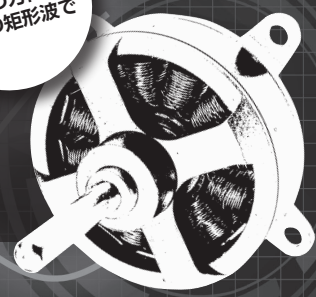
ダウンロード・データあります

初めての方向け…基本の矩形波で



ブラシレス・モータを回すプログラム 書き方講座

第7回　レジスタで直接マイコンの周辺機能を設定する その1…クロック

大黒 昭宣

連載では，モータとマイコン・ボードがセットになったキットP-NUCLEO-IHM001（STマイクロエレクトロニクス）を使ってDCブラシレス・モータを矩形波駆動で制御する方法を解説します．

精緻な制御を行うためにはモータ駆動電流を計測する必要があります．その際にはマイコンの持つA-D変換機能を使います．ところがプログラム開発フレームワークのMbedを使うと，A-D変換にかかる時間が長く，モータの回転数を上げにくいことが前回までに分かりました．

そこでマイコンの持つA-D変換機能をより効率良く利用するため，直接レジスタを設定する方法を数回に分けて紹介します．

Mbedは使いやすいけど細かい設定ができない

● 開発フレームワークMbedの特徴

開発用のフレームワークとして連載で使っているMbedでは，ウェブ・ブラウザ上で動作するオンライン・コンパイラと呼ばれる統合開発環境（IDE）が提供されています．この環境で使われているコンパイラは，Keil MDK-Armなどにも採用されているArmコンパイラ（armcc）です．GCCよりも効率の良いバイナリを生成できると言われています．

他の方法を見てみると，Keil MDK-ARMやIAR EMbedded Workbenchなどの商用のIDEやコンパイラは費用がかかってしまいますし，無償のGCCで開発環境を構築するのは非常に手間がかかります．

Mbedのオンライン・コンパイラはウェブ・ブラウザ上で動作するので環境構築の必要がありません．Mbedのアカウントを作成し，ログインするだけで自分のワークスペースをクラウド上に持つことができます．Mbedのアカウントは，登録や維持，利用に費用が一切発生しません．

その他に，無償で利用できるコンパイラとして，System Workbench for STM32があります．

これらMbed以外の4種のコンパイラは，バージョンが頻繁に更新されます．その都度動作が異なったり，時には動作しないといった問題が出るため，せっかく新バージョンで高速化されているのに以前のバージョンを使い続ける場合もあります．新バージョンを使っても問題がないことを確認するためにマイコンの動作設定命令を調査するのも面倒です．

