第 3 章　スクリプト言語 Pawn を用いたプログラミング（補足資料）

この資料では，紙面の都合で書籍に載せられなかった情報を提供しています．

1　周辺回路構成の変更方法

1−1　SetDefaultConfig 関数が行っていること

書籍中のほとんどのサンプルでは，SetDefaultConfig 関数によって設定した周辺回路構成を使用していました．SetDefaultConfig 関数を実行した際の周辺回路構成は，第 3 章の**表 1**〜**表 5** に示される状態になっていて，付録基板上のコネクタへの信号配置は**図 1** のようになっています．

CN1

5[V]		34	33	PB7	DI4
#BootSel	PH0	32	31	GND	
	PB6	30	29	PB4	#SD_CS
	PE2	28	27	PE3	
	PD4	26	25	PD5	
	PD6	24	23	PD7	
	PE7	22	21	PE6	AD1
AD2	PE5	20	19	PE4	AD3
	PD0	18	17	PD1	Enc1A
Enc2A	PD2	16	15	PD3	Enc2B
DO4	PG2	14	13	PG1	PWM5
DI3	PC7	12	11	PC6	DI2
DI1	PC5	10	9	PC4	DI0
U0TX	PA1	8	7	PA2	SSI0Clk
SSI0RX	PA4	6	5	PA5	SSI0TX
	PA6	4	3	PA7	
PWM7	PG7	2	1	GND	

CN2

GND		34	33	N.C.	
VBUS		32	31	PA3	
	PB5	30	29	PH2	USBSel
TDI	PC2	28	27	PC3	TDO
	PH1	26	25	PA0	U0RX
	PE1	24	23	GND	
	PB2	22	21	PE0	
GND		20	19	PG0	PWM4
	PB3	18	17	PG3	
PWM1	PF1	16	15	3.3[V]	
PWM3	PF3	14	13	PF2	PWM2
BT		12	11	PF4	
GND		10	9	GND	
#WAKE		8	7	PF0	PWM0
DO3	PF6	6	5	PF5	DO2
DO1	PG4	4	3	PF7	Enc1B
PWM6	PG6	2	1	PG5	DO0

図 1　SetDefaultConfig 関数によって設定した状態での機能割り当て

図 1 において，水色に塗られている信号が SetDefaultConfig 関数によって設定した部分になっています．ただし，PG2 は基板上の LED と接続されていて，PB7 は基板上の WAKE スイッチと接続されています．また，A-D 変換シーケンスの第 1 ステップは LM3S3748 の内蔵温度センサ値を取得するように設定されています．

