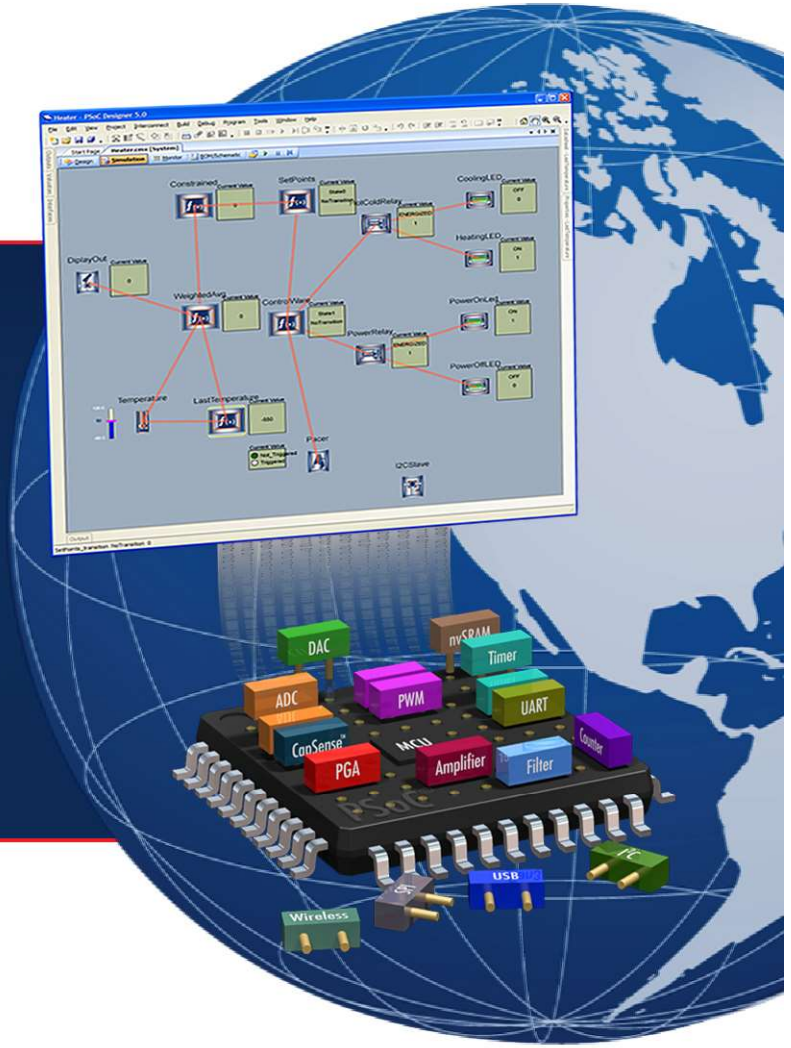
タイマー割込みとPWM周波数設定の演習

timer_pwm2

PSoC Experiment Lab



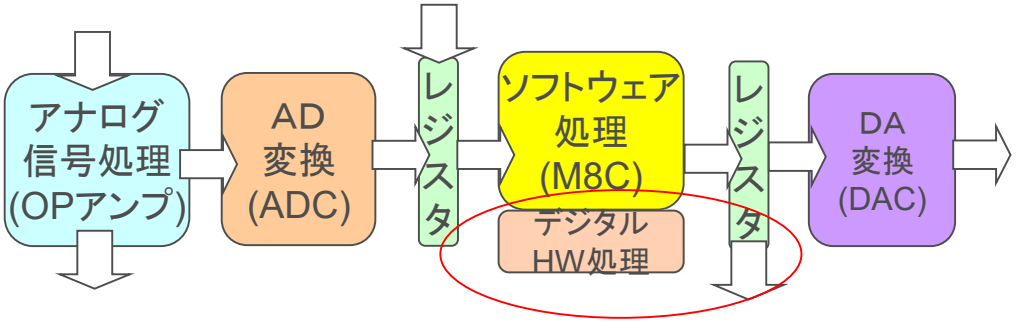
Experiment Course Material V1.20
October 6th., 2020
timer_pwm2.pptx (19 Slides)
Renji Mikami

タイマー割込みを使用してPWMで音楽を演奏

ラボ (ドレミ)

timer_pwm2

タイマーからの割り込みとPWM周波数をレジスタ値の
書き換えでのコントロール法がポイント



音楽を演奏するために必要な要素

1.音符は、音程と時間の要素で成り立つ

2.音程を作るために周波数を生成する機能が必要

PWMユーザーモジュールで実現(実際的には16ビット程度必要-PWM16使用)