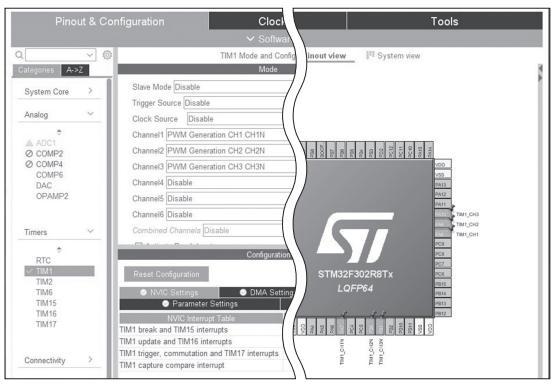


図1　STマイクロエレクトロニクスの開発環境はGUIでソースを吐き出しできる

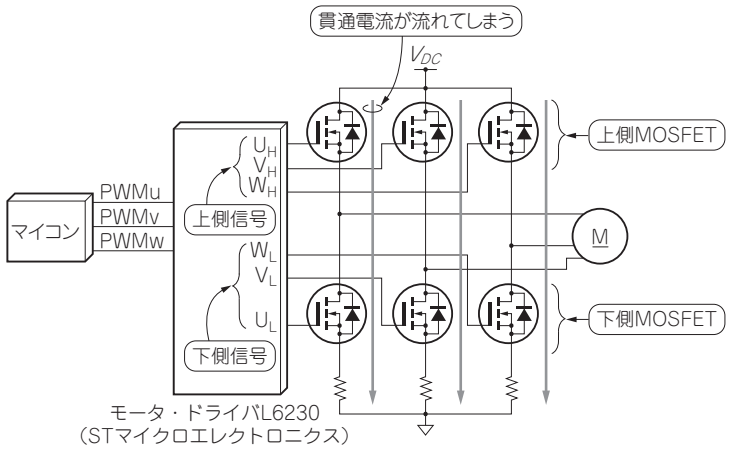
最近は**図1**のようにGUIでマイコンのハードウェアを設定し，各コンパイラに適応したコードを生成するツールが出ており便利です．しかし，何か問題が発生した場合やGUIで設定できないコードを追加したい場合に何を改変し，追加すればよいか分からず，行き詰まってしまう場合が多々あります．

Mbedでは主としてオフィシャル・ライブラリを利用してプログラミングします．従って周辺機能に対する動作の設定はかなり制限を受けます（細かい設定ができない）．

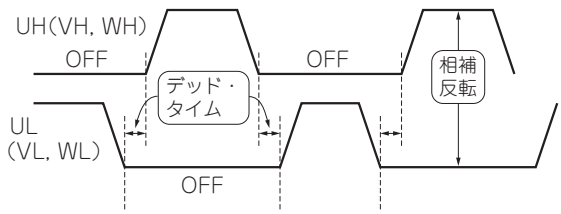
連載でも，これまではオフィシャル・ライブラリだけを使っていました．そのためモータの制御において十分な制御性能を出すことができませんでした．

● レジスタを直接叩いて設定する方法もある

今回はブラシレス・モータ制御のためにマイコンの周辺機能の動作について細かく設定します．これはMbedのオフィシャル・ライブラリを介さず，周辺機能のレジスタを直接設定することで行います．



（a）上下のMOSFETは同時にONできない



（b）デッド・タイム

図2　モータ・ドライバには上側と下側のMOSFETを同時にONにしないようにデッドタイムが設定されている

表1　Mbedオフィシャル・ライブラリを使う場合とマイコンのレジスタを直接設定する場合との比較

設定項目	Mbedオフィシャル・ライブラリ	直接レジスタ設定
周期	制限なし	制限なし
Duty	制限なし	制限なし
相補出力	できない	できる
デッドタイム設定	できない	できる
タイマ：オーバー，アンダーフロー割り込み	できない	できる

（a）PWM

設定項目	Mbedオフィシャル・ライブラリ	直接レジスタ設定
変換速度	固定（19.5クロック・サイクル）	可変（1.5～601.5クロック・サイクル）
高速スキャン	できない	できる
変換結果DMA転送	できない	できる

（b）A-Dコンバータ

DCブラシレス・モータの制御で特に重要なマイコンの周辺機能は，TIM1（PWM制御）とA-Dコンバータになります．**表1**にPWM出力機能とA-D変換機能についての比較を示します．それぞれオフィシャル・ライブラリを使った場合と，直接レジスタ設定した場合を比較しています．

Mbedオフィシャル・ライブラリを使ってDCブラシレス・モータを回す場合の注意点は，PWM制御で相補出力ができない点です．従って駆動方法は3PWM方式のインバータ駆動に限定されます．

連載で利用している「STマイコンで始めるブラシレス・モータ制御体験キット」のモータ・ドライバL6230（STマイクロエレクトロニクス）は3PWM方式ですので，マイコンから6PWMの相補出力ができなくてもモータを回せます．この方式ではマイコンからL6230へは上側MOSFETを駆動するPWM信号を入力します．下側MOSFETへの信号はL6230の内部で上側PWM信号を基に相補反転して生成するようになっています．つまりL6230内部で6PWMに変換し，かつデッド・タイムも自動で調整されます（**図2**）．

モータ駆動の最終段では電流を切り替えるためパワーMOSFETが使われています．このON/OFFを制御するゲート・ドライバや単一部品群で組んだゲート・ドライブ回路を使う場合，またはデッド・タイムの制御を詳細に行いたい場合などは相補出力の6PWM方式で駆動する必要があります．