なお，microSD カードへアクセスするための信号に関しては，SetDefaultConfig 関数を呼び出す前に設定を済ませてあります．

SetDefaultConfig 関数が行っていることを，Pawn のスクリプトで記述すると**リスト 1** となります．

リスト 1　SetDefaultConfig 関数と同じ働きをするスクリプト

```
1  /* GPIOOutput の利用情報 */
2  PinTypeGPIOOutput( GPIO_PORTG , GPIO_PIN_5 );                    PG5 を GPIOOutput として予約(ドライブ強度は 2mA)
3  PinTypeGPIOOutput( GPIO_PORTG , GPIO_PIN_4 );                    PG4 を GPIOOutput として予約(ドライブ強度は 2mA)
4  PinTypeGPIOOutput( GPIO_PORTF , GPIO_PIN_5 );                    PF5 を GPIOOutput として予約(ドライブ強度は 2mA)
5  PinTypeGPIOOutput( GPIO_PORTF , GPIO_PIN_6 );                    PF6 を GPIOOutput として予約(ドライブ強度は 2mA)
6  PinTypeGPIOOutput( GPIO_PORTG , GPIO_PIN_2 , STRENGTH_4MA );     PG2 を GPIOOutput として予約(ドライブ強度は 4mA)
7
8  /* GPIOInput の利用情報 */
9  PinTypeGPIOInput( GPIO_PORTC , GPIO_PIN_4 , TYPE_PULLUP );       PC4 を GPIOInput として予約(プルアップする)
10 PinTypeGPIOInput( GPIO_PORTC , GPIO_PIN_5 , TYPE_PULLUP );       PC5 を GPIOInput として予約(プルアップする)
```

```
11  PinTypeGPIOInput( GPIO_PORTC , GPIO_PIN_6 , TYPE_PULLUP );  PC6 を GPIOInput として予約(プルアップする)
12  PinTypeGPIOInput( GPIO_PORTC , GPIO_PIN_7 , TYPE_PULLUP );  PC7 を GPIOInput として予約(プルアップする)
13  PinTypeGPIOInput( GPIO_PORTB , GPIO_PIN_7 , TYPE_PULLUP );  PB7 を GPIOInput として予約(プルアップする)
14
15  /* A-D 変換シーケンスの利用情報 */
16  ADCStepTypeTempSensor();                                    温度センサを A-D 変換シーケンスへ含める(予約)
17  PinTypeADC(GPIO_PORTE , GPIO_PIN_6 );                       PE6 を A-D 変換シーケンスへ含める(予約)
18  PinTypeADC(GPIO_PORTE , GPIO_PIN_5 );                       PE5 を A-D 変換シーケンスへ含める(予約)
19  PinTypeADC(GPIO_PORTE , GPIO_PIN_4 );                       PE4 を A-D 変換シーケンスへ含める(予約)
20
21  /* PWM の利用情報 */
22  PinTypePWM( GPIO_PORTF , GPIO_PIN_0 );                      PF0 を PWM 出力として予約
23  PinTypePWM( GPIO_PORTF , GPIO_PIN_1 );                      PF1 を PWM 出力として予約
24  PinTypePWM( GPIO_PORTF , GPIO_PIN_2 );                      PF2 を PWM 出力として予約
25  PinTypePWM( GPIO_PORTF , GPIO_PIN_3 );                      PF3 を PWM 出力として予約
26  PinTypePWM( GPIO_PORTG , GPIO_PIN_0 );                      PG0 を PWM 出力として予約
27  PinTypePWM( GPIO_PORTG , GPIO_PIN_1 );                      PG1 を PWM 出力として予約
28  PinTypePWM( GPIO_PORTG , GPIO_PIN_6 );                      PG6 を PWM 出力として予約
29  PinTypePWM( GPIO_PORTG , GPIO_PIN_7 );                      PG7 を PWM 出力として予約
30
31  /* エンコーダ入力の利用情報*/
32  PinTypeQEI( GPIO_PORTD , GPIO_PIN_1 );                      PD1 を QEI 入力として予約(QEI 回路で計測)
33  PinTypeQEI( GPIO_PORTD , GPIO_PIN_7 );                      PD7 を QEI 入力として予約(QEI 回路で計測)
34  PinTypeQEI( GPIO_PORTD , GPIO_PIN_2 );                      PD2 を QEI 入力として予約(割り込みで計測)
35  PinTypeQEI( GPIO_PORTD , GPIO_PIN_3 );                      PD3 を QEI 入力として予約(割り込みで計測)
36
37  PeriphStart();                                              予約情報をもとに周辺回路を実際に設定する
```

リスト 1 の 1 行目から 35 行目で，各機能ごとに利用するピンの予約を行っています．この段階では，まだ実際の設定は行っておらず 37 行目の PeriphStart 関数を実行したときに，各周辺回路（PWM 以外）が設定されます．

また，予約を行う段階で同一のピンに重複した機能割り当てを行っていないかどうかのチェックを行っています．一部のピン（PA0 や PA1 等）は Pawn の仮想マシン自体が使用していますので，Pawn のスクリプトからは機能を割り振れないようになっています．**リスト 1** では省略していますが，PinType**関数は指定されたピンに既に別の機能が予約されていた場合には「-1」を返しますので，戻り値をきちんとチェックすることで，「ダブルブッキング」を検出することができます．

GPIOOpuput と GPIOInput に関しては，予約を行った順にチャネル番号（ビット番号）が割り振られていきます．従ってデフォルトの設定の場合，Pawn 内部ではディジタル出力，ディジタル入力共に 5 ビットのデータとして取り扱うこともできるようになっています．

setDIAll 関数を使って，5 本のディジタル入力ピンの状態を | PB7 | PC7 | PC6 | PC5 | PC4 | のような 5 ビットのデータとして取得することができます．逆に setDOAll 関数の引き数として | PG2 | PF6 | PF5 | PG4 | PG5 | のような 5 ビットのデータを渡すことで，5 本あるディジタル出力ピンの状態を一括して設定することができます．

A-D 変換のシーケンスに関しては，予約を行った順に A-D 変換を行うシーケンスが構成されます．デフォルトの設定では，A-D 変換シーケンスは 4 ステップで構成され，温度センサ→PE6→PE5→PE4 の順に A-D 変換が実行されます．

エンコーダ入力関しては，予約を行う順番とは関係なく役割が割り振られます．例えば，PD1 であれば QEI 回路の A 相入力，PD3 であれば割り込みを使った計数モジュールの B 相入力として取り扱われます．ただし，インクリメンタル・エンコーダのカウントを行うためには，2 相の入力が必要になりますので，「PD1」のみや「PD3」のみ