3.音程を変えるにはPWMのPeriod レジスタとPulse Width レジスタの

値をプログラムで変えていけばよい

4.音の長さ(時間の要素)はどう実現するか

タイマー・モジュールを動作させておき一定時間ごとにMPUに割り込みをかける.この割り込みを演奏の最小制御単位時間として、”音符”を実現する

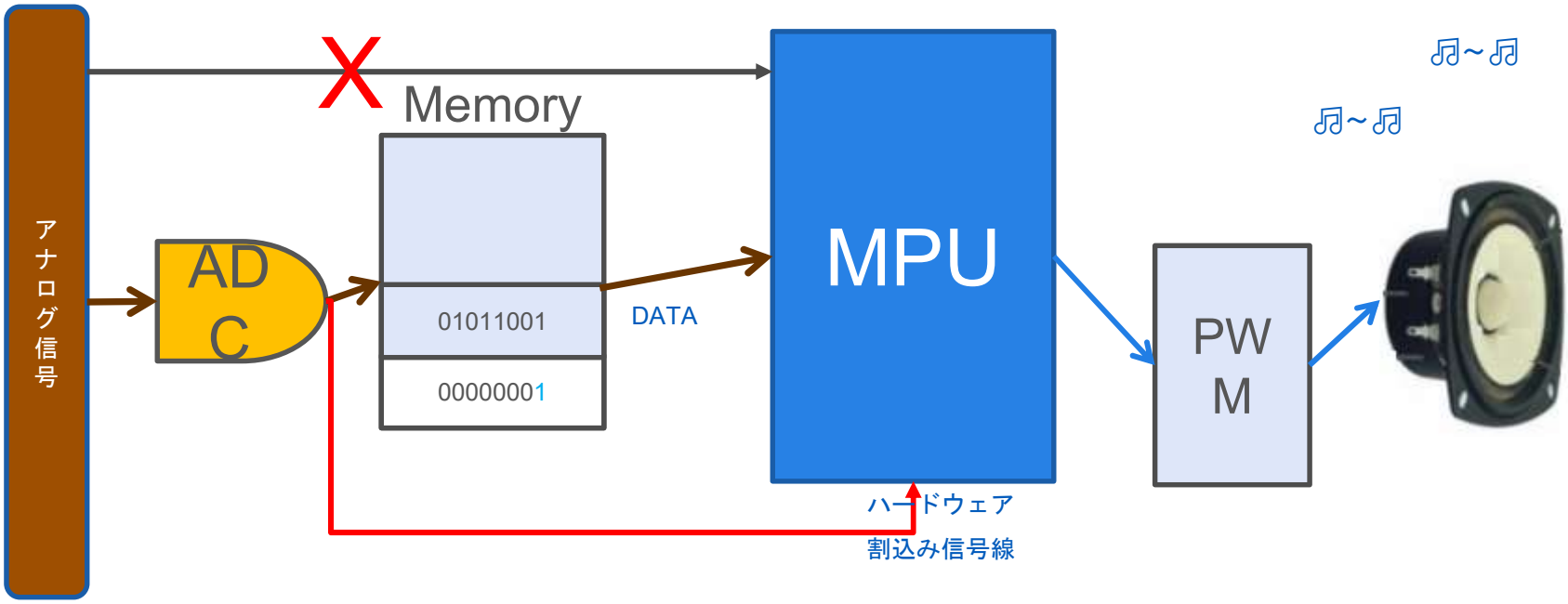
ハードウェア割込みの利点

CB22-1R2

MPUは、ADCからの割込み要求を受けると
即DATAを読みにいくので処理が速い

入力側

出力側



ハードウェア割込みでは、同時処理のプログラムの負荷の影響が非常に少ないので、正確なタイミングが必要な音楽の演奏ができる

ポーリングでは、♫と♫の間の時間が正確とれないことがある

Timer_pwm2 ラボでは
ハードウェア割込みで音階を発生

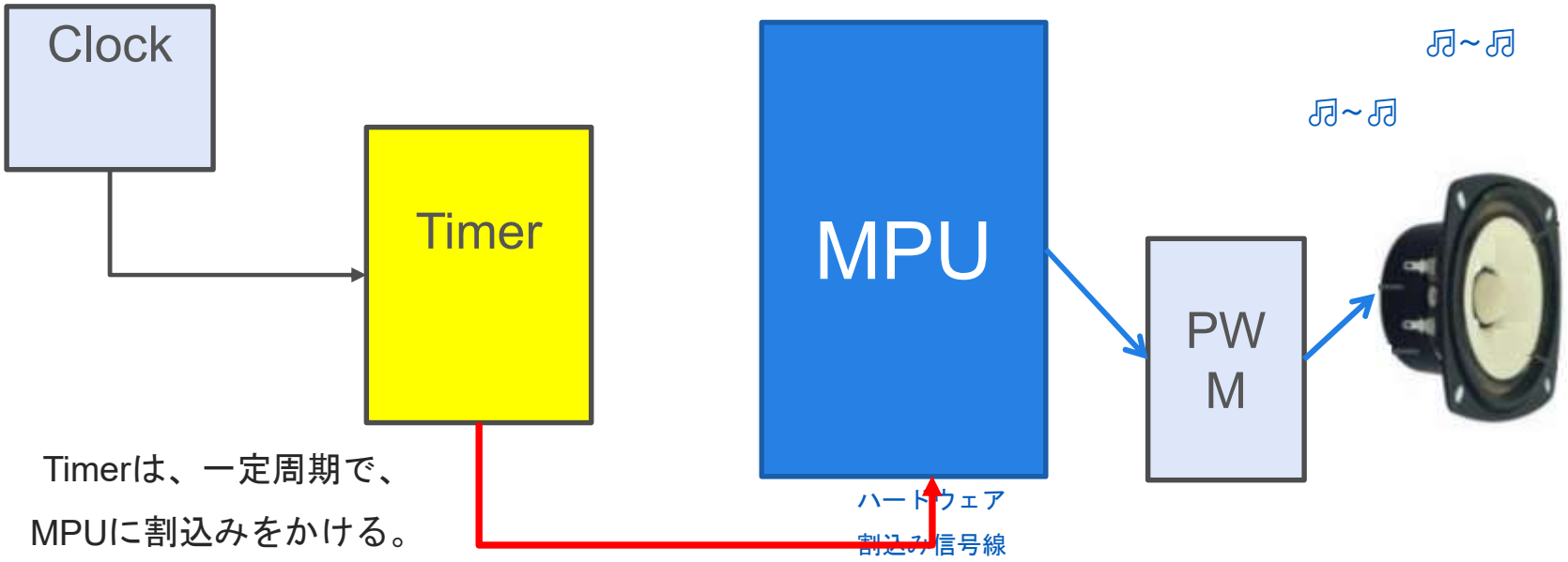
Timer_pwm2 では、main 文はほとんど

使わず割込み処理プログラム部

viod myISR で音階を発生している

入力側

出力側



Timerは、一定周期で、

MPUに割込みをかける。

MPUは割込みを最優先して

処理を行う。

時間が正確に計測できる。

でもMPU(M8)には、ハードウェア割込み

信号線は1本だけ

ハードウェア割り込み

1.ユーザーモジュールのハードウェアは、一定の条件が成立したときMPUの割り込み信号ラインを駆動する

2.MPUはソフトウェアの処理を実行中にハードウェア割り込みが発生した場合は、それまでのソフトウェアの処理を中断して、割り込みの処理に移る

3.MPUは、現在の（作業中の）レジスタの状態(値)をスタックにPUSHしてから割り込み処理ルーチンの実行に入る。割り込み処理ルーチン(ISR = Interrupt Service Routine)は、割り込みを行ったハードウェアごとに決められているジャンプアドレスにとんで特定の割り込み処理時のプログラムを実行する

4.割り込み処理が終了したら、MPUはスタックにPUSH(退避させておいた)データを順にPOP（取り出して）レジスタの状態を復帰させる。これによって割り込み前の処理状態に戻ることができる。

おまけ

複数のハードウェアから同時に割り込みが発生した場合は、プライオリティー・エンコーダで選択する。割り込みはプログラムからマスクすることもできる。

割り込み発生時のジャンプ先は割り込み・ベクタ・テーブルに書かれている。

いろいろな割り込み

1.ハードウェア割り込み

最強の割り込みは、RESETでこれはプログラムでマスクができない(Non Maskable Interrupt) マスク可能な割り込み（Maskable Interrupt ）はプログラム上で特定のレジスタ値を割り込み不許可に書き込むことで設定できる

2.ソフトウェア割り込み

オペレーティング・システムなどがサポートしてる。MS-DOSではINT21

ソフトウェアのプログラム処理で実行する

3.割り込みを使わないで同等の処理を行う方法 – (プログラムによる)ポーリング

ポーリングで、特定のレジスタの値を直接読みに行き、そのレジスタの値によって処理を決める。例えばハードウェアのタイマーが勝手にカウントしているとき、ソフトウェアから何度も何度もポーリングでカウンタ値を読み出しに行く。

タイマーが一定値になったときにプログラムで条件分岐して処理を行う

ポーリングの問題点:時間が計測しにくい.プログラムが煩雑になる.処理が遅くなる.

（ソフトウェアのポーリングのタイミングよりはるかに高速でハードウェアのタイマが動作していたらどうなるかを考えてみればよい）

ラボ timer_pwm2 手順

1.PWM16, Timer16 ユーザーモジュールを配置

2.Generate Cofigration を実行

~~3.PWM16_1INT.asm~~とTimer16_1INT.asmにロングジャンプ命令を追加してISR(インタラプト・サービス・ルーチン)に跳べるようにする