Mbedオフィシャル・ライブラリを使った場合，A-Dコンバータの変換速度は19.5クロック・サイクルと遅いため，A-D変換を頻繁に行うブラシレス・モータ制御では大きなネックになります．またA-D変換の結果をDMA転送できないのもネックになります．

今回は，オフィシャル・ライブラリの設定方法と直接レジスタ設定方法とを比較しながら進めていきます．

● マイコンをよく知るにはレジスタを触った方がよい

直接レジスタ設定方式では，使用するマイコンの周辺デバイスとレジスタ構成の理解が重要になります．

第1回　ハードウェアと開発環境（2021年12月号）
第2回　ロータの回転位置に合わせて通電コイルを切り替える（2022年1月号）
第3回　ドローン用ブラシレス・モータを矩形波駆動で回す（2022年2月号）

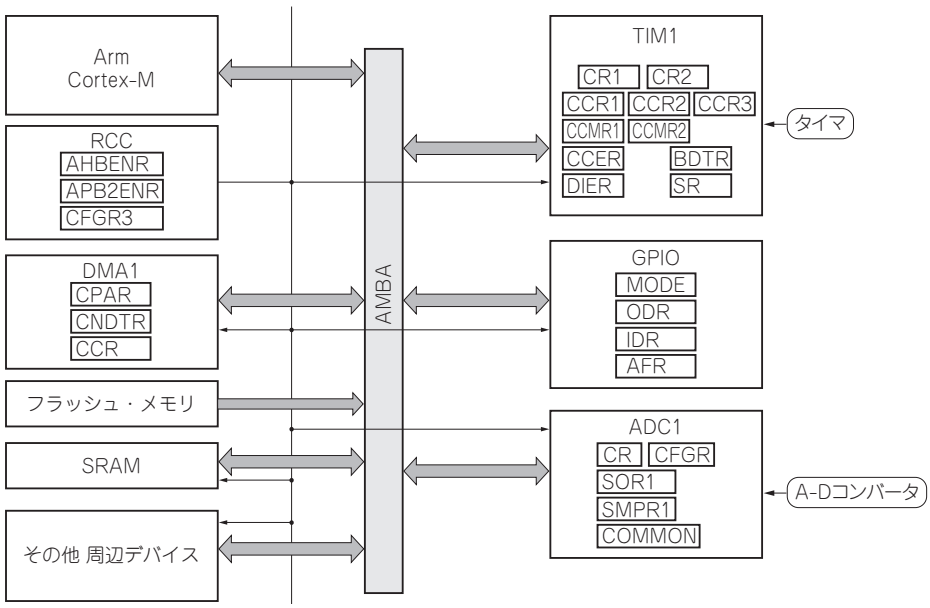


図3　STM32F302R8マイコンのレジスタのブロック図

