit Buffer Empty Interrupt Priority | Priority 0 | データ送信のギャップができない様に優先度を上げる（念のため） |
| Clock Phase | Sampling : ODD<br>Variation: EVEN | 波形出力タイミングに影響<br>（H 期間+0.5, L 期間+3 サイクルが変わる） |
| Full or Transmit Only Mode | Transmit Only | 送信側しか使わない |
| MOSI Idle State | MOSI Idle Value Fixing Low | データ送信が終わって未送信時 L |
| Clock Delay | SPI_DELAY_COUNT1 | ディレイを最小に設定 |
| Next Access Delay | SPI_DELAY_COUNT1 | ディレイを最小に設定 |

Pins は、

| | | |
|---|---|---|
| MOSI | P101 | LED DIN に接続 |
| RSPCK | P102 | 未使用であるが割り当てないと SPI 動作しないので、割り当ては行う |

を設定します。

## 1.7. フルカラーLED 制御関数

### spi_led_init

概要：初期化関数
宣言：
　　void spi_led_init(void);
説明：
　・初期化
　・端子設定
を行います
引数：
　　なし
戻り値：
　　なし

### spi_led_data_send

概要：データ送信関数
宣言：
　　void spi_led_data_send(unsigned short led_num, rgb_data *data);
説明：
　・LED に対するデータ送信
を行います

引数:

led_num: 何個の LED にデータを送るかを指定します

*data: rgb_data 構造体の配列、送信するデータを設定

戻り値:

なし

## 1.8. フルカラーLED 制御で使用する定数値

#define LED_MAX_NUM (30) //LED30 個分のバッファを確保

データ送信バッファを 30 個の LED 制御分確保します。LED_MUX_NUM×3×8 バイトのメモリを消費するので使用可能なメモリを超えない様に注意してください。

#define LED_DATA_BYTE (3) //24bit データの LED を制御（32bit のデータの場合は 4 を定義）

RGB タイプの LED では、3。RGBW タイプの LED では、4 を定義してください。

#define RESET_TINIMG (300) //[us]

LEDのリセット時間。280usのLEDに合わせて余裕を持った時間を確保。

#define BYTE_ORDER BYTE_ORDER_GRB //OSTW3535C1A は、GRBの並び順

制御する LED のタイプに合わせて設定。
BYTE_ORDER_RGB (=0)
BYTE_ORDER_GRB (=1)
BYTE_ORDER_RGBW (=2)
のいずれか。

#define LED_BIT_0_PATTERN 0xF000 //H:4(+0.5), L:12(+3) cycle, H:281ns, L:938ns, 設計ターゲット(H:300ns, L:900ns)
#define LED_BIT_1_PATTERN 0xFFF8 //H:13(+0.5), L:3(+3) cycle, L:844ns, H:375ns, 設計ターゲット(H:850ns, L:400ns)

0 データ, 1 データの波形タイミング設定。変更により、L/H の信号幅の調整が可能です。

## 1.9. 送信データの設定とデータ送信関数

本チュートリアルでは、シリアル端末から LED に送信するデータを編集可能としていますが、プログラム内では

void spi_led_data_send(unsigned short led_num, rgb_data *data);

関数を呼び出しています。この関数に与える引数（LED に送信するデータ）は、rgb_data 構造体としています。

```
typedef struct{
  unsigned char r;
  unsigned char g;
  unsigned char b;
} rgb_data;
```

構造体のメンバは、.r, .g, .b（RGBW タイプの LED を使用するときは、.w も）です。

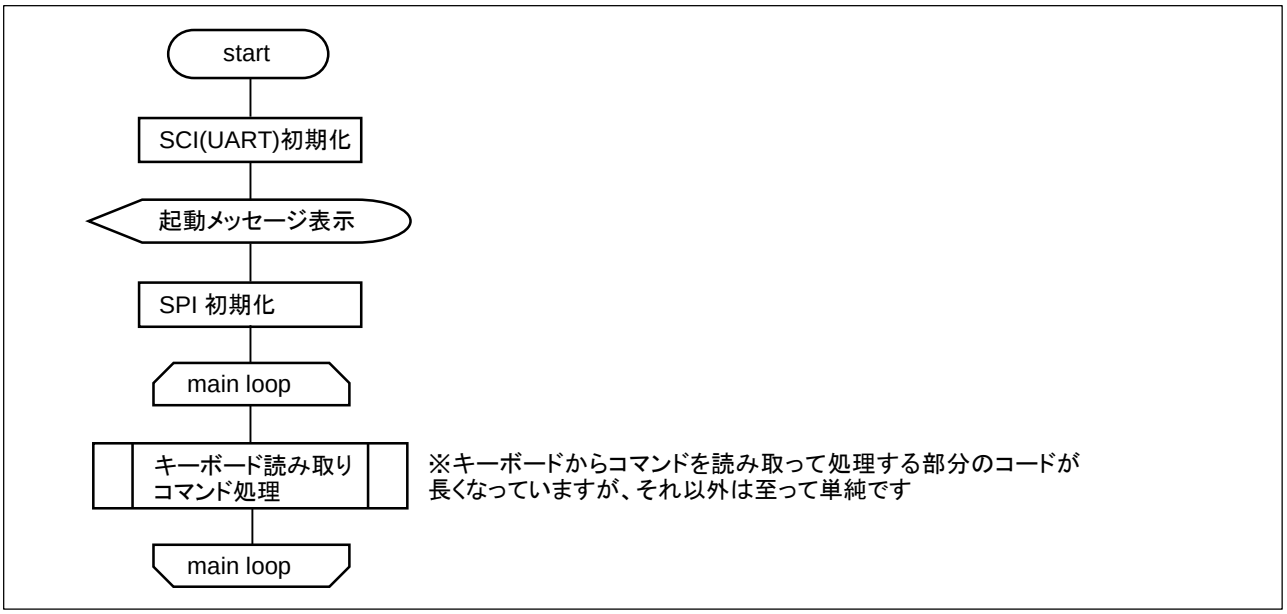
rgb_data data[30];

data[0].r = 250;
data[0].g = 100;
data[0].b = 50;

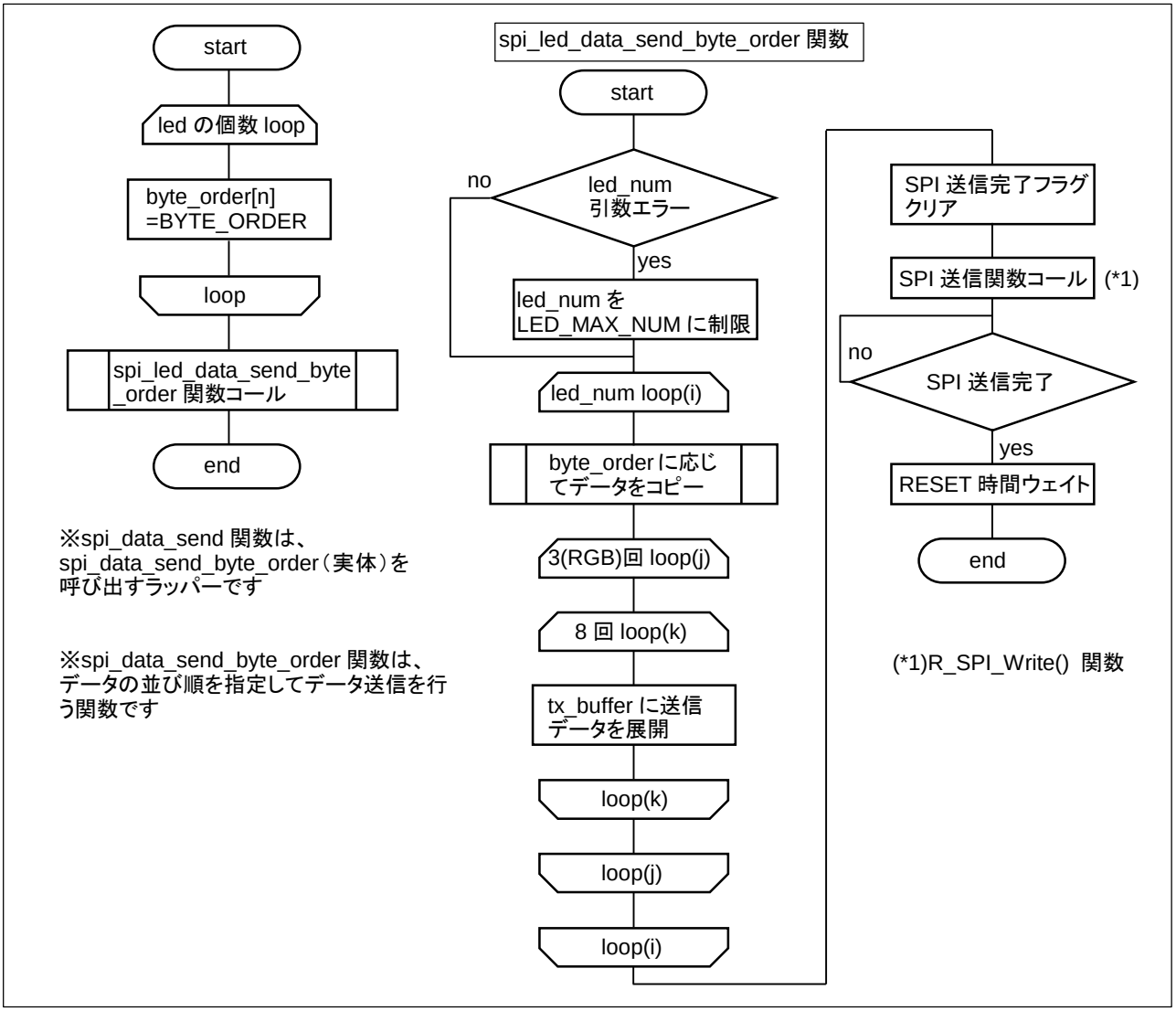
LED0（マイコンから一番近い LED）のデータを(R,G,B) = (250, 100, 50)に設定。