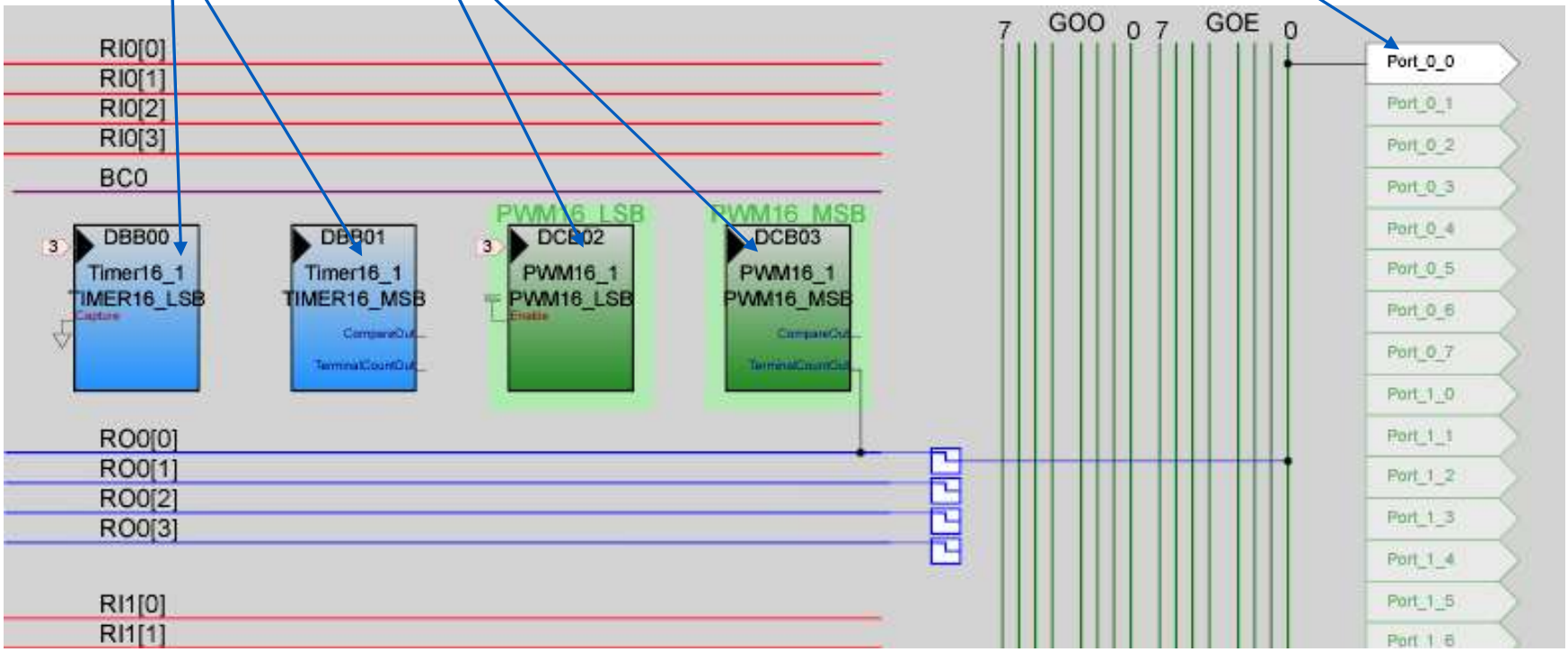
4.main.cに割り込みが発生したときのISR処理を記述する

5.音楽を演奏できる(演習例ではハ調の音階)