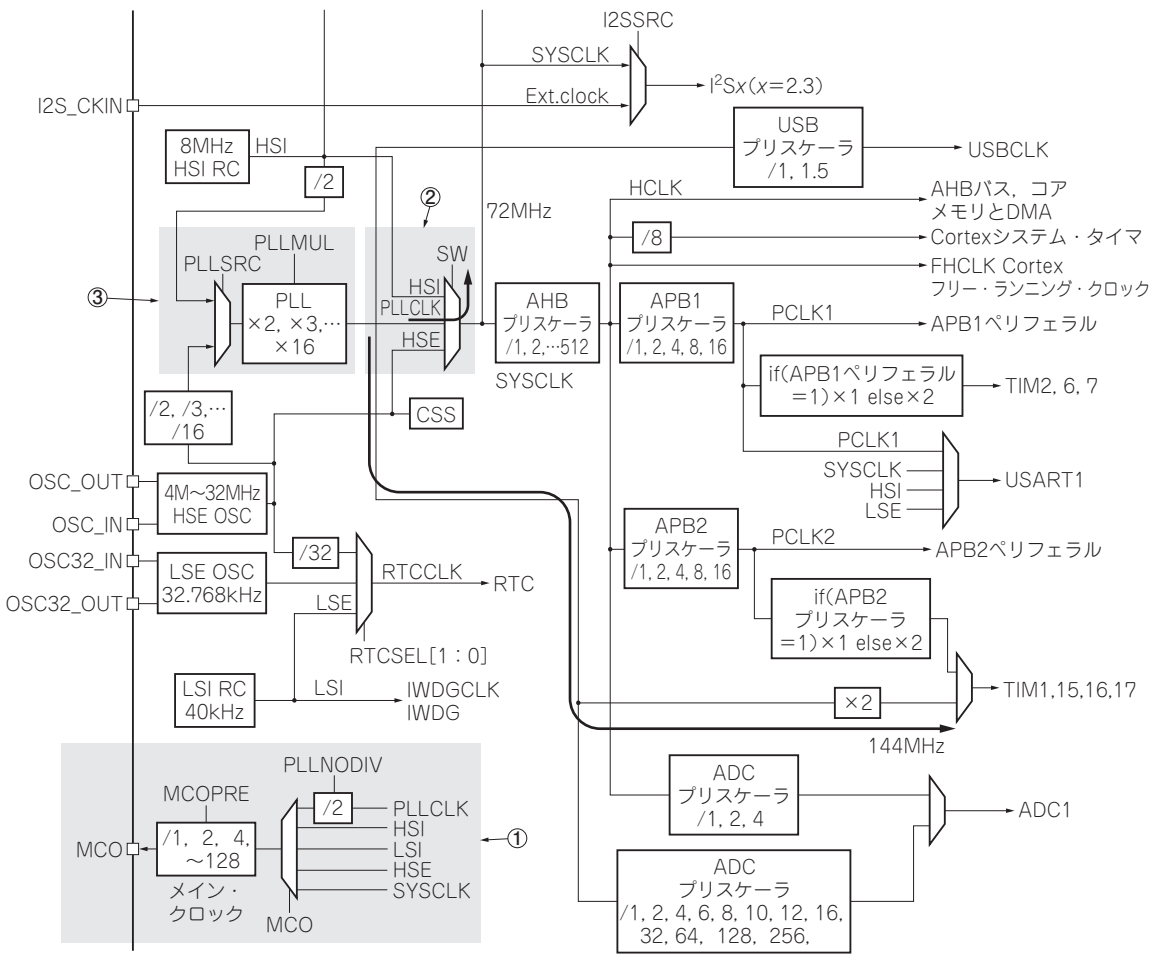


図4　STM32F302R8マイコンのクロック・ツリー

第4回　ボリュームの位置を読み取って正転/逆転を切り替える（2022年4月号）
第5回　負荷が変わっても回転速度を一定に保つ（2022年5月号）
第6回　電流PI制御を使ってモータ負荷に対する応答性を向上させる（2022年6月号）

リスト1　HSEを選択，分周はなし

```
//RCC->CFGR  |= RCC_CFGR_MCO_SYSCLK;
RCC->CFGR  |= RCC_CFGR_MCO_HSE ;
//RCC->CFGR  |= RCC_CFGR_MCO_HSI ;
//RCC->CFGR  |=RCC_CFGR_MCO_PLL;
//RCC->CFGR  |=RCC_CFGR_MCOPRE_2;
//RCC->CFGR  |=RCC_CFGR_MCOPRE_1;
//RCC->CFGR  |=RCC_CFGR_MCOPRE_0;
```