をエンコーダ入力として予約した場合は，カウント動作は行われません．

同様に PWM に関しても，PWM ジェネレータと PWM 出力ピンの接続がハード的に決まっていますので，予約を行う順番とは関係なしに役割が決定されます．つまり，PF0 であれば PWM ジェネレータ 0 番の第 1 出力ピン，PG7 であれば PWM ジェネレータ 3 番の第 2 出力ピンとして取り扱われます．ただし，デッド・バンド・モードを使用する場合は，各 PWM ジェネレータに接続されている PWM 信号を 2 本とも使用する設定にしてください．

PWM は使用状況に応じて PWM 周期やパルス幅の設定が変更になるので，PeriphStart 関数においても実際の設定は行わず「PWM に使用するという予約」が行われただけの状態になっています．Pawn のスクリプトにおいて PWMSetPeriod 関数を用いて PWM 周期を設定し，PWMSetPulseWidth 関数を用いてパルス幅の設定した際に PWM 出力ピンとしての設定が行われます．

周辺回路の設定を行う際に使用する関数は**表 1** となります．

表 1　周辺回路設定関数

関数名	引き数	動作
PinTypeGPIOOutput(port_no , pin_no , strength)	port_no ：ポート名称, pin_no ：ピン番号 strength：出力強度（下記のいずれかの値，省略可） STRENGTH_2MA ：出力強度＝2mA（デフォルト） STRENGTH_4MA ：出力強度＝4mA STRENGTH_8MA ：出力強度＝8mA STRENGTH_8MA_SC ：出力強度＝8mA slew 制御付き	指定したピンをディジタル出力として使用する予約を行う
PinTypeGPIOInput(port_no , pin_no , type)	port_no ：ポート名称, pin_no ：ピン番号 type ：ピンのタイプ(下記のいずれかの値) TYPE_STD ： 標準 TYPE_PULLUP ： プルアップ TYPE_PULLDOWN ： プルダウン	指定したピンをディジタル入力として使用する予約を行う
PinTypeADC(port_no , pin_no)	port_no ：ポート名称, pin_no ：ピン番号	指定したピンを A-D 変換シーケンスへ含める（予約）
ADCStepTypeTempSensor()	なし	温度センサを A-D 変換シーケンスへ含める（予約）
PinTypePWM(port_no , pin_no)	port_no ：ポート名称, pin_no ：ピン番号	指定したピンを PWM 出力として使用する予約を行う
PinTypeQEI(port_no , pin_no)	port_no ：ポート名称, pin_no ：ピン番号	指定したピンをエンコーダ入力として使用する予約を行う
PeriphStart()	なし	予約情報をもとに周辺回路の設定を行う

Pawn のスクリプト内で SetDefaultConfig 関数を実行したときには，C 言語側では**リスト 2** の処理が実行されています．

リスト 2　SetDefaultConfig 関数から呼び出される C 言語側の処理

各ピンの利用目的の予約	
DORegPinInfo("G5" , GPIO_STRENGTH_2MA);	PG5 を GPIOOutput として予約（ドライブ強度は 2mA）
DORegPinInfo("G4" , GPIO_STRENGTH_2MA);	PG4 を GPIOOutput として予約（ドライブ強度は 2mA）
DORegPinInfo("F5" , GPIO_STRENGTH_2MA);	PF5 を GPIOOutput として予約（ドライブ強度は 2mA）