この演習では、操作手順の解説はしないので、これまでの演習の手順を参照して各自で設計を進めてみること

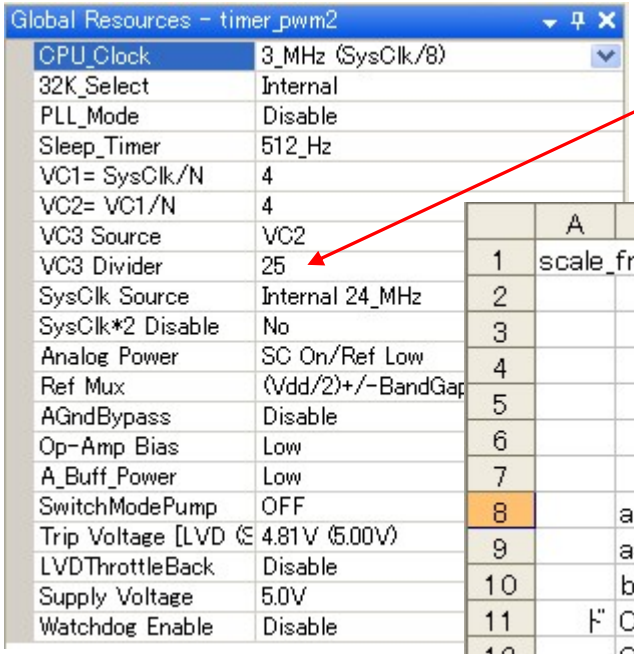
モジュールの配置と配線

Timer16とPWM16を配置してPWM16の出力をPort00に接続



これまでの演習を参考に各自でツールを使用して作業をすすめてください

音階周波数とクロック分周



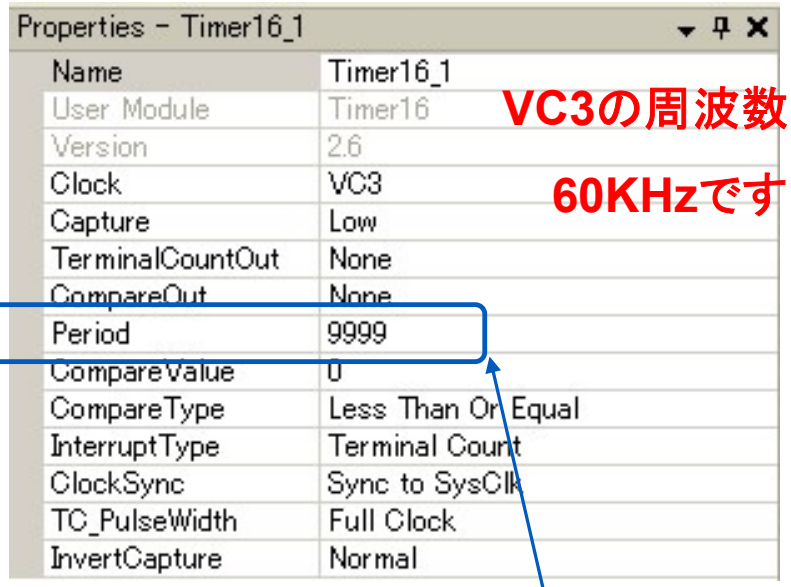
VC3の周波数はいくつになりますか？

24MHz / 4 / 4 / 25

	A	B	C	D	E	F	G	H	I
1	scale_freq.xls				Sys Clock	VC1	VC2	VC3	
2				Devide	1	4	4	25	
3		r=	1.05946	fc(KHz)	24,000	6,000	1,500	60	
4				fc(Hz)	24,000,000	6,000,000	1,500,000	60,000	var
5									
6					PWM16 Period	65536			
7			Hz	Int Hz					
8		a	220	220	*109,091*	27,273	6,818	273	
9		a#	233.0812	233	*103,004*	25,751	6,438	258	
10		b	246.9402	247	*97,166*	24,291	6,073	243	
11	ド	C	261.6233	**262**	*91,603*	**22,901**	**5,725**	**229**	D
12		C#	277.1794	277	*86,643*	21,661	5,415	217	
13	レ	D	293.6605	**294**	*81,633*	**20,408**	**5,102**	**204**	L
14		D#	311.1215	311	*77,170*	19,293	4,823	193	
15	ミ	E	329.6208	**330**	*72,727*	**18,182**	**4,545**	**182**	M
16	ファ	F	349.2201	**349**	*68,768*	**17,192**	**4,298**	**172**	F
17		F#	369.9847	370	64,865	16,216	4,054	162	
18	ソ	G	391.984	**392**	**61,224**	**15,306**	**3,827**	**153**	S
19		G#	415.2914	415	57,831	14,458	3,614	145	
20	ラ	A	440	**440**	**54,545**	**13,636**	**3,409**	**136**	R
21		A#	466.1624	466	51,502	12,876	3,219	129	
22	シ	B	493.8804	**494**	**48,583**	**12,146**	**3,036**	**121**	C