リスト2　SYSCLKを選択，分周は16

```
RCC->CFGR  |= RCC_CFGR_MCO_SYSCLK;
//RCC->CFGR  |= RCC_CFGR_MCO_HSE ;
//RCC->CFGR  |= RCC_CFGR_MCO_HSI ;
//RCC->CFGR  |=RCC_CFGR_MCO_PLL;
RCC->CFGR  |=RCC_CFGR_MCOPRE_2;
//RCC->CFGR  |=RCC_CFGR_MCOPRE_1;
//RCC->CFGR  |=RCC_CFGR_MCOPRE_0;
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLLNODIV	MCOPRE[2:1]		MCOF/MCOPRE0	予約	MCO[2:0]			I2SSRC	USBPRE	PLLMUL[3:0]				PLLXTPRE	PLLSRC
rw	rw	rw	r / rw	—	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLLSRC(1)	予約	PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	—	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

ビット 30:28 MCOPRE：マイクロコントローラ･クロック出力プリスケーラ
（STM32F303x6/8およびSTM32F328x8，STM32F303xDxEおよびSTM32F398xEのみ）
これらのビットはソフトウェアによってセット/クリアされる．MCO出力を有効にする前に，
このプリスケーラの変更を強く推奨する．
000：MCOを1分周　　001：MCOを2分周
010：MCOを4分周
…
111：MCOを128分周

ビット 26:24 MCO：マイクロコントローラ・クロック出力 ソフトウェアでセット/クリアされる．
000：MCO出力無効，MCOにクロックなし　　001：予約済み
010：LSIクロックの選択　　011：LSEクロックの選択
100：システム・クロック（SYSCLK）の選択　　101：HSIクロックの選択
110：HSEクロックの選択
111：PLLクロックの選択（PLLNODIVビットに応じて1または2で分周）

図5　クロック設定レジスタ（RCC_CFGR）

図3にSTM32F302R8のレジスタのブロック図を示します．今回開発するプログラムで利用する周辺デバイスはRCC（リセットとクロック制御），A-Dコンバータ，タイマ1，GPIO，DMAです．

一見，周辺デバイスのレジスタ設定は面倒で複雑ですが，初心者が組み込みソフトウェア開発を理解するための近道になります．レジスタ設定で自由に周辺デバイスの動作を決定できるので，レジスタのあるビットを設定するとどう動作するかを一度覚えてしまうとマイコンを自由自在にあやつれるようになります．

MbedはC++対応なので，レジスタ設定をクラス定義しカプセル化する方法があります．しかし，C++に精通していないと，ちまたに用意されているクラスのどこをどう触ればよいかチンプンカンプンになります．マイコンの動作理解には妨げになりますので，ここでは地道にレジスタを設定していきます．

クロック設定用レジスタをたたく

DCブラシレス・モータを円滑に制御するためのペリフェラル設定用のレジスタを**図3**に示します．モータ制御を目的とする主なレジスタは，以下です．

- 各ペリフェラルへクロック停止/供給するRCCレジスタ
- PWMでの相補出力，デッド・タイムを設定するためのTIMER1とGPIOレジスタ
- A-Dコンバータ高速化のためのADC1レジスタ
- A-D変換の結果を高速転送するためのDMA1レジスタ

● クロック制御用のRCCレジスタの設定