シリアル端末からではなく、一定時間毎に表示を変えたり、LED をスクロール表示させる場合等は、rgb_data 構造体変数に RGB のコードを格納して、spi_led_data_send()関数を呼び出してください。

## 1.10.サンプルプログラムのフローチャート




ーデータ送信関数(spi_led_data_send)フローチャートー



## 1.11.tx_buffer に展開されるデータ

プログラムでは、tx_buffer に送信データを展開後、SPI の送信関数(R_SPI_WRITE)に渡されます。

送信するデータが、0b1011....の場合（送信データは、LED の個数の 24（または 32）の倍数となります）

tx_buffer[0] = LED_BIT_1_PATTERN (0xFFF8)
tx_buffer[1] = LED_BIT_0_PATTERN (0xF000)
tx_buffer[2] = LED_BIT_1_PATTERN (0xFFF8)
tx_buffer[3] = LED_BIT_1_PATTERN (0xFFF8)
…

となります。tx_buffer は、16bit の変数で、SPI での送信も 16bit 単位で行われます。

送信する 1bit が tx_buffer では、16bit に伸長されます（LED30 個で、1440bytes のメモリを消費します…RA2L1 のメモリは 32kB です）。

※メモリの消費量という観点では CPU リソースに優しくないので、LED を多数制御したいという場合は、見直しも考慮すべきポイントです。

## 1.12.使用する LED の種類に応じた変更箇所

使用する LED の種類に応じて、プログラム（定数値）を変更して、再ビルドが必要になります。変更するファイルは、

src¥spi_full_color_led¥spi_full_color_led.h

です。

・RGB 順（PL9823 等）の LED

```
#define LED_MAX_NUM     (30)    //LED30個分のバッファを確保
#define LED_DATA_BYTE   (3)     //24bitデータのLEDを制御（32bitのデータの場合は4を定義）

#define RESET_TINIMG    (300)   //[us]

#define BYTE_ORDER_RGB   (0)
#define BYTE_ORDER_GRB   (1)
#define BYTE_ORDER_RGBW  (2)

#define BYTE_ORDER  BYTE_ORDER_RGB  //PL9823  は、GRBの並び順
//#define BYTE_ORDER  BYTE_ORDER_GRB  //OSTW3535C1A  は、GRBの並び順
//#define BYTE_ORDER  BYTE_ORDER_GRBW //OST45050  は、実際はGRBWの並び順（LED_DATA_BYTE (4)の設定も必
```

・GRB 順（OSTW3535C1A 等）の LED [デフォルト]

```
#define LED_MAX_NUM     (30)    //LED30個分のバッファを確保
#define LED_DATA_BYTE   (3)     //24bitデータのLEDを制御（32bitのデータの場合は4を定義）

#define RESET_TINIMG    (300)   //[us]

#define BYTE_ORDER_RGB   (0)
#define BYTE_ORDER_GRB   (1)
#define BYTE_ORDER_RGBW  (2)

//#define BYTE_ORDER  BYTE_ORDER_RGB  //PL9823  は、GRBの並び順
#define BYTE_ORDER  BYTE_ORDER_GRB  //OSTW3535C1A  は、GRBの並び順
//#define BYTE_ORDER  BYTE_ORDER_GRBW //OST45050  は、実際はGRBWの並び順（LED_DATA_BYTE (4)の設定も必
```

・RGBW 順（OST45050 等）の LED

```
#define LED_MAX_NUM     (30)    //LED30個分のバッファを確保
#define LED_DATA_BYTE   (4)     //24bitデータのLEDを制御（32bitのデータの場合は4を定義）

#define RESET_TINIMG    (300)   //[us]

#define BYTE_ORDER_RGB   (0)
#define BYTE_ORDER_GRB   (1)
#define BYTE_ORDER_RGBW  (2)

//#define BYTE_ORDER  BYTE_ORDER_RGB  //PL9823  は、GRBの並び順
//#define BYTE_ORDER  BYTE_ORDER_GRB  //OSTW3535C1A  は、GRBの並び順
#define BYTE_ORDER  BYTE_ORDER_GRBW //OST45050  は、実際はGRBWの並び順（LED_DATA_BYTE (4)の設定も必要）
```

LED_DATA_BYTE と BYTE_ORDER 定数を LED に合わせて変更して、プログラムをビルドしてください。（RESET_TIMING や LED_MAX_NUM も必要に応じて変更してください）

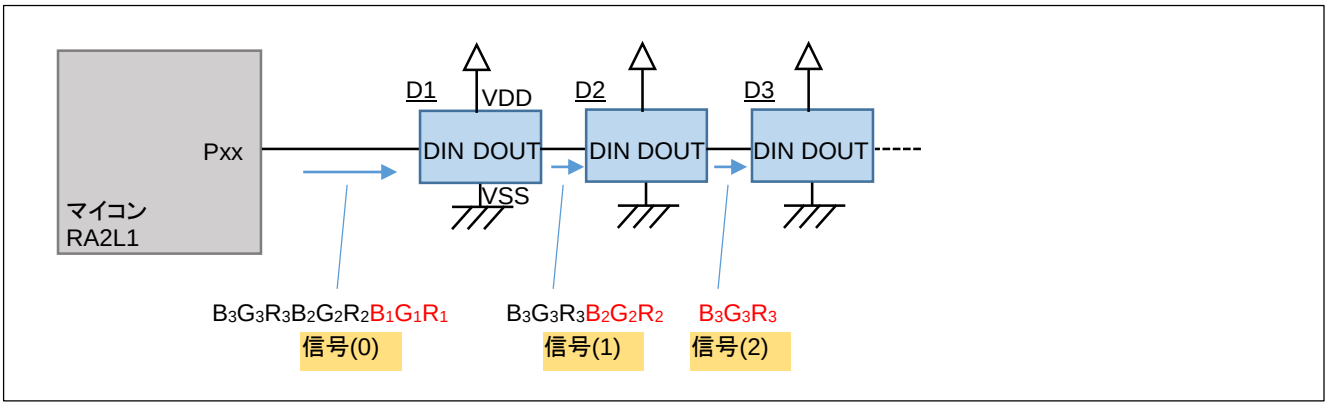
spi_led_data_send 関数は、RGB データの並び順を#define で定義しているデフォルトの並び順(BYTE_ORDER)として、spi_led_data_send_byte_order 関数を呼び出します。

なお、spi_led_data_send_byte_order 関数を直接実行しても問題ありません。RGB の並び順が異なる LED を混在して使用する場合は、spi_led_data_send_byte_order 関数を実行してください。

```
unsigned char byte_order[5] = {0, 0, 0, 1, 1};
spi_led_data_send_byte_order(5, data, byte_order);
```

で、最初の 3 個の LED は RGB 順、4,5 個目の LED は GRB 順とすることができます。（但し、LED 種の混在は上手く行かない場合もありますので、注意が必要です。後述「LED 種を混在させる場合の注意点」を参照のこと。

## 1.13.LED 種を混在させる場合の注意点




基本的には、D1, D2, D3, ...は同じタイプの LED が望ましいです。

| | PL9823 | WS2812B | WS2813<br>OSTW3535C1A | SK6812 | OST45050 |
|---|---|---|---|---|---|
| T0H | 0.35±0.15 | 0.4±0.15 | 0.22~0.38 | 0.3±0.15 | 0.3±0.15 |
| T0L | 1.36±0.15 | 0.85±0.15 | 0.58~1.6 | 0.9±0.15 | 0.9±0.15 |
| T1H | 1.36±0.15 | 0.8±0.15 | 0.58~1.6 | 0.6±0.15 | 0.6±0.15 |
| T1L | 0.35±0.15 | 0.45±0.15 | 0.22~0.42 | 0.6±0.15 | 0.6±0.15 |

上図の信号(1)は、D1 の LED の内部クロック基準で再生中継（D1 が受信した信号を D1 のクロック基準で再送信）するので、信号(0)と信号(1)ではタイミング（L/H の幅）が変わります。

・OSTW3535C1A を通したタイミングの変化

| | | 信号(0)<br>設定時間[ns] | 信号(1)<br>実測値[ns] |
|---|---|---|---|
| 0 データ | H 期間 | 281.25 | 288 |
| | L 期間 | 937.5 | 912 |
| 1 データ | H 期間 | 843.75 | 568 |
| | L 期間 | 375.0 | 648 |

例えば、信号(0)で PL9823 を駆動した際には正しく制御可能ですが、信号(1)で PL9223 を駆動するとどのようなデータを送っても、PL9823 は全て 0 と認識します。（信号(1)で OSTW3535C1A を駆動した場合は正しく制御されます。）

LED 種を混在させる事自体は問題ではないのですが、タイミングパラメータが異なる LED を混在させた場合には、手前の LED のタイミングに引きずられますので注意してください。

なお、LED を通すことによる信号の遅延は、OSTW3535C1A では実測で 1 段あたり 160ns 程度でした。（信号を受け取った直後に次へ流す感じです。）