23	ド	C	523.2465	**523**	**45,889**	**11,472**	**2,868**	**115**	DD

音階の周波数と
その音を出す
ためのPWM16の
レジスタ設定値の
計算表です

Timer16とPWM16のプロパティー設定



VC3の周波数は

60KHzです

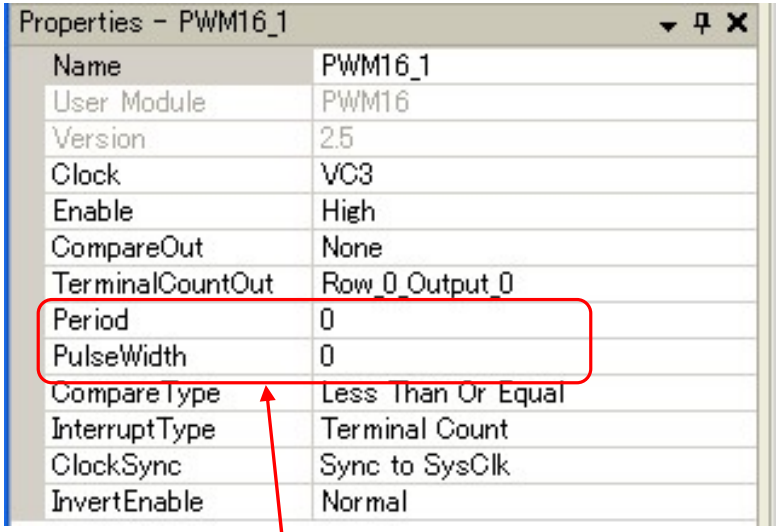
60KHzをクロックとしてタイマーは

カウントダウンしていきます

スタート値は、Period値で9999です

Period値が0になると割り込みを発生し、

再びPeriod値9999に戻ります



レジスタ値を直接プログラムで

書き換えるので、

初期値は入力しない

イニシャライズ時は音がでない

(パルス周波数がゼロ)

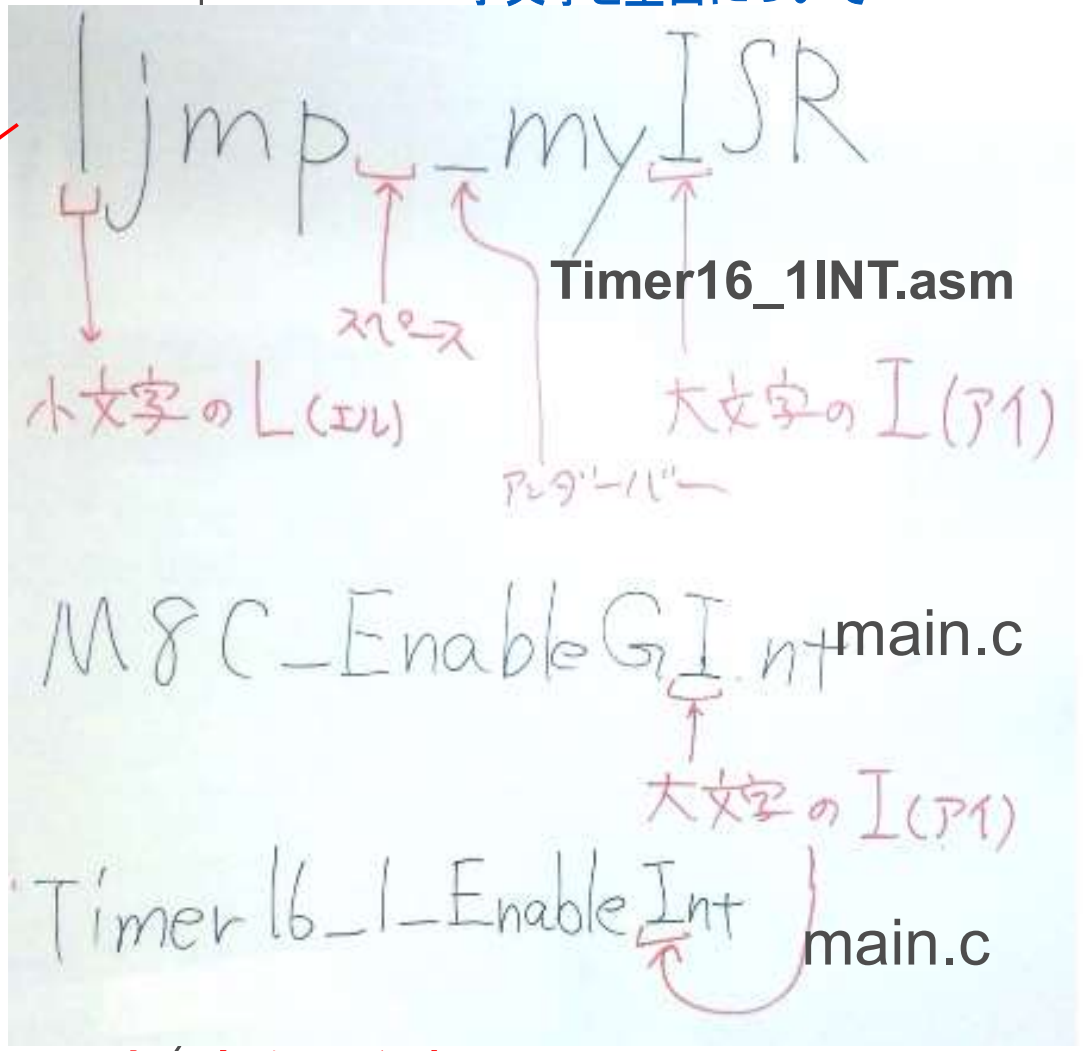
タイマーからは何sec毎に割り込みが発生しますか？

```
_Timer16_1_ISR:
   ;@PSoC_UserCode_BODY@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom assembly code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.

   ljmp _myISR

   ;---------------------------------------------------
   ; Insert your custom assembly code above this banner
   ;---------------------------------------------------
   ;---------------------------------------------------
   ; Insert a lcall to a C function below this banner
   ; and un-comment the lines between these banners
   ;---------------------------------------------------
   ;PRESERVE_CPU_CONTEXT
   ;lcall _My_C_Function
   ;RESTORE_CPU_CONTEXT
   ;---------------------------------------------------
   ; Insert a lcall to a C function above this banner
   ; and un-comment the lines between these banners
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
   reti
```