連載で使っているマイコン・ボードNucleo-F302R8（STマイクロエレクトロニクス）は，マイコンとしてSTM32F302R8を搭載しています．このマイコンのクロック・ツリーを**図4**に示します．

まず，購入時のSTM32F302R8クロック設定を確認

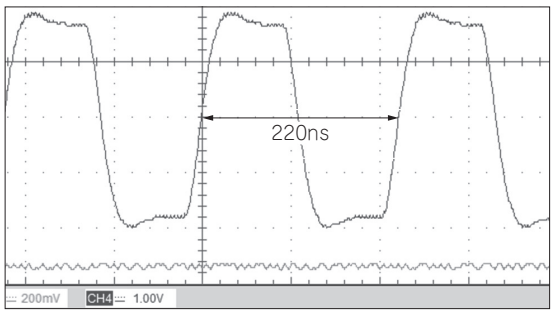


図6　システム・クロック（SYSCLK）を16分周して測定した

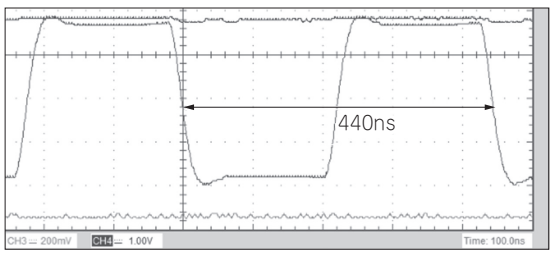


図7　PLLCLKの波形（32分周）

してみます．クロックの設定コードを**リスト1**，**リスト2**に示します．

図5にRCC_CFGRレジスタの構成を示します．クロック出力選択を利用する場合はビット30:28とビット26 ～ 24を設定します．レジスタのビットについて設定の詳細は後ほど説明します．

図4の①部分が基本クロックをモニタするブロックです．ビット26 ～ 24のMCOは基本クロック群のPLLCLK（発振逓倍），HIS（内部発振），HSE（外部発振），SYSCLKのいずれか1つをポートMCOに出力します．

▶SYSCLK

マイコンの動作周波数を決めるSYSCLKをオシロスコープで測定したものを**図6**に示します．ビット30 ～ 28のMCOPREで16分周に設定して測定しています．周波数を求める計算式は次の通りです．

$$SYSCLK=\frac{1}{\frac{220\text{ns}}{16\text{分周}}}=\frac{1}{13.75\times10^{-9}}\fallingdotseq72\times10^{6}=72\text{MHz} \quad \cdots\cdots\cdots(1)$$

STM32F302R8の最高速度に設定されています．72MHzを普通のプローブを使ってオシロスコープで測定すると波形がつぶれてしまうので分周してから測定しています．

図4の②のセレクタ部でSYSCLKはPLLCLK（発振逓倍），HIS（内部発振），HSE（外部発振）のいずれかが選択されていますので確認してみます．

▶PLLCLK

PLLCLKの波形を**図7**に示します．周波数の計算式は次の通りです

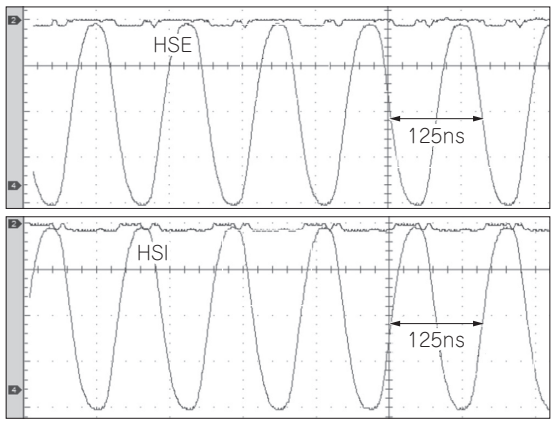


図8　HSEとHSIの波形（分周なし）

$$PLLCLK=\frac{1}{\frac{440\text{ns}}{32\text{分周}}}=\frac{1}{13.75\times10^{-9}}\fallingdotseq72\times10^{6}=72\text{MHz} \quad \cdots\cdots\cdots(2)$$

図4の①の部分で2分周してから16分周していますので32分周となります．測定値をもとに計算するとPLLCLKは72MHzになります．

▶HSEとHIS

図8は外部発振器HSEとHSIの波形です．内部発振器HSIは分周なしで8MHzになります．このことからSYSCLKはPLLCLKを利用していることが確認されました．

図4の③を見るとPLLCLKは8MHzのHSEまたはHSIの逓倍で作成されています．PLLCLKが72MHzということは分周＋逓倍で合わせて9倍に設定されています．内部発振HSIは精度が落ちますので利用せず，外部発振HSEを利用します．USB通信などでは高精度なクロックを発生させたいのでNucleoボードのデバッグ部にある，USB通信用マイコン（STN32F103）の水晶発振器を利用しています．

以上のように購入時には72MHzの最高速度に設定されていますので，基本クロック関係はこのまま使っていきます．

● ペリフェラルへの供給クロックの設定

各ペリフェラルへのクロック供給を見ていきます．タイマ1（TIM1）へのクロック供給は144MHzになります．**図9**，**図10**が各ペリフェラルへのクロック停止・供給するためのレジスタになります．レジスタ設定を**リスト3**に示します．例えば1行目の，