DORegPinInfo("F6" , GPIO_STRENGTH_2MA);	PF6 を GPIOOutput として予約（ドライブ強度は 2mA）
DORegPinInfo("G2" , GPIO_STRENGTH_4MA);	PG2 を GPIOOutput として予約（ドライブ強度は 4mA）
DIRegPinInfo("C4" , GPIO_PIN_TYPE_STD_WPU);	PC4 を GPIOInput として予約（プルアップする）
DIRegPinInfo("C5" , GPIO_PIN_TYPE_STD_WPU);	PC5 を GPIOInput として予約（プルアップする）
DIRegPinInfo("C6" , GPIO_PIN_TYPE_STD_WPU);	PC6 を GPIOInput として予約（プルアップする）
DIRegPinInfo("C7" , GPIO_PIN_TYPE_STD_WPU);	PC7 を GPIOInput として予約（プルアップする）
DIRegPinInfo("B7" , GPIO_PIN_TYPE_STD_WPU);	PB7 を GPIOInput として予約（プルアップする）
ADCRegPinInfo("TS");	温度センサを A-D 変換シーケンスへ含める（予約）
ADCRegPinInfo("E6");	PE6 を A-D 変換シーケンスへ含める（予約）
ADCRegPinInfo("E5");	PE5 を A-D 変換シーケンスへ含める（予約）
ADCRegPinInfo("E4");	PE4 を A-D 変換シーケンスへ含める（予約）
PWMRegPinInfo("F0");	PF0 を PWM 出力として予約
PWMRegPinInfo("F1");	PF1 を PWM 出力として予約
PWMRegPinInfo("F2");	PF2 を PWM 出力として予約
PWMRegPinInfo("F3");	PF3 を PWM 出力として予約
PWMRegPinInfo("G0");	PG0 を PWM 出力として予約
PWMRegPinInfo("G1");	PG1 を PWM 出力として予約
PWMRegPinInfo("G6");	PG6 を PWM 出力として予約
PWMRegPinInfo("G7");	PG7 を PWM 出力として予約
QEIRegPinInfo("D1");	PD1 を QEI 入力として予約（QEI 回路で計測）
QEIRegPinInfo("F7");	PD7 を QEI 入力として予約（QEI 回路で計測）
QEIRegPinInfo("D2");	PD2 を QEI 入力として予約（割り込みで計測）
QEIRegPinInfo("D3");	PD3 を QEI 入力として予約（割り込みで計測）
各モジュールの初期化（Pawn の PeriphStart 関数に対応）	
DOStart();	ディジタル出力モジュールの初期化
DIStart();	ディジタル入力モジュールの初期化
ADCStart();	A-D 変換モジュールの初期化
QEIStart(0);	QEI モジュールの初期化
QEIStart(1);	割り込みを用いた計数モジュールの初期化

Pawn のスクリプトの実行が終了すると，周辺回路設定の初期化（**リスト 3** の処理）を行った後に，microSD カード・アクセス用のインターフェースのみを持つ状態で仮想マシンが再起動するようになっています．そのため，ハードウェアにアクセスするスクリプトでは，SetDefaultConfig 関数を入れておく必要があります．

リスト 3　スクリプト終了時に実行される C 言語側の処理（ClearPeriphConfig）

ADCStop();	A-D 変換モジュールの停止
PWMStop(0);	PWM ジェネレータ 0 番の停止（スクリプト中で起動していなければ何もしない）
PWMStop(1);	PWM ジェネレータ 1 番の停止（スクリプト中で起動していなければ何もしない）
PWMStop(2);	PWM ジェネレータ 2 番の停止（スクリプト中で起動していなければ何もしない）
PWMStop(3);	PWM ジェネレータ 3 番の停止（スクリプト中で起動していなければ何もしない）
QEIStop(0);	QEI モジュールの停止
QEIStop(1);	割り込みを用いた計数モジュールの停止
ClearPinFunction();	ピンの利用状況リストの初期化
DOInfoClear();	ディジタル出力ピン予約情報の初期化
DIInfoClear();	ディジタル入力ピン予約情報の初期化
ADCInfoClear();	A-D 変換シーケンス予約情報の初期化
PWMInfoClear();	PWM 出力ピン予約情報の初期化
QEIInfoClear();	エンコーダ入力ピン予約情報の初期化

現在の実装では，SetDefaultConfig 関数（または PeriphStart 関数）を実行した後は，スクリプト中で周辺回路構成を変更する機能がありません．ただ，各モジュールごとに「設定のクリア」，「初期化の実行」を行う C 言語の関数は用意されていますので，Pawn のスクリプトから呼び出せるようにネイティブ関数リストへ追加登録す

ることで，スクリプト実行中に周辺回路構成を修正する機能を実装することは可能です.

1−2　周辺回路構成の変更例

本書で使用している Pawn の仮想マシンは，スクリプト側で毎回周辺回路構成を行う仕様になっていますので，「アナログ入力を増やしたい」,「ディジタル出力を増やしたい」といった場合は，周辺回路を設定するスクリプトを作成することで，使用目的に合った回路構成を実現可能です.

周辺回路構成を設定する際の制約事項としては下記のものがあります.