# 2. RA2L1_FULL_COLOR_LED2

別なプロトコルの、フルカラーLED の制御を行うチュートリアルです。

## 2.1. 本章で扱う LED の制御信号

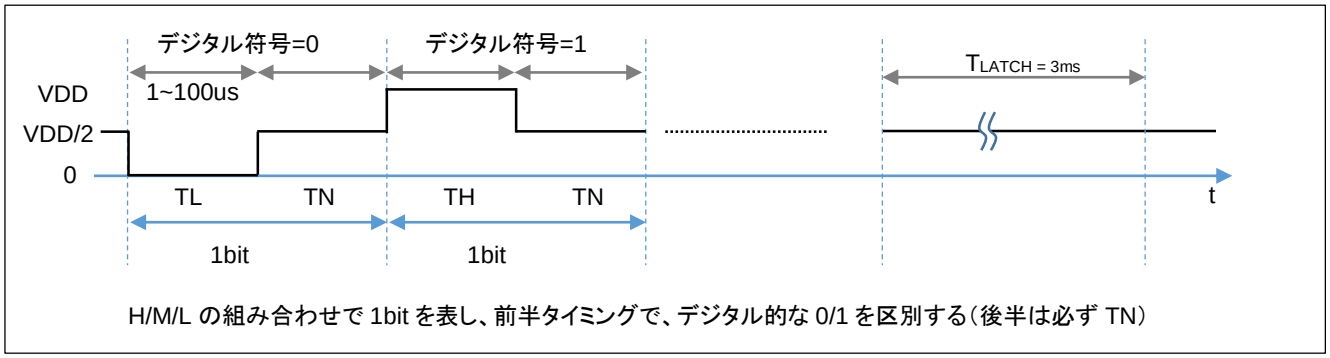



**図 2-1 フルカラーLED の制御波形（電圧 3 値タイプ）**

VDD(5V), VDD/2(2.5V), 0(0V)の 3 値の電圧で信号を受信します。1 章で扱ったタイプは、最短で 300ns 程度のタイミングが要求されましたが、3 値で制御するタイプは、パルス幅が 1~100us と幅があるのでタイミング的には楽です。但し、マイコンから 3 値を出力するのは多少面倒です。チュートリアル 6 で扱った、D/A コンバータを使用すると、3 値の電圧を出力可能ですが、ここではもっと単純な方法を使います。

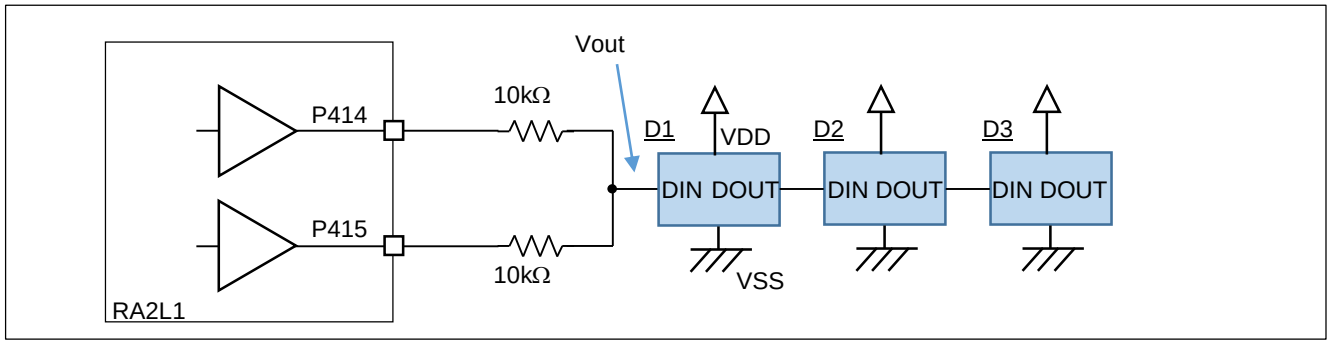



**図 2-2 3 値の電圧を生成する方法**

2 つの汎用 I/O 出力を使い、2 本の端子出力を抵抗 2 本で接続します。

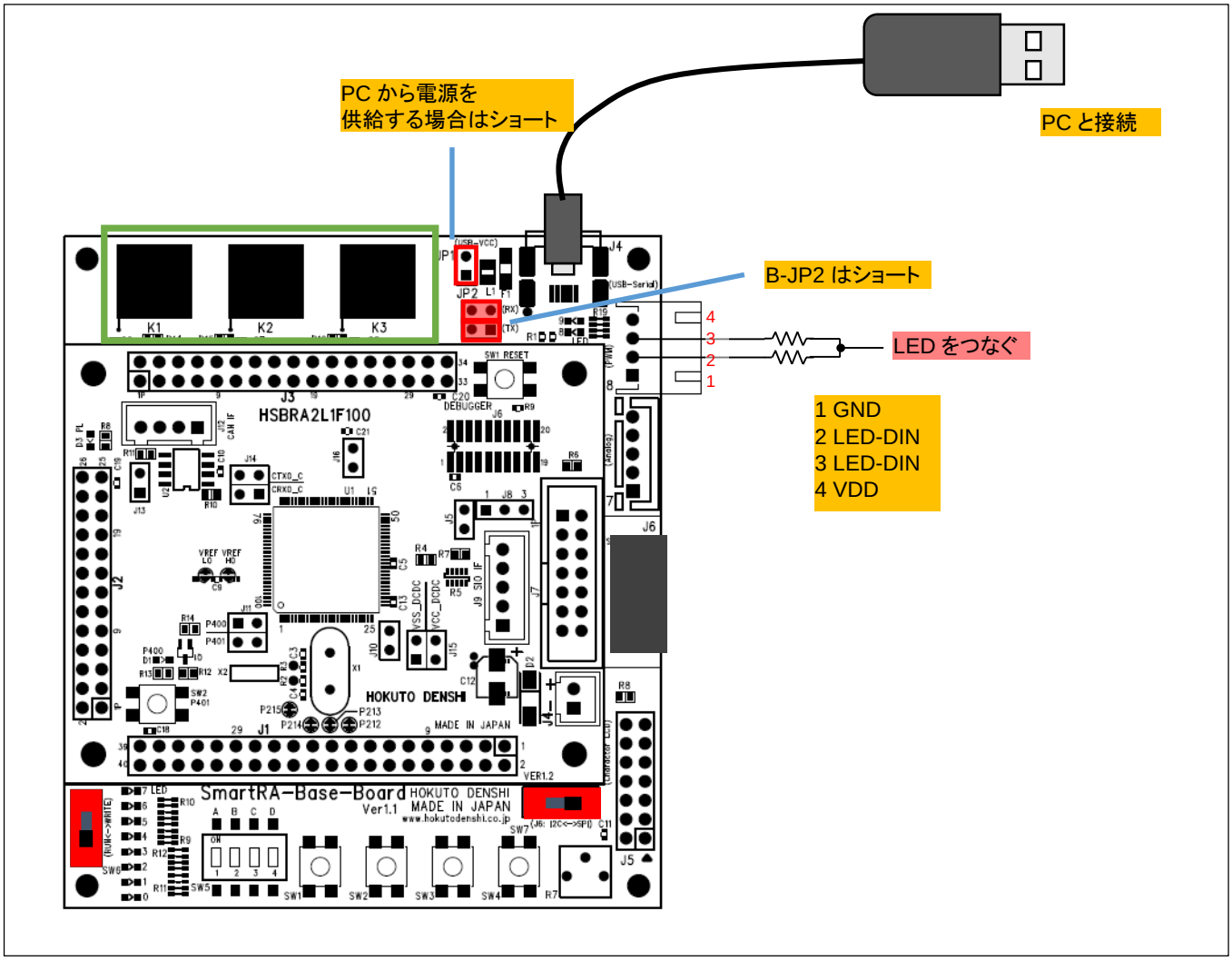
－真理値表－

| P414 | P415 | Vout | 符号 |
|---|---|---|---|
| L | L(*1) | 0V | TL |
| L | H | 2.5V | TN |
| H(*1) | H | 5V | TH |

(*1)Hi-Z でも可

2 本の出力を逆（L と H）で駆動すると、Vout は、1/2VDD となります。両方 H とすると Vout=VDD，両方 L とすると、Vout=0 となります。

## 2.2. ボードの設定




## 2.3. プログラムの動作

1 章と全く同じです。コマンドで、RGB のコードを LED に送信します、

## 2.4. 対象 LED のパラメータ

表 2-1 フルカラーLED のタイミングパラメータ

| | OST4ML5B32A<br>OST4ML8132A | 設計<br>ターゲット |
|---|---|---|
| TH | 1~100[us] | 10[us] |
| TN | 1~100[us] | 10[us] |
| TL | 1~100[us] | 10[us] |
| $T_{LATCH}$ | 3[ms] | 3[ms] |

表 2-2 フルカラーLED のデータ順

| | OST4ML5B32A<br>OST4ML8132A |
|---|---|
| データ順 | ~~RGB~~<br>BGR |
| bit 数 | 24 |

データシートでは、RGB となっていますが、実際の動作は BGR 順です

このタイプのデバイスは、データは LSB ファースト

MSB LSB

0x82 = 0b 1000 0010

データ送信は 0 1 0 0 0 0 0 1 の順番となります。

波形切り替わりのタイミングは 10us 周期とします。(10us に決めた理由は後述)