```
RCC->AHBENR |= (1 << 17);
```

は，RCCペリフェラル内のAHBENRレジスタの17ビット目に1を設定します．レジスタ内の他のビットに上書きを行わないためにOR（|=）操作にしています．これでポートA群にクロックが供給されます．

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約			ADC12 EN	予約			TSCEN	IOPG EN(1)	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	IOPH EN(1)
−			rw	−			rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
予約									CRC EN	FMC EN(1)	FLITF EN	予約	SRAM EN	DMA2 EN	DMA1 EN
−									rw			−	rw		

図9　AHBペリフェラル・クロック設定用レジスター覧（RCC_AHBENR）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
予約													TIM17 EN	TIM16 EN	TIM15 EN
−													rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI4 EN	USART 1EN	予約	SPI1 EN	TIM1 EN	予約										SYS CFGEN
rw		−	rw		−										rw

図10　APB2ペリフェラル・クロック設定用レジスタ（RCC_APB2ENR）

リスト4　ヘッダ・ファイル（stm32f302x8.h）**にはレジスタ設定用の定数が定義されている**

```
/********  Bit definition for RCC_AHBENR register  ********/
#define RCC_AHBENR_DMA1EN  ((uint32_t)0x00000001)  /*!< DMA1 clock enable */
#define RCC_AHBENR_SRAMEN  ((uint32_t)0x00000004)  /*!< SRAM interface clock enable */
#define RCC_AHBENR_FLITFEN ((uint32_t)0x00000010)  /*!< FLITF clock enable */
#define RCC_AHBENR_CRCEN   ((uint32_t)0x00000040)  /*!< CRC clock enable */
#define RCC_AHBENR_GPIOAEN ((uint32_t)0x00020000)  /*!< GPIOA clock enable */
#define RCC_AHBENR_GPIOBEN ((uint32_t)0x00040000)  /*!< GPIOB clock enable */
#define RCC_AHBENR_GPIOCEN ((uint32_t)0x00080000)  /*!< GPIOC clock enable */
#define RCC_AHBENR_GPIODEN ((uint32_t)0x00100000)  /*!< GPIOD clock enable */
#define RCC_AHBENR_GPIOFEN ((uint32_t)0x00400000)  /*!< GPIOF clock enable */
```

リスト3　ペリフェラルのクロックを設定するためにレジスタを叩くコード

```
RCC->AHBENR |= (1 << 17); // IOPA EN
RCC->AHBENR |= (1 << 18); // IOPB EN
RCC->AHBENR |= (1 << 19); // IOPC EN
RCC->APB2ENR |= (1 << 11);// TIM1 EN
RCC->AHBENR |= (1 << 28); // turn on ADC12 EN clock
RCC->AHBENR |= RCC_AHBENR_DMA1EN;
                        // supply clock to DMA1 EN
```

直接数字でビットを指定する代わりにヘッダ・ファイル（stm32f303x8.h）で定義されている定数を使用することもできます．6行目のDMA設定では，RCC_AHBENR_DMA1ENという定数を使ってレジスタを設定しています．

ヘッダ・ファイルstm32f303x8.h内の一部抜粋を**リスト4**に示します．RCCのAHBENRレジスタのビット位置を分かりやすい名前で定義しています．このようにレジスタへの書き込みを名前で定義することもできます．好みにより好きな方法でレジスタ設定してください．

レジスタについての詳細は，ペリフェラル・レジスタ設定用の資料として仕様書(1)と，ADC番号，オルターネート・ファンクション・ピン定義などが書かれたデータシート(2)および前述のヘッダ・ファイルstm32f302x8.hを参照してください．

＊　　＊　　＊

次回は，PWMやA-D変換，DMA関係のレジスタ設定について解説します．

◆参考文献◆

(1) STM32F302xx リファレンス マニュアル．
https://www.stmcu.jp/design/document/reference_manual/51493/
(2) STM32F302x6 STM32F302x8 データシート．
https://www.st.com/resource/en/datasheet/stm32f302r8.pdf

おおぐろ・あきよし