①ディジタル出力数，ディジタル出力数の最大値は 16 本　（ソフトウェア上の制約）

②アナログ入力，PWM 出力の最大数は 8 本　（LM3S3748 固有の制約）

③QEI モジュールを用いたエンコーダ計測は 1 チャネル　（LM3S3748 固有の制約）

④割り込みを用いたエンコーダ計測は 1 チャネル　（ソフトウェア上の制約）

CN1

5[V]		34	33	PB7	DI0
#BootSel	PH0	32	31	GND	
	PB6	30	29	PB4	#SD_CS
	PE2	28	27	PE3	
AD7	PD4	26	25	PD5	AD6
AD5	PD6	24	23	PD7	AD4
AD0	PE7	22	21	PE6	AD1
AD2	PE5	20	19	PE4	AD3
	PD0	18	17	PD1	
	PD2	16	15	PD3	
DO0	PG2	14	13	PG1	
	PC7	12	11	PC6	
	PC5	10	9	PC4	
U0TX	PA1	8	7	PA2	SSI0Clk
SSI0RX	PA4	6	5	PA5	SSI0TX
	PA6	4	3	PA7	
	PG7	2	1	GND	

CN2

GND		34	33	N.C.	
VBUS		32	31	PA3	
	PB5	30	29	PH2	USBSel
TDI	PC2	28	27	PC3	TDO
	PH1	26	25	PA0	U0RX
	PE1	24	23	GND	
	PB2	22	21	PE0	
GND		20	19	PG0	
	PB3	18	17	PG3	
	PF1	16	15	3.3[V]	
	PF3	14	13	PF2	
BT		12	11	PF4	
GND		10	9	GND	
#WAKE		8	7	PF0	
	PF6	6	5	PF5	
	PG4	4	3	PF7	
	PG6	2	1	PG5	

図 2　周辺回路の変更例

図 2 の示すように，ディジタル入力 1 本，ディジタル出力 1 本，アナログ入力 8 チャネルの構成にする場合の Pawn スクリプトは**リスト 4** となります.

リスト 4　図 2 の回路構成を実現する Pawn スクリプト

```
1   /* GPIOOutput の利用情報 */
2   PinTypeGPIOOutput( GPIO_PORTG , GPIO_PIN_2 , STRENGTH_4MA );   PG2 を GPIOOutput として予約
3                                                                          ドライブ強度は 4mA
4   /* GPIOInput の利用情報 */
5   PinTypeGPIOInput( GPIO_PORTB , GPIO_PIN_7 , TYPE_PULLUP );     PB7 を GPIOInput として予約
6
7   /* A-D 変換シーケンスの利用情報 */
8   PinTypeADC(GPIO_PORTE , GPIO_PIN_7 );                          PE7 を A-D 変換シーケンスへ含める（予約）
9   PinTypeADC(GPIO_PORTE , GPIO_PIN_6 );                          PE6 を A-D 変換シーケンスへ含める（予約）
10  PinTypeADC(GPIO_PORTE , GPIO_PIN_5 );                          PE5 を A-D 変換シーケンスへ含める（予約）
11  PinTypeADC(GPIO_PORTE , GPIO_PIN_4 );                          PE4 を A-D 変換シーケンスへ含める（予約）
12  PinTypeADC(GPIO_PORTD , GPIO_PIN_7 );                          PD7 を A-D 変換シーケンスへ含める（予約）
13  PinTypeADC(GPIO_PORTD , GPIO_PIN_6 );                          PD6 を A-D 変換シーケンスへ含める（予約）
14  PinTypeADC(GPIO_PORTD , GPIO_PIN_5 );                          PD5 を A-D 変換シーケンスへ含める（予約）
15  PinTypeADC(GPIO_PORTD , GPIO_PIN_4 );                          PD4 を A-D 変換シーケンスへ含める（予約）
```

16 17	PeriphStart(); 	予約情報をもとに周辺回路を実際に設定する

Pawn のスクリプトでの周辺回路設定に限ったことではありませんが，実際のハード的な接続を充分確認の上，周辺回路設定を行ってください．

2　Pawn の仮想マシンに機能を追加する（C 言語から Pawn の関数を呼ぶ）

書籍中では，Pawn のスクリプトから C 言語で定義した関数を使う方法を紹介しました．Pawn は C 言語側から Pawn のスクリプト上で定義された関数を呼び出す仕組みも提供しています．この仕組みを使うと，割り込みを使ったスクリプトを記述することが可能になります．

ここでは，LM3S3748 の SysTick モジュールの割り込みを使った，簡易的なタイマ機能を Pawn の仮想マシンに追加してみたいと思います．

Pawn の仮想マシンを改造しますので，StellarisWare 等の開発ツールが必要になります．インストールしていない場合は，書籍 3 章「●開発環境の準備」の手順でインストールを行ってください．