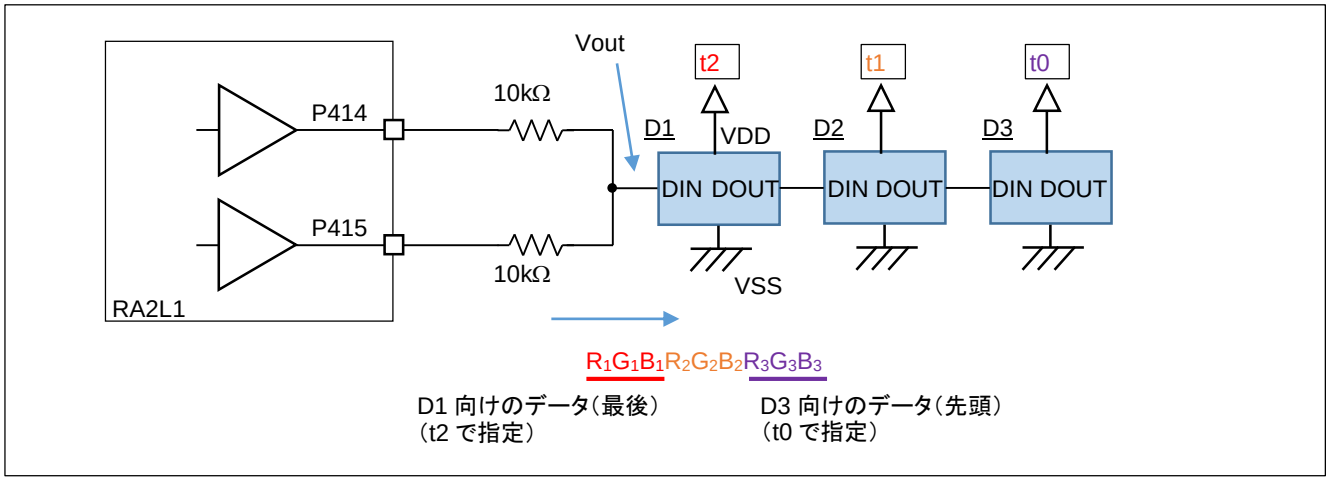



図 2-3 データの送信順

1 章で扱った LED とは逆で、最初に送出したデータがマイコンから一番遠い LED に伝わります。このタイプの LED はラッチ信号(TN の長い周期の信号)が来ると、その前のデータを自分宛のデータとしてラッチします。

2 章のタイプの LED で、1 章と異なる点は、以下となります。

・t(ターゲット番号)が逆となる

マイコンから一番遠い LED から、t0, t1...となります。

※プログラムでは最初に送出するデータが t0 で指定したデータです

・n コマンドで LED の個数を正しく設定する事が必要

→1 章の LED では、LED が 5 個つながっている場合、n=3 を指定して"s"コマンドでデータ送信した場合、マイコンから近い 3 個の LED に信号が伝播される(4 個目以降の LED にはデータが送られない)という挙動でしたが、2 章の LED は最後に送られるデータがマイコンから一番近い LED に送信されます。つながっている LED の個数と、n コマンドで指定した個数が異なると、ちょっとおかしな挙動となります。

## 2.5. フルカラーLED 制御方式

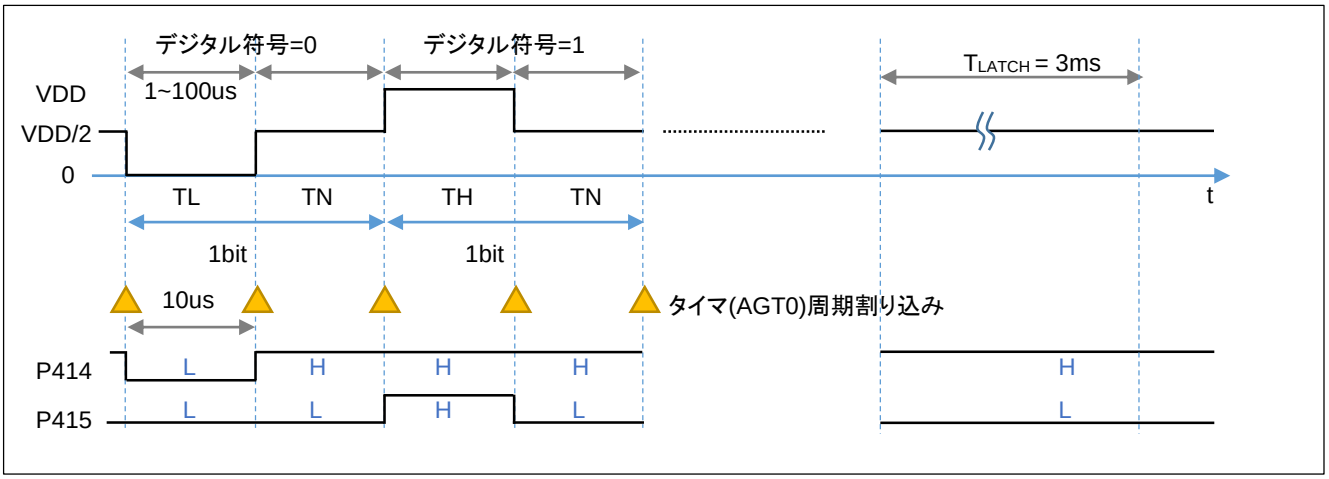



**図 2-4 波形制御タイミング**

一定周期(10us)毎にタイマ割り込みを掛け、割り込みのタイミングで次の符号を送る様にします。1bit 送るのに、2 周期(20us)掛かります。ここで、割り込みの基準時間を 10us に決めた理由ですが、単純に一連の処理にかかる時間から逆算したものです。割り込みで汎用 I/O ポートを叩く(切り替える)時間が、コンパイラの最適化オプションデフォルト(-O2)で、4.7us。最適化 OFF(-O0)で、8.8us 掛かりましたので、それより多少長い時間としました。

本プログラムでは、タイマーは AGT を使っています。AGT は 16 ビットのタイマーで、2ch あります。サブクロック(32.768kHz)をクロックソースに選択できたり、2ch のタイマーをカスケード接続できたりする点が、GPT と異なります。(一定時間毎に信号を切り替える用途では、GPT, AGT のどちらを使っても大差ないと思います。GPT は、モータ制御で使ったことがありますので、ここでは AGT を使う事とします。)

サンプルプログラムでは、

・FSP で設定した AGT0（デフォルト）
・設定を書き下した AGT1

を選んで使用できるようになっています。

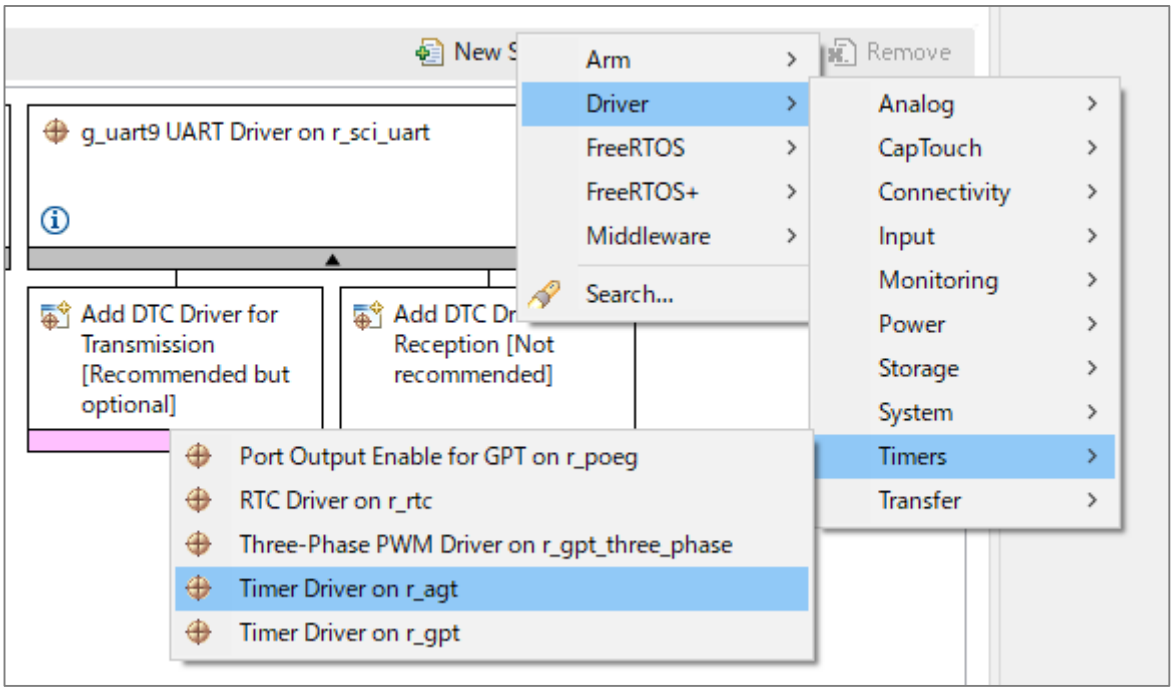
src¥agt_full_color_led¥agt_full_color_led.h

```
//#define USE_AGT1    //AGT1 を使う場合定義
```