ロングジャンプ命令、割込み許可関数の大文字、小文字と空白について

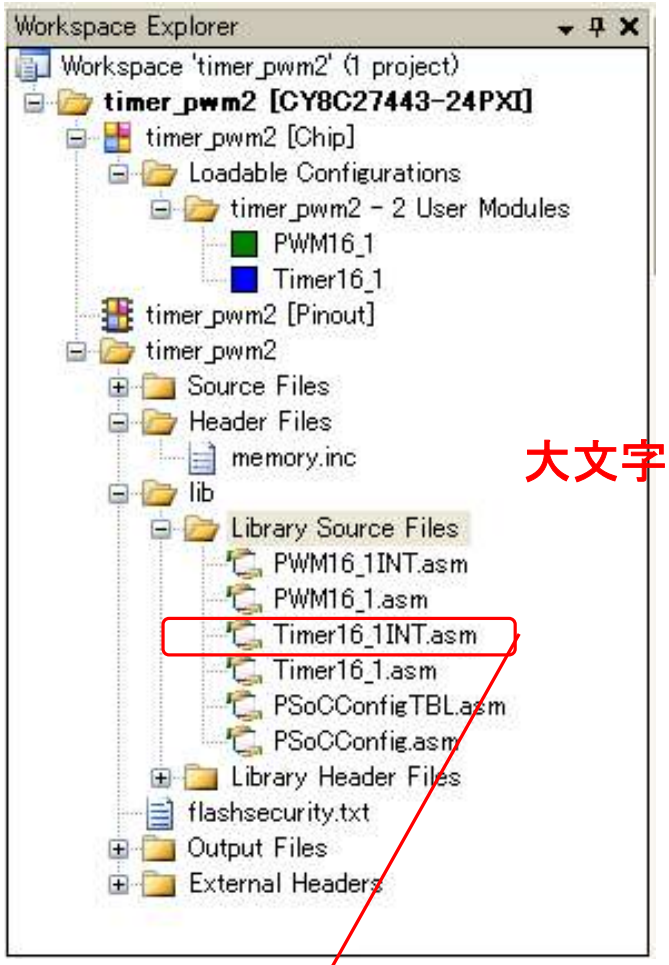


大文字、小文字、空白に注意

Generate Configrationを実行し.asmを編集

タイマーからの割り込みが発生したときにmain.c

にあるmy_ISRにロングジャンプする記述を追加します




```
_Timer16_1_ISR:
  ;@PSoC_UserCode_BODY@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom assembly code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.

ljmp _myISR

   ;---------------------------------------------------
   ; Insert your custom assembly code above this banner
   ;---------------------------------------------------
   ;---------------------------------------------------
   ; Insert a lcall to a C function below this banner
   ; and un-comment the lines between these banners
   ;---------------------------------------------------
   ;PRESERVE_CPU_CONTEXT
   ;lcall _My_C_Function
   ;RESTORE_CPU_CONTEXT
   ;---------------------------------------------------
   ; Insert a lcall to a C function above this banner
   ; and un-comment the lines between these banners
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
   reti
```