修正作業の概要は，以下のようになります．

C 言語側

①SysTick モジュールの割り込みハンドラの作成

C 言語側の割り込みハンドラから Pawn のスクリプト上で定義された関数を呼び出します．

②SysTick モジュールの割り込みを有効にする関数の作成

③SysTick モジュールの割り込みを無効にする関数の作成

④上記二つの関数を PawnVM に登録する

⑤割り込み関連の準備/後始末

Pawn 側

・追加した機能の定義をインクルード・ファイル（Stellaris.inc）へ追加する

2－1　C 言語側の修正

まず，C 言語側の修正です．**リスト 5** の赤字の部分を「pawn_run.c」に追加します．

リスト 5　pawn_run.c の修正

168
169
170
171
172
172
 	
185
186
	void ClearPeriphConfig()
	{
	　MAP_SysTickIntDisable();　/* SysTick の割り込みを無効にする */
	
	　ADCStop();
	　PWMStop(0);
	
	　　中略
	
	　QEIInfoClear();
	}
		⑤-2

```
      中略

738  void SysTickHandler(void) /* SysTick 用割り込みハンドラ  */
739  {
740    int funcIdx;
741    cell ret;
742
743    if( amx_FindPublic(&g_amx , "@SysTickIntHandler", &funcIdx) == AMX_ERR_NONE )
744    {
745      amx_Exec(&g_amx , &ret, funcIdx);
746    }
747  }
748
749  static cell AMX_NATIVE_CALL n_SysTickIntEnable(AMX *amx,const cell *params)
750  {
751    /* SysTick の割り込みを無効にする */
752    MAP_SysTickIntDisable();
753    /* 割り込みハンドラの登録 */
754    SysTickIntRegister( SysTickHandler );
755    /* SysTick の割り込みを有効にする */
756    MAP_SysTickIntEnable();
757
758    return 0;
759  }
760
761  static cell AMX_NATIVE_CALL n_SysTickIntDisable(AMX *amx,const cell *params)
762  {
763    /* SysTick の割り込みを無効にする */
764    MAP_SysTickIntDisable();
765
766    return 0;
767  }
768
769  const AMX_NATIVE_INFO MY_Natives[] =
770  {
771    { "printf"             , n_printf },

       中略

805    { "ICP"            , n_ICP },
806    { "SysTickIntEnable"   , n_SysTickIntEnable },   /* SysTickIntEnable を PawnVM に登録する */
807    { "SysTickIntDisable"  , n_SysTickIntDisable },  /* SysTickIntDisable を PawnVM に登録する */
808    { NULL , NULL }         // terminator
809  };
810



1187   MAP_SysTickPeriodSet(MAP_SysCtlClockGet() / SYSTICKS_PER_SECOND);
1188   MAP_SysTickEnable();
1189   MAP_SysTickIntDisable();  /* SysTick 割り込みを無効にしておく */
```

①
②
③
④
⑤-1

リスト 5 で加えた修正の解説ですが，一番重要な部分が「割り込みハンドラ（**リスト 5** の①）」になります．このハンドラは，割り込みを有効にしたときに 10ms ごとに呼び出されます．割り込みが発生した際の具体的な処理内容は「Pawn のスクリプトで記述」する形式を採りたいため，ハンドラ内では「スクリプト側で定義された関数を呼び出す」処理だけを行っています．

```
void SysTickHandler(void) /* SysTick 用割り込みハンドラ  */
{
```

```
  int funcIdx;
  cell ret;

  if( amx_FindPublic(&g_amx , "@SysTickIntHandler", &funcIdx) == AMX_ERR_NONE )
  {                                                    /* 関数@SysTickIntHandler の検索 */
    amx_Exec(&g_amx , &ret, funcIdx);                  /* 関数が存在した場合は，関数を実行 */
  }
}
```

C 言語から Pawn で定義された関数を呼び出す手順は

○関数の名前（文字列）を指定して，関数インデックスを取得する（amx_FindPublic を使用する）

○引き数が必要な場合は，引き数のデータを仮想マシンに渡す（amx_Push を使用する）

○Pawn 側の関数を実行する（amx_Exec を使用する）

となります.

Pawn で定義された関数を呼び出す際に使用する関数は**表 2** となります.

表 2　Pawn のスクリプト内で定義された関数を呼び出すための API

関数名	引き数	動作
int amx_FindPublic (AMX *amx, char *funcname, int *index)	 AMX *amx　　: 仮想マシンのポインタ char *funcname : 関数名 int *index　　: 関数インデックス	指定された名称の関数インデックスを public 関数データーブルから取得する
int amx_Push(AMX *amx, cell value)	 AMX *amx　　: 仮想マシンのポインタ cell value　　: 仮想マシンへ渡すデータ	仮想マシンへデータを渡す
int amx_Exec(AMX *amx, long *retval, int index)	 AMX *amx　　: 仮想マシンのポインタ long *retval　: 関数の戻り値 int index　　: 関数インデックス	インデックスで指定された関数を実行する

今回の例では，Pawn のスクリプトで定義された「@SysTickIntHandler」と言う名前の関数を検索し，関数が見つかった場合に「@SysTickIntHandler」の実行を Pawn の仮想マシンへ依頼しています．後に出てきますが，今回使用している「@SysTickIntHandler」は引き数を持たない関数ですので，関数を探した直後に「amx_Exce」によって関数の実行依頼をしていますが，引き数を持つスクリプト側の関数を呼び出す場合は，「amx_Exce」を実行する前に，「amx_Push」を使用して仮想マシンへデータを渡しておく必要があります.