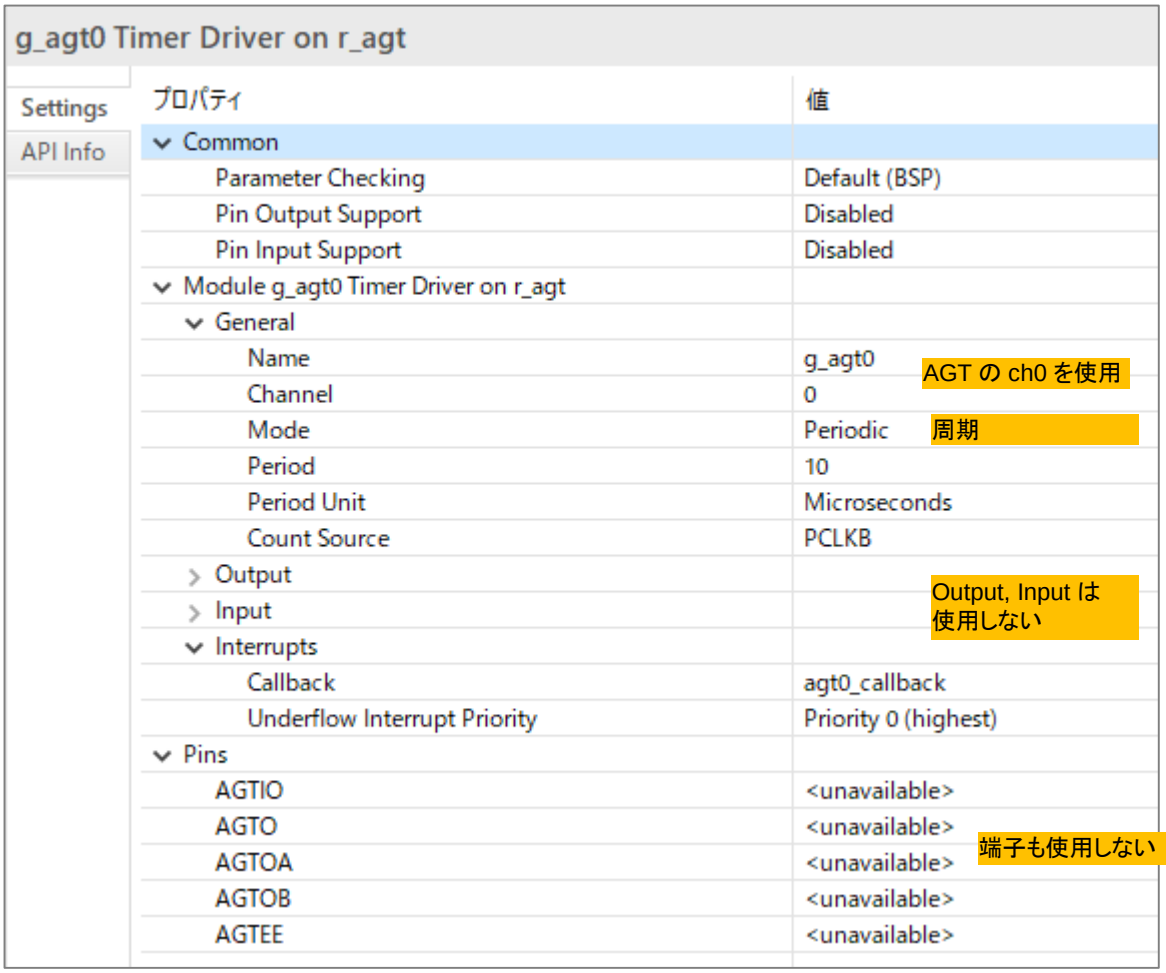
上記行のコメントアウトを外した場合、AGT1 が使われます。（デフォルトは、AGT0 です）

## 2.6. FSP の設定

### 2.6.1. AGT0(FSP を使った設定）




New Stacks – Driver – Timer – Timer Driver on r_agt を追加。



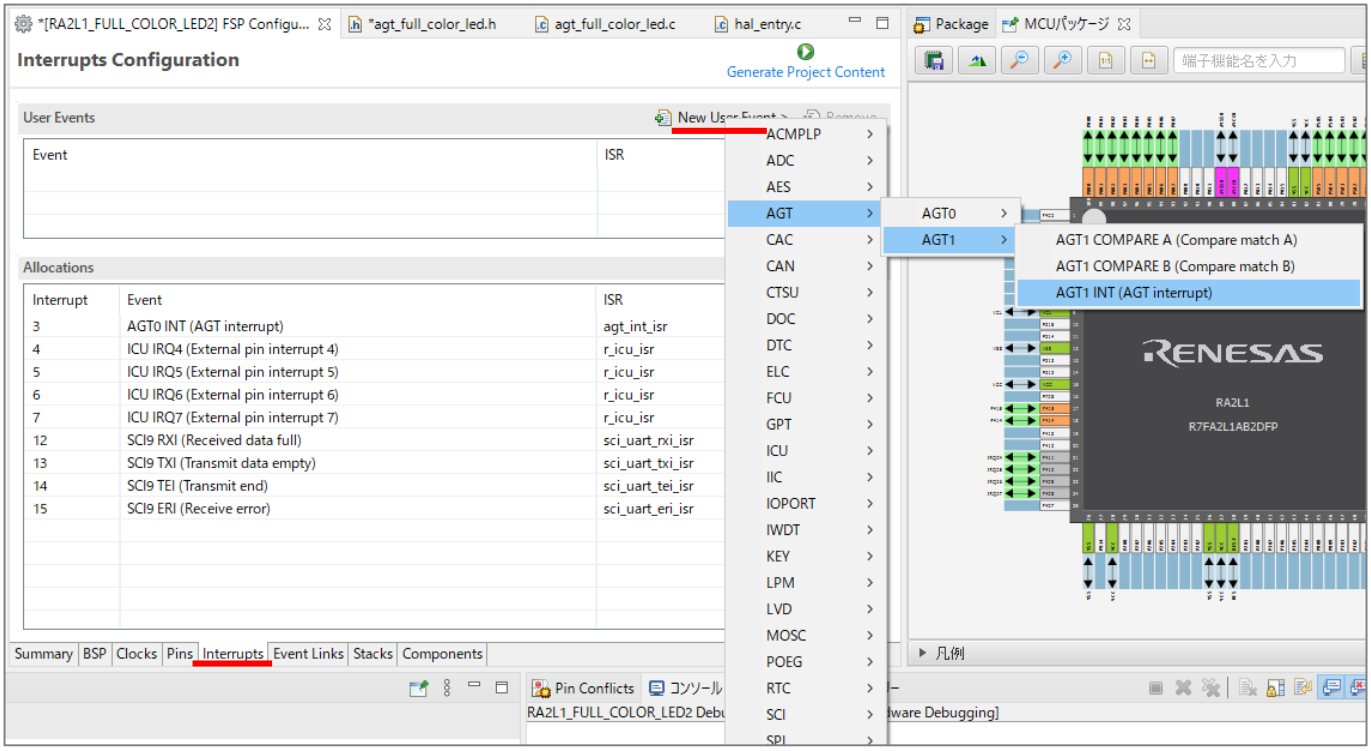

設定項目（主要なもの）

| | | |
|---|---|---|
| Mode | Periodic | 周期タイマ |
| Period | 10 | 10[us]を指定 |
| Period Unit | Microseconds | |
| Callback | agt0_callback | コールバック関数を定義 |
| Underflow Interrupt Priority | Priority 0(hishest) | 優先度は最高に設定 |

上記の様に設定して、タイマをスタートすると、10us 毎に
agt0_callback()
関数が呼ばれますので、この関数内で TH/TN/TL の切り替えを行っていけば良いです。

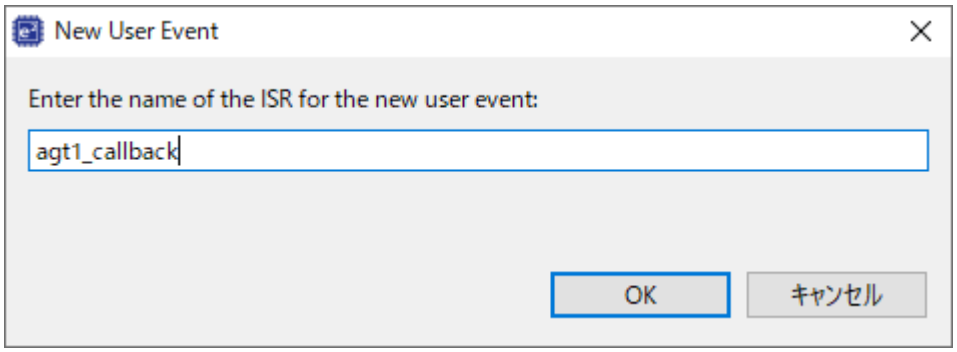
## 2.6.2. AGT1

AGT1 は、プログラムコードを書き下す事とします。但し、割り込み番号の設定は、FSP を使って設定している他の割り込みと割り当てが重複しない様、FSP で割り込み番号を確保します。