大文字、小文字、空白に注意

Timer16_1INT.asmを編集

main.cの記述

2

```
void mylSR(void)
{
TC += 1; //TC = TC +1
PWM16_1_WritePulseWidth(PW);
  if(TC == 1)
PWM16_1_WritePeriod(D);
  else if(TC == 4)
PWM16_1_WritePeriod(L);
  else if (TC == 8)
PWM16_1_WritePeriod(M);
  else if(TC == 13)
PWM16_1_WritePeriod(F);
  else if(TC == 19)
PWM16_1_WritePeriod(S);
  else if (TC == 26)
PWM16_1_WritePeriod(R);
  else if(TC == 34)
PWM16_1_WritePeriod(C);
    else if(TC == 43)
PWM16_1_WritePeriod(DD);
    else if (TC == 53)
PWM16_1_WritePeriod(SILENT);
 }
```

1

```
//-------------------------------------------------------------------------
// timer_pwm2 : REF scale_freq.xls
// Sys Clock 24MHz, PWM16 Clock = VC3/25, VC3=VC2/4, VC2=VC1/4
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#pragma interrupt_handler mylSR
int PW=125;             // Pulse Width
int SILENT=0, D=229, L=204, M=182, F=172, S=153, R=136, C=121, DD=115;
//Scale
int TC=0; //Time Count
```

1.音階の周波数対応のPulseWidth値

2.タイマー割り込み毎に実行する

　mylSR割り込み処理プログラム部

3.main()のプログラム部

　割り込み許可

3

```
void main()
{
    // Insert your main routine
code here.
M8C_EnableGInt;
PWM16_1_Start();
Timer16_1_EnableInt();
Timer16_1_Start();
while(1)
{ };
}
```

大文字、小文字、空白に注意

main.cの記述 1

```
//-----------------------------------------------------------------------
// timer_pwm2 : REF scale_freq.xls
// Sys Clock 24MHz, PWM16 Clock = VC3/25, VC3=VC2/4, VC2=VC1/4
//-----------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#pragma interrupt_handler mylSR
int PW=125;             // Pulse Width
int SILENT=0, D=229, L=204, M=182, F=172, S=153, R=136, C=121, DD=115;
//Scale
int TC=0; //Time Count
```

1

割り込み発生時の処理
ハンドラ名の記述

PWMのパルスの幅125
を定数としてPWに代入

PWMのピリオド値を定数化, これで周波数を決定する
ハ長調のドには,Dという変数を割り当て定数229を代入
これでハ長調の”ド”≒262Hzの方形波が生成される

どうして定数が229のとき262Hzになるかわからない人は、
前のスライドに戻って、よーく考え、理解してから先に進むこと

main.cの記述 2

```
void mySR(void)
{
TC += 1; //TC = TC +1
PWM16_1_WritePulseWidth(PW);
  if(TC == 1)
PWM16_1_WritePeriod(D);
  else if(TC == 4)
PWM16_1_WritePeriod(L);
  else if (TC == 8)
PWM16_1_WritePeriod(M);
  else if(TC == 13)
PWM16_1_WritePeriod(F);
  else if(TC == 19)
PWM16_1_WritePeriod(S);
  else if (TC == 26)
PWM16_1_WritePeriod(R);
  else if(TC == 34)
PWM16_1_WritePeriod(C);
    else if(TC == 43)
PWM16_1_WritePeriod(DD);
    else if (TC == 53)
PWM16_1_WritePeriod(SILENT);
 }
```

2

割り込みが発生したときに行う処理を main() の前に書く

TC(Time Count)変数を1インクリメント

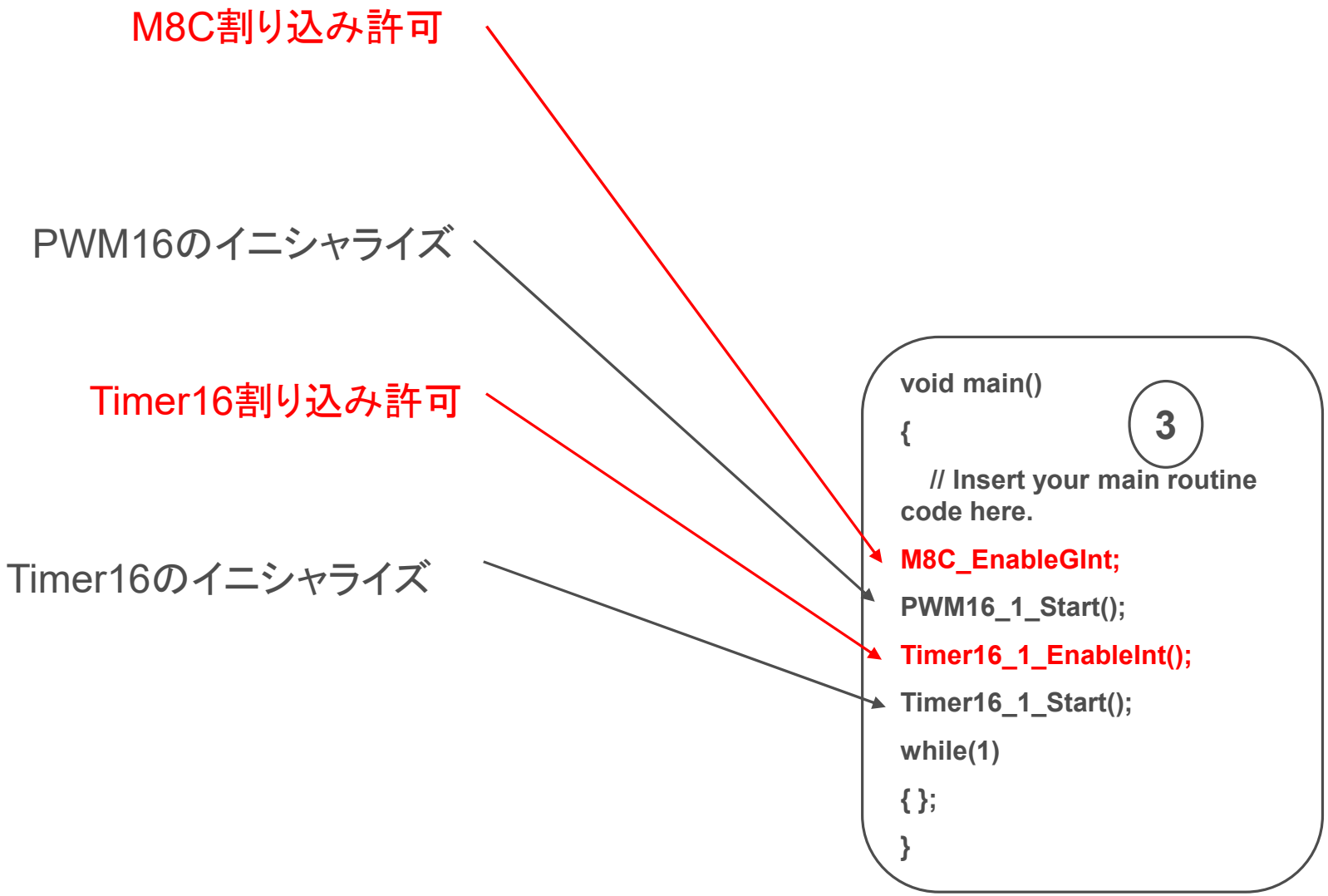
PWM16のパルスの幅は演奏中すべて同じにするので
PW(値は定数の125)をパルス幅として設定