例えば，引き数を一つ持つ関数「@func1(param)」を検索して呼び出す処理は，下記のようになります.

```
/* 仮想マシン amx は別途実体化されているものとする */
int funcIdx;
cell param , ret;

/* param の値をセットする  */

if( amx_FindPublic(&amx , "@func1", &funcIdx) == AMX_ERR_NONE ) /* 関数@func1 を探す */
{
  amx_Push(&amx , param );   /* 引き数のデータを仮想マシンへ渡す */
  amx_Exec(&amx , &ret, funcIdx); /* 関数を呼び出す */
}
```

また，複数の引き数を持つ関数を呼び出す場合は，「引き数の順番」と「仮想マシンへ渡す順番」が逆になりますので注意が必要です．例えば，引き数を 2 個持つ関数「@func2(param1 , param2)」を呼び出す処理は，

```
/* 仮想マシン amx は別途実体化されているものとする */
int funcIdx;
```

```
cell param1 , param2 , ret;

/* param1 と param2 の値をセットする  */

if( amx_FindPublic(&amx , "@func2", &funcIdx) == AMX_ERR_NONE ) /* 関数@func2 を探す */
{
  amx_Push(&amx , param2 );   /* 第 2 引き数のデータを仮想マシンへ渡す */
  amx_Push(&amx , param1 );   /* 第 1 引き数のデータを仮想マシンへ渡す */
  amx_Exec(&amx , &ret, funcIdx); /* 関数を呼び出す */
}
```

となります.

割り込みを「有効化」/「無効化」する処理を Pawn スクリプトから行う際に使用する C 言語側の関数（**リスト 5** の②と③）は下記のようになります．これらは，スクリプト側から呼び出される関数となっています．「n_SysTickIntEnable」では，割り込みハンドラ（SysTickHandler）の登録を行い，SysTick モジュールの割り込みを有効にしています．逆に「n_SysTickIntDisable」では，SysTick モジュールの割り込みを無効にしています．どちらも，引き数を持たない関数ですので，書籍第 3 章の例と比べて簡素な作りになっています．

```
static cell AMX_NATIVE_CALL n_SysTickIntEnable(AMX *amx,const cell *params)
{
  /* SysTick の割り込みを無効にする */
  MAP_SysTickIntDisable();
  /* 割り込みハンドラの登録 */
  SysTickIntRegister( SysTickHandler );
  /* SysTick の割り込みを有効にする */
  MAP_SysTickIntEnable();

  return 0;
}

static cell AMX_NATIVE_CALL n_SysTickIntDisable(AMX *amx,const cell *params)
{
  /* SysTick の割り込みを無効にする */
  MAP_SysTickIntDisable();

  return 0;
}
```

作成した二つの拡張機能を Pawn の仮想マシンから呼び出せるように，**リスト 5** の④でネイティブ関数リスト（MY_Natives）へ登録しています．これにより追加した拡張機能は，それぞれ「SysTickIntEnable」，「SysTickIntDisable」という関数名で Pawn のスクリプト中で使用できるようになります.

```
const AMX_NATIVE_INFO MY_Natives[] =
{
  { "printf"            , n_printf },

     中略

  { "ICP"           , n_ICP },
  { "SysTickIntEnable"  , n_SysTickIntEnable },   /* SysTickIntEnable を PawnVM に登録する */
  { "SysTickIntDisable" , n_SysTickIntDisable },  /* SysTickIntDisable を PawnVM に登録する */
  { NULL , NULL }       // terminator
};
```

Pawn のスクリプトにおいて，「SysTickIntEnable」が実行された時から，SysTick モジュールの割り込みを使用しますので,システムが起動した時点では SysTick モジュールの割り込みを無効にしています(**リスト 5** の⑤-1).

また，スクリプト内で SysTick モジュールの割り込みを有効にした際の「割り込みの止め忘れ」を防止するた

めに，スクリプトの実行が終了した時に行われる処理（ClearPeriphConfig）の中でSysTickモジュールの割り込みを無効にしています（**リスト5**の⑤-2）.