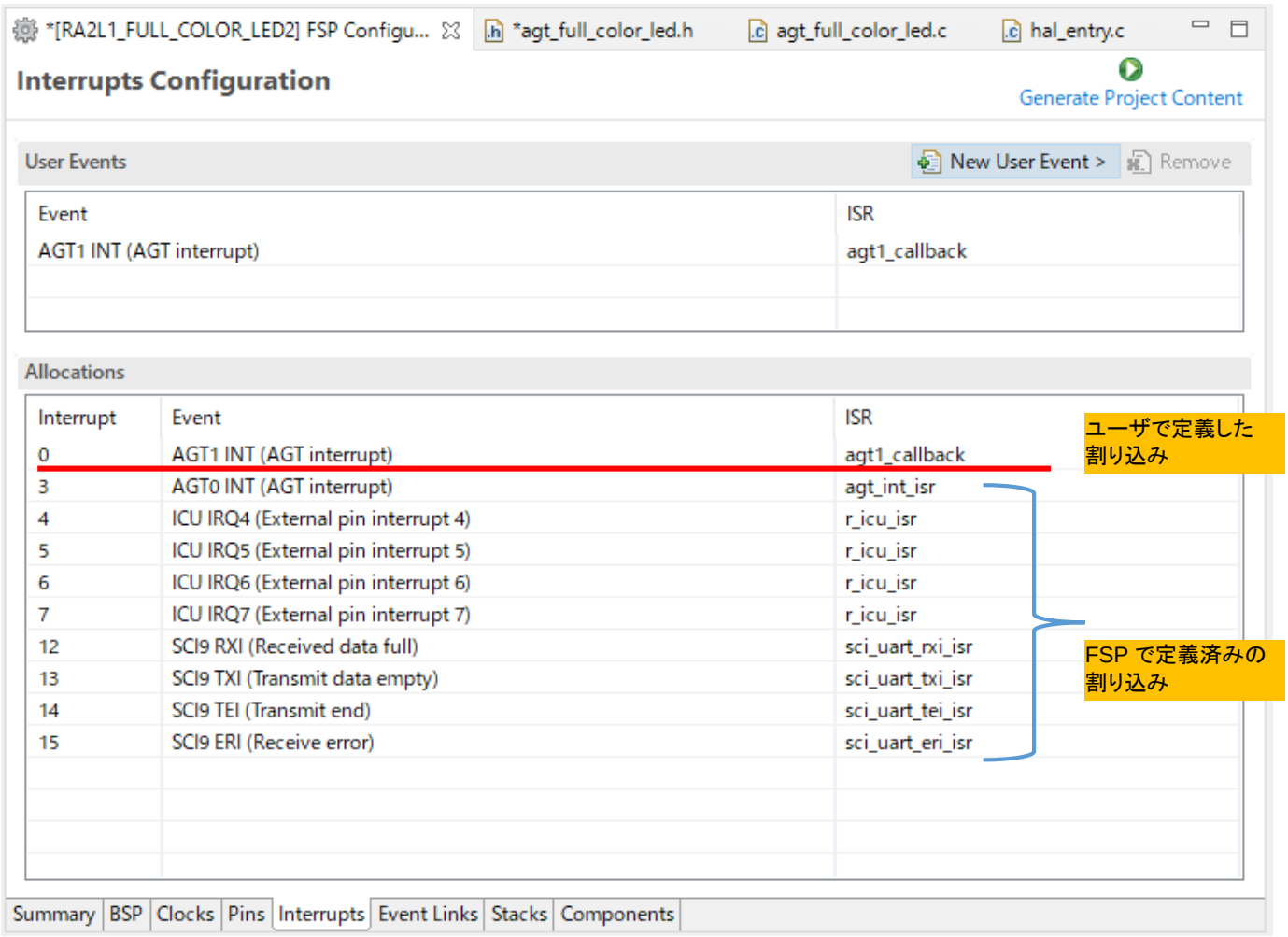



Interrupt タブ

New User Event – AGT – AGT1 – AGT1 INT(AGT Interrupt)を追加




名称として、「agt1_callback」を定義。（ここで入力した名称が、割り込み関数の関数名となります）




追加後は、上記の様になります。ここで、Interrupt の番号（0 番）は、プログラムで使用するので、何番に割り当てられたかは、気にしておいてください。（定義済みの番号と重ならない様に割り振られますので、AGT1 の割り込みを追加した際、0 番以外の割り込み番号に割り振られることもあります。）

なお、RA2L1 は、グループ割り込みを使うマイコンで、AGT1 INT は
グループ 0（割り込み番号、0, 8, 16, 24）
グループ 4（割り込み番号、4, 12, 20, 28）
のいずれかに設定可能です（上記以外の番号に割り振られる事はありません）。

（RA6 等のマイコンは、定義順に若い番号に割り振られます。RA2L1, RA2E1 は、番号の割り当てがグループ化されていて、グループ内の番号に割り振られます（割り振る必要があります）。）

## 2.7. フルカラーLED 制御関数

spi_led_init　→　agt_led_init　：初期化関数

spi_led_data_send　→　agt_led_data_send　：データ送信関数

1 章で定義した関数の spi が agt に変わります。プログラムの動作、引数の与え方等は同様です。

## 2.8. フルカラーLED 制御で使用する定数値

※変更箇所のみ記載

```
#define DOUT_H_PORT        R_PORT4->PODR_b.PODR14
#define DOUT_L_PORT        R_PORT4->PODR_b.PODR15
```

H 出力ポートを、P414, L 出力ポートを P415 に設定。
（P414, P415 を出力ポートに設定するのは、FSP の端子設定で行ってください）

```
//#define USE_AGT1        //AGT1 を使う場合定義
```

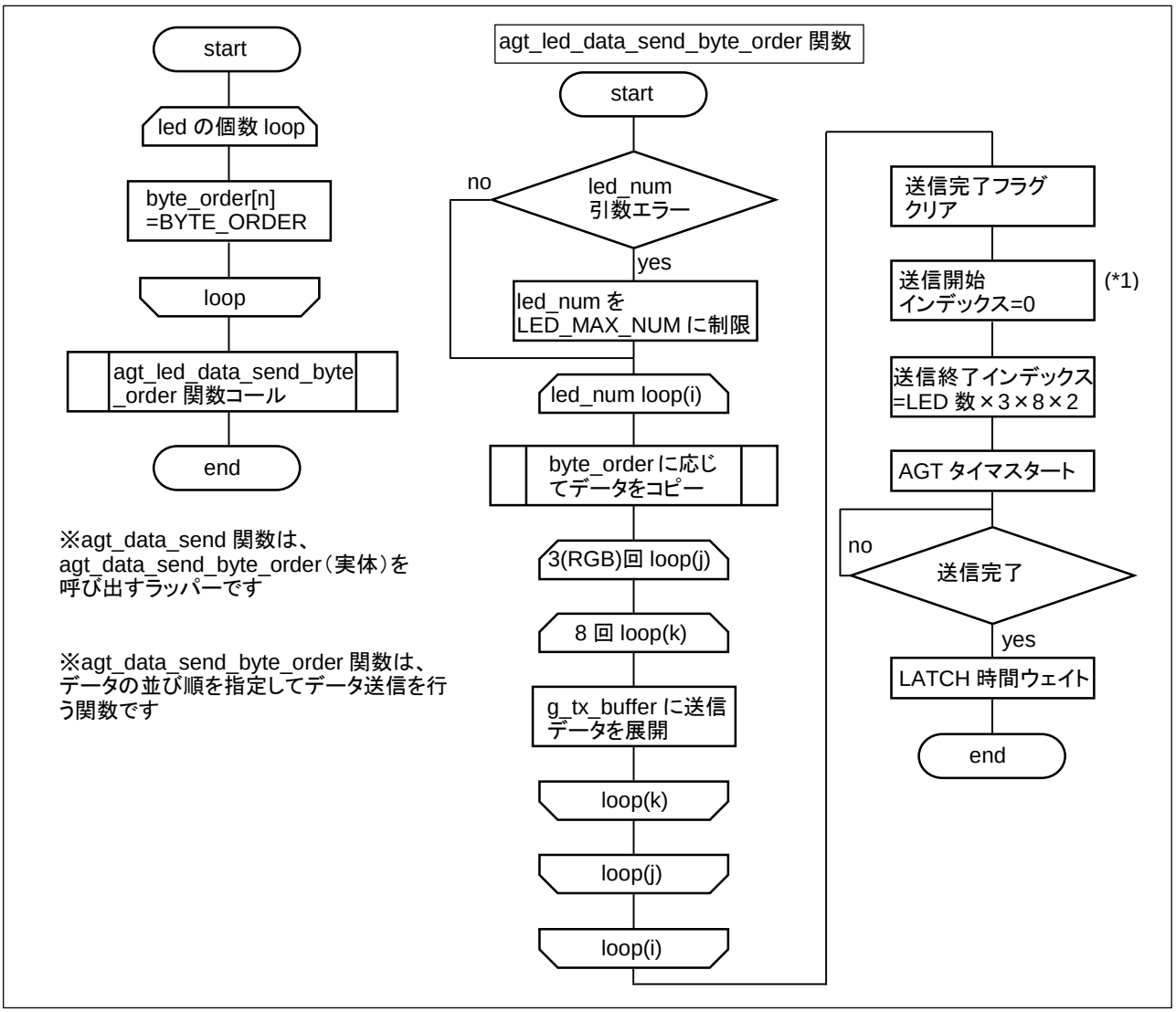
コメントアウトを外すと、AGT1 を使います。デフォルトは、AGT0 です。

```
#define INTR_AGT1_AGTI        0       //割込みベクタ
```