PWM16のピリオド値としてD（定数229）を設定
これでハ長調の”ド”≒262Hzの方形波が生成される

TC(Time Count)が4になるのを待つ、
つまり割り込みが3回発生するのを待ちます
割り込みは60KHz * (9999 +1) 毎に1回
発生します

PWM16のピリオド値としてL（定数204）を設定
これでハ長調の”レ”≒294Hzの音がでます

main.cの記述 3

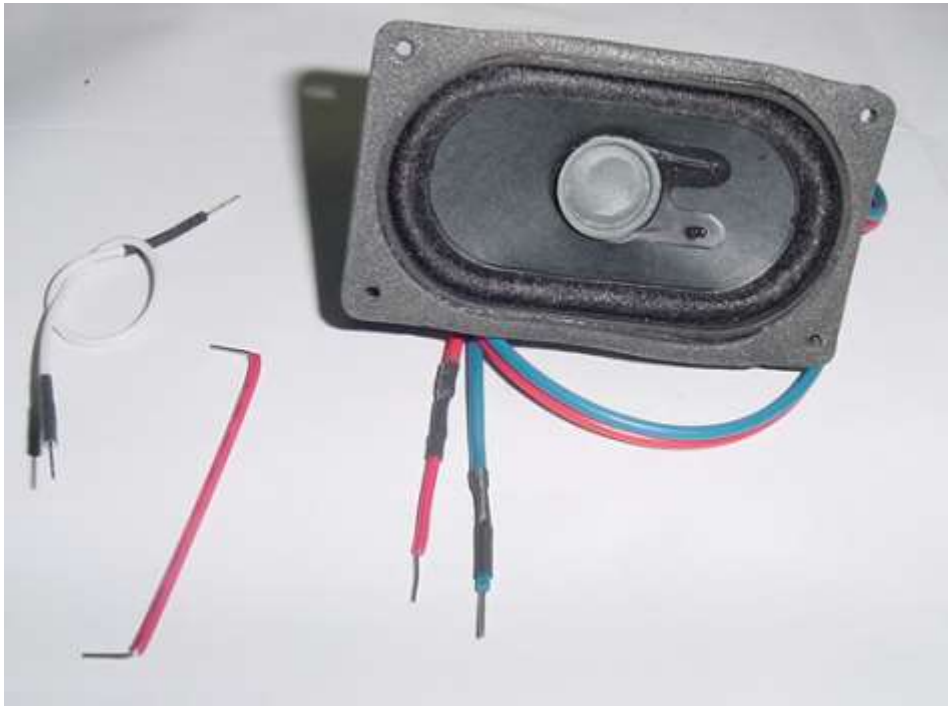

M8C割り込み許可
PWM16のイニシャライズ
Timer16割り込み許可
Timer16のイニシャライズ
void main()
{
3
// Insert your main routine code here.
M8C_EnableGInt;
PWM16_1_Start();
Timer16_1_EnableInt();
Timer16_1_Start();
while(1)
{ };
}

書き込みをして音楽(音階)を聴こう

P00とGNDにスピーカーをつなぐ

演奏が終わると停止しますがRESET SWを押すと再演します

割り込みの
メカニズムと
詳細に関しては
別資料で解説
します.



Memo

フォローアップURL (Revised)

http://mikami.a.la9.jp/meiji/MEIJI.htm

担当講師

三上廉司(みかみれんじ)

Renji_Mikami(at_mark)nifty.com (Default - Recommended)

mikami(at_mark)meiji.ac.jp (Alternative)

http://mikami.a.la9.jp/_edu.htm