リスト5の修正を加えた新しいPawnの仮想マシンは,書籍3章の手順を参考に付属基板へ書き込んでください.

2−2　Pwan側の準備とサンプル・スクリプト

仮想マシンへ追加した二つの拡張機能と割り込み処理に使用する関数をPawnのコンパイラが認識できるように,インクルード・ファイル（PawnDev¥host_app¥include¥Stellaris.inc」）に下記の修正を加えます(赤字の部分が修正個所).

リスト6　Stellaris.incの修正

```
119 stock ADCStepTypeTempSensor()
120 {
121   return PinTypeADC( TEMP_SENSOR , TEMP_SENSOR );
122 }
123
124 native SysTickIntEnable( );
125 native SysTickIntDisable( );
126
127 forward @SysTickIntHandler( );
```

新しい拡張機能「SysTickIntEnable」の定義
新しい拡張機能「SysTickIntDisable」の定義

関数「@SysTickIntHandler」の宣言

割り込み処理に使用する関数「@SysTickIntHandler」は，具体的な中身をユーザのスクリプト中で記述しますので，キーワード「forward」を付けてプロトタイプの宣言をしておきます.「forward」宣言された関数に関しては，具体的な中身はユーザ・スクリプト中で書くことができるため，割り込みを使ったシステムのテストも手軽に行うことが可能になります.

今回追加した割り込みを使用したサンプル・スクリプトの例は**リスト7**となります.

リスト7　SysTick割り込みを使ったサンプル・スクリプト

```
1  #include <Stellaris>
2
3  new count;
4
5  @SysTickIntHandler(){
6    count++;
7    if( (count % 50) == 0) {
8      printf("count=%d¥r¥n" , count);
9    }
10 }
11
12 main(){
13   SetDefaultConfig();
14
15   GPIOWrite(PG2 , 1);
16
17   count = 0;
18   SysTickIntEnable();
19   printf("irq test ¥r¥n");
20
21   while ( count < 100 ) {
22     GPIOWrite(PG2 , ~GPIORead(PG2) & 0x1  );
23     mwait(100);
24   }
25
26   GPIOWrite(PG2 , 1);
```

グローバル変数 count

割り込み処理関数の実体
  変数 count を 1 増やす

  割り込み 50 回ごとに count 値を表示

メイン関数
  周辺回路をデフォルト設定にする

  PG2(LED が接続)へ 1 を出力→LED が点灯

  count を 0 にする
  SysTick モジュールの割り込みを有効にする

  count 値が 100 より小さい間ループする（①）
    PG2 の出力をトグルする（0 なら 1，1 なら 0）.
    100ms 休む

  LED を点灯

```
27    count = 0;                                          count を 0 にする
28    SysTickIntDisable();                                SysTick モジュールの割り込みを無効にする
29    printf("IntDisable ¥r¥n");
30
31    while ( count < 10 ) {                              count 値が 10 より小さい間ループする (②)
32      GPIOWrite(PG2 , 0);                                 LED を消灯
33      mwait(175);                                         175ms 休む
34      GPIOWrite(PG2 , 1);                                 LED を点灯
35      mwait(25);                                          25ms 休む
36      count++;                                            count を 1 増やす
37    }
38
39    GPIOWrite(PG2 , 1);                                 LED を点灯
40    count = 0;                                          count を 0 にする
41    SysTickIntEnable();                                 SysTick モジュールの割り込みを有効にする
42    printf("IntEnable ¥r¥n");
43
44    while ( count < 200 ) {                             count 値が 200 より小さい間ループする (③)
45      GPIOWrite(PG2 , ~GPIORead(PG2) & 0x1  );            PG2 の出力をトグルする.
46      mwait(100);                                         100ms 休む
47    }
48
49    printf("bye ¥r¥n");
50  }
```

リスト 7 の 5 行目～10 行目が，割り込み関数（@SysTickIntHandler）の具体的な処理内容になっています．今回は，10ms ごとの割り込み発生時に，変数 count の値を 1 増やし，割り込み 50 回ごとに count 値を表示するという簡単なものになっています．

一方，メイン関数で登場する while ループのうち①と③のループについては，割り込みを有効にしていますので，ループ中で count 値を更新していませんが，①のループであれば 1 秒，③のループであれば 2 秒以上経過した時にループが終了します．逆に，割り込みを使用していない②のループでは，ループ内で count 値の更新を行っています．

今回の例題のように，割り込みを C 言語側で捉え，Pawn 側で実装した処理を呼び出す仕組みを使用すると，「スイッチが押された瞬間に行う処理」や「シリアル通信で何かデータを受信した際に行う処理」を Pawn のスクリプトを使って手軽に記述することもできます．