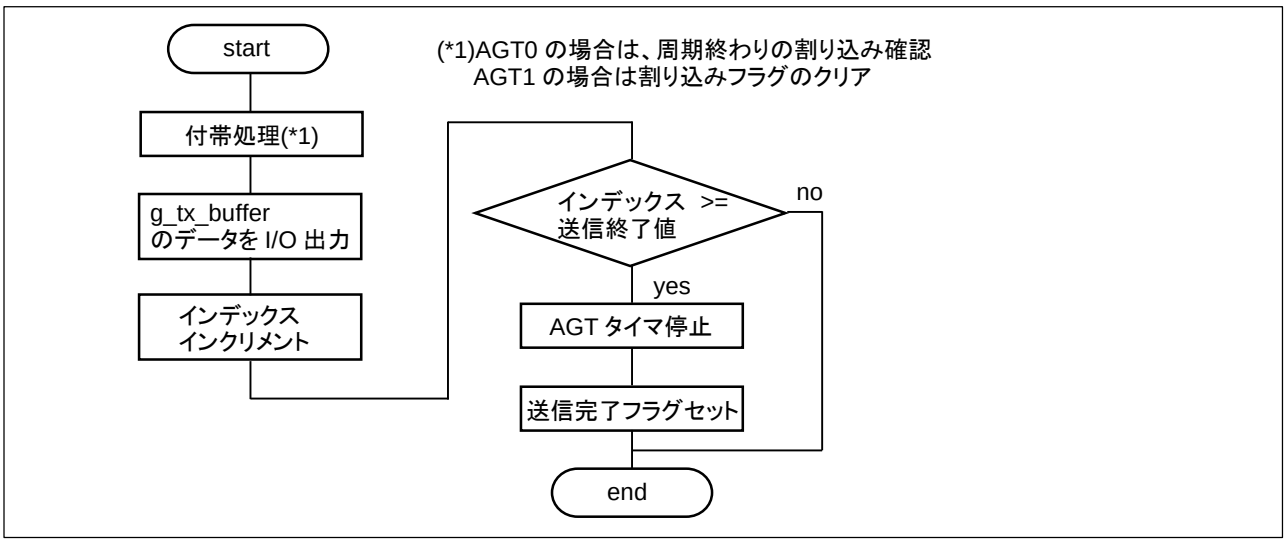
AGT1_AGTI の割り込み番号を定義します。FSP の割り込み設定で割り当てられた番号を定義してください。

## 2.9. サンプルプログラムのフローチャート

－データ送信関数(agt_led_data_send)フローチャート－




－AGT 割り込み関数 agt0_callback, agt1_callback－



## 2.10.g_tx_buffer に展開されるデータ

プログラムでは、g_tx_buffer に送信データを展開後、AGT 割り込み関数内で処理されます。

送信するデータが、0b1011....の場合（送信データは、24 の倍数となります）

g_tx_buffer[0] = LED_TH (1)
g_tx_buffer[1] = LED_TN (0)
g_tx_buffer[2] = LED_TL (-1)
g_tx_buffer[3] = LED_TN (0)
g_tx_buffer[4] = LED_TH (1)
g_tx_buffer[5] = LED_TN (0)
g_tx_buffer[6] = LED_TH (1)
g_tx_buffer[7] = LED_TN (0)
…

となります。tx_buffer は、8bit の変数で、1, 0 -1 の値が展開されています。

このタイプの LED は、LED_TN(1/2×VDD)を挟む形になりますが、LED_TN のデータも g_tx_buffer に展開されています。

これは、タイマ割り込みの処理をできるだけ軽くしたかったので、この様な構成にしています。

3 値で制御するタイプの LED も 30 個で、1440(+1)bytes(*1)のメモリを消費します（1 章のタイプと同じだけのメモリを消費します）。

・8bit の変数に 3 値の情報を保持している
・奇数インデックスのデータは必ず LED_TN の状態(*2)

という事となっているので、単純に(*2)のデータを削除して LED_TH, LED_TL のデータの後に、LED_TN のデータを AGT の割り込み関数側で付与するという手法も簡単に実現できるかと思います。また、LED_TH, LED_TL の 2 値のデータとしてしまえば、8bit の変数×3 に LED1 つ分の情報を格納する事もできます。

3 値タイプの制御では、使用メモリがネックになる場合は、比較的単純な手法で大きく削減する事も可能です。

(*1)で(+1)しているのは、タイマ周期を、タイマ割り込み内で掛かる時間よりも短く設定した場合、g_tx_buffer[index] の index が確保しているメモリを超えてアクセスする事（バッファオーバラン）の対策です。

## 2.11.割り込み処理のオーバヘッドに関して[参考]

本プログラムをどこまで高速に処理できるかは AGT の割り込み処理の実行時間で決まります。

FSP を使ってコールバック関数を定義すると、少なからず FSP で生成した関数のオーバヘッドが加算されます。

実際に、最短でどの程度の速度での信号切り替えが可能かを調べたのが下記となります。

・コンパイラオプション　最適化なし(-O0)の場合

| | |
|---|---|
| AGT0（FSP で生成） | 8.8[us] |
| AGT1（ユーザで書き下した場合） | 3.0[us] |

・コンパイラオプション　デフォルト(-O2)の場合

| | |
|---|---|
| AGT0（FSP で生成） | 4.7[us] |
| AGT1（ユーザで書き下した場合） | 2.6[us] |

LED 側は、1 値(LED_TN, LED_TH, LED_TL)あたり、1us(min)の信号を受けられますが、マイコン側は割り込み＋信号切り替え処理の時間必要なため、そこまで速い時間での切り替えができていません。実際の処理時間は上記より長い時間とする必要があります。

AGT0 を使った処理では、1 値 10us（データ 1bit あたり、2 値なので、20us）。

AGT1 を使った処理では、1 値 5us（データ 1bit あたり、2 値なので、10us）。

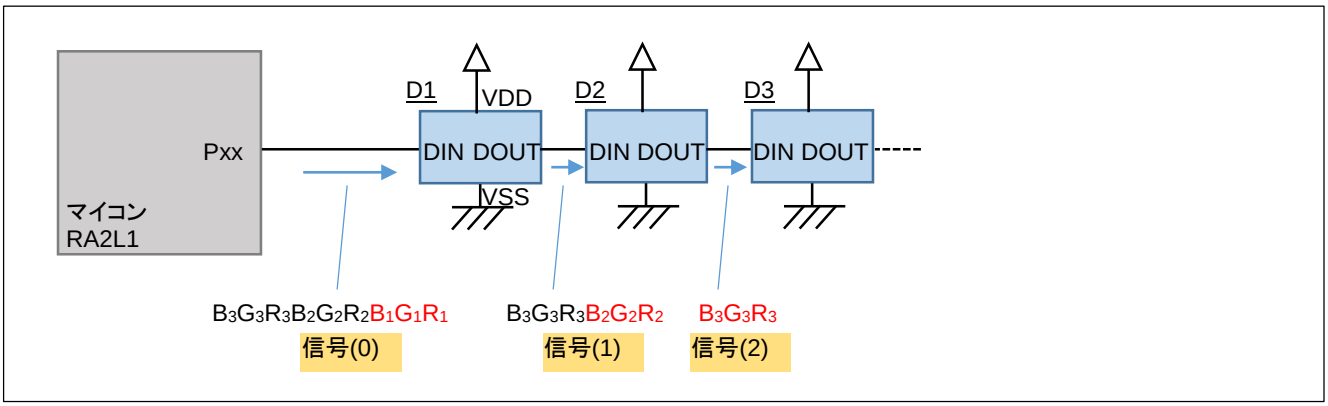
に設定しています。例えば、LED を 30 個駆動した場合、30×3×8×2=1440 値、AGT0 を使った場合、14.4ms。AGT1 を使った場合、7.2ms の処理時間となります。（+ラッチ時間の 3ms が必要）

LED のスペック上は、1440 値で、1.44ms(min)です。

現状 FSP が生成した処理でのオーバヘッドが大きい様に見受けられます。高速化オプションの様な選択肢が追加されることを期待します。

## 2.12.LED のデータハンドリングの挙動[参考]

### 2.12.1.TH/TL のタイミングで制御するタイプの LED の挙動（1 章で扱ったタイプ）

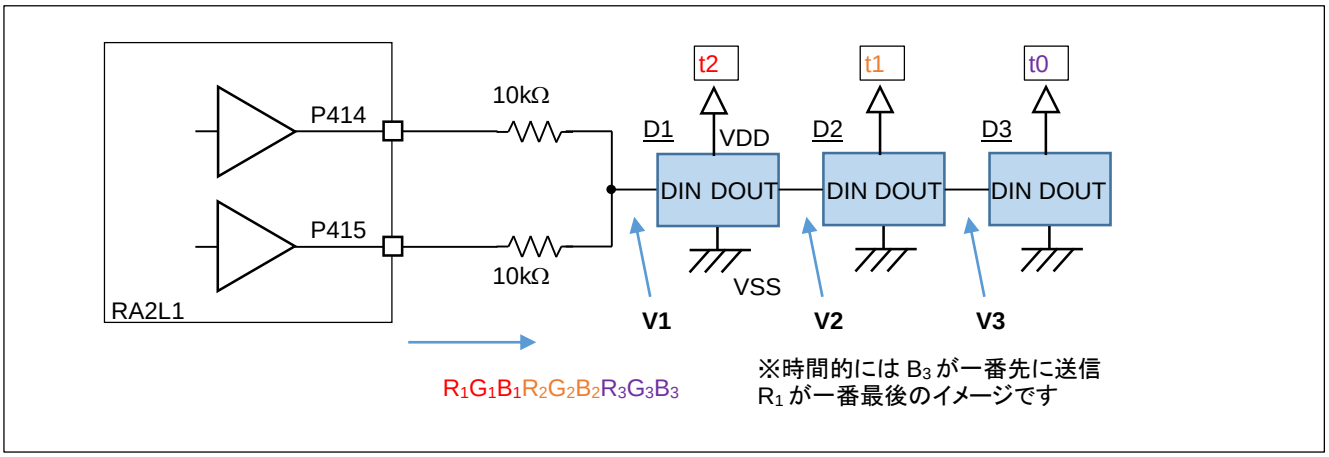



1 章の LED は、LED の視点に立つと（機器とか回路の側に立って考えるというのは、エンジニアの悪い癖なのかも知れませんが、気になってしまいます）、

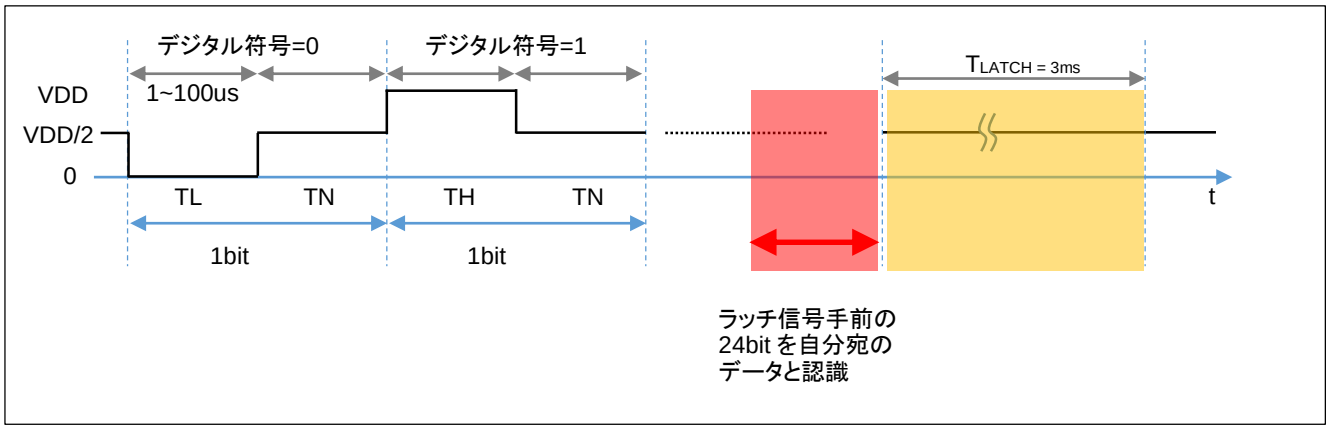
・流れてきたデータの先頭 24bit を自分宛のデータと認識し、データを取り込む
（先頭 24bit(3byte)のデータは次へ伝播させない）
・25bit 目以降のデータを次へそのままスルーさせる
※波形を見ると再生中継（D1 が自分自身のクロックで叩き直している…D1 が入力波形の 0/1 を認識した上で、D1 の内部クロックに同期させて、データを送り出している）の様です

挙動としては判りやすいです。

### 2.12.2.TH/TL/TN の 3 符合で制御するタイプの LED の挙動（2 章で扱ったタイプ）



D1, D2, D3 がどのように信号をハンドリングしているのか調べてみました。




ポイントは、信号送信後のラッチ（TN 期間 3ms 以上）のタイミングの直前の 24bit のデータを自分宛のデータと認識するという点です。

・V1（D1 に対する入力、マイコンの出力）

| t0 | t1 | t2 | |
|---|---|---|---|

| B3 G3 R3 | B2 G2 R2 | B1 G1 R1 | $T_{LATCH}$ |
|---|---|---|---|

時間的に前 → 時間的に後 t

※ここでは、波形送出のタイミングをオシロスコープで観測した時のイメージ（横軸－時間）で示します
（前の図、R1G1B1…R3G3B3 とは左右が逆な事に注意）

・V2（D2 に対する入力）

| ? ? ? | B3 G3 R3 | B2 G2 R2 | $T_{LATCH}$ |
|---|---|---|---|

D2 に対する入力は全体的に 24bit 分遅延したデータとなるようです。

・V3（D3 に対する入力）

| ? ? ? | ? ? ? | B3 G3 R3 | $T_{LATCH}$ |
|---|---|---|---|

D3 に対する入力は、D2 の入力に対し、全体的に 24bit 分遅延したデータとなるようです。

この仕組みにより、LED 側から見ると単純に $T_{LATCH}$ 前のデータを自分宛のデータとして処理をすれば良いという事となります。

LED 基準で考えると、

・受信したデータはバッファリングしておき 24bit 遅延させて送出する
・$T_{LATCH}$（TN 期間が 3ms 以上続いた）が来たらその直前に受信しているデータを基に LED の点灯を制御する

ということで良さそうです。

では、上記???の部分のデータはどうなるのでしょうか。

・V1

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| B3 | G3 | R3 | B2 | G2 | R2 | B1 | G1 | R1 | $T_{LATCH}$ |

・V2

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| B1 (n-1) | G1 (n-1) | R1 (n-1) | B3 | G3 | R3 | B2 | G2 | R2 | $T_{LATCH}$ |

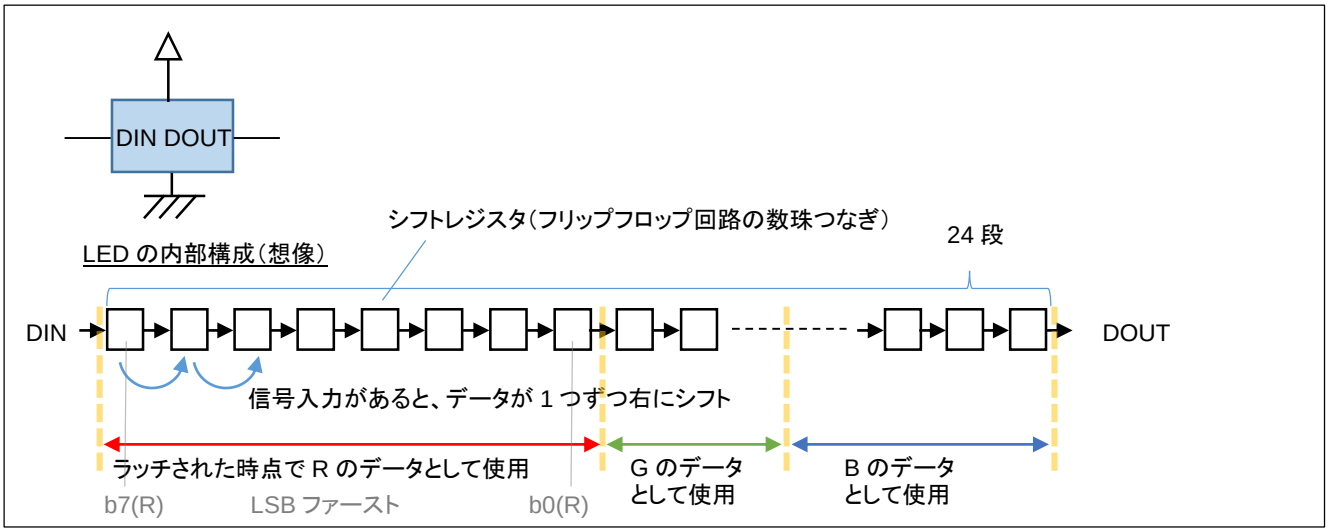
最初の 24bit データは、「信号受信前に D1 が自分自身の点灯パターンとしてラッチしていたデータ」でした。（一世代前の B1, G1, R1）

・V3

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| B2 (n-1) | G2 (n-1) | R2 (n-1) | B1 (n-1) | G1 (n-1) | R1 (n-1) | B3 | G3 | R3 | $T_{LATCH}$ |

この部分意味があるか？

同様に、D3 の最初の 24bit データは、「信号受信前に D2 が自分自身の点灯パターンとしてラッチしていたデータ」です。




LED の内部構造としては、24 段のシフトレジスタを持っており、外部から信号が来ると単純にデータをシフトさせる（DIN から 1 つ信号が入ってくると、DOUT から 1 つデータが出ていく、「ところてん」方式です）様になっているものと思われます。そのため、D2 が受信する最初の 24bit のデータは、D1 に格納されている 24bit のデータが出てくる様になっているものと思います。よって、D2, D3 にとって最後の 24bit 以外のデータは（回路構成上、一世代前のデータが入ってくるだけで、意味のないものであると思います。）

前述の構成により、

・24bit 分遅延したデータが次の LED に渡される

という動作が実現されます。（その反面、2 つ目以降の LED から見ると、最後の 24bit データ以外のデータは一見謎のデータとなります）

LED 側の立場で考えると、

・24 段のシフトレジスタ（一次記憶領域）を持たせておき、DIN と DOUT はシフトレジスタの入力と出力につなぐ
・ラッチ信号（3ms 以上の TN レベル）が入ってきたら、シフトレジスタのデータを LED に反映させる

ということで良さそうです。

実際に接続されている LED の数と、"n"コマンドで指定した LED の数（送信データ数）が一致しないときは、指定した LED と異なる LED が前回送ったデータで点灯したり、一見意味不明な挙動を示しますが、理由としては LED の内部構造に拠るものです（あくまで想像ですが）。このタイプの LED を使用する際は、LED の個数と n の指定を合わせる様にしてください。

## 2.13.まとめ

マイコン制御タイプのフルカラーLED は、バリエーションも豊富で、数珠つなぎになっているものも市販されています、専用のコントローラをわざわざ買わなくても、マイコンを使えば比較的簡単に制御ができますので、クリスマスの電飾等に活用してみてください。

## 取扱説明書改定記録

| バージョン | 発行日 | ページ | 改定内容 |
|---|---|---|---|
| REV.1.0.0.0 | 2021.6.29 | — | 初版発行 |

## お問合せ窓口

最新情報については弊社ホームページをご活用ください。

ご不明点は弊社サポート窓口までお問合せください。

株式会社 北斗電子

〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

TEL 011-640-8800　FAX 011-640-8801

e-mail：support@hokutodenshi.co.jp (サポート用)、order@hokutodenshi.co.jp (ご注文用)

URL:http://www.hokutodenshi.co.jp

ルネサス エレクトロニクス RA マイコン搭載
HSB シリーズマイコンボード 評価キット

# SmartRA 学習キット
# チュートリアル 9

株式会社北斗電子

 2021 年 6 月 29 日改訂 REV.1.0.0.0 (210629)