

bgm driver API Reference Manual

目次

1. はじめに.....	3
2. 組み込み方.....	3
2.1. bgm_driver 初期化処理.....	4
3. API リファレンス.....	7
3.1. bgmdriver_initialize.....	7
3.2. bgmdriver_play.....	8
3.3. bgmdriver_stop.....	8
3.4. bgmdriver_check_playing.....	9
3.5. bgmdriver_fadeout.....	9
3.6. bgmdriver_mute_psg.....	10
3.7. bgmdriver_play_sound_effect.....	10
3.8. bgmdriver_interrupt_handler.....	10
4. データの構造.....	11
4.1. 効果音データ.....	11
4.2. BGM データの構造.....	12

1. はじめに

本マニュアルは、MSX 用の bgm_driver の組み込み方と、API の使い方について説明する資料です。

アセンブラ ZMA を使った記述に統一しています。

2. 組み込み方

ソースコードは、source/ に置いてある 2 つのファイルのみです（表 1 ソースコード一覧）。

表 1 ソースコード一覧

ファイル名	内容
bgmdriver.asm	bgm_driver 本体
bgmdriver_d.asm	定数宣言ファイル

ご自身のプログラム本体の「bgm_driver をリンクしたい場所」に include "bgmdriver.asm" を記述するだけで使えるようになります。

sample/の中に、実際に組み込んでいるサンプルプログラムを置いておきました。

まず、compile.bat を見てみてください。

```
0 | 1 | 2 | 3 | 4 |
1 | ..\mml_compiler\mc.exe xeviyoke.txt bgm.asm↵
2 | ..\tool\zma.exe sample.asm .\BGMSMPL.BIN↵
3 | pause↵
   | [EOF]
```

ZMA の第 1 引数が sample.asm になっていますので、これがアセンブル対象だとわかります。

次に、sample.asm を見てみましょう。ソースコードを眺めてみると、下記の部分に確かに include "bgmdriver.asm" の記述があるのが確認できると思います。

```
マイドキュメント\github\bgm_driver\sample\sample.asm - sakura 2.3.2.0
ファイル(E) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)
0 1 2 3 4 5 6 7 8 9
65 ^ call ^ ^ bgmdriver_interrupt_handler<
66 h_timi_next::<
67 ^ ret<
68 ^ ret<
69 ^ ret<
70 ^ ret<
71 ^ ret<
72 ^ endscope<
73 <
74 ; =====<
75 ; ^ BGM driver<
76 ; =====<
77 ^ include^^ "bgmdriver.asm"<
78 bgm001::<
79 ^ include^^ "bgm.asm"<
80 sound_effect001:<
81 ^ ^ db ^ ^ 32 ^ ^ ^ ^ ^ ^ ; priority [小さい方が優先]<
82 ^ ^ db ^ ^ BGM_SE_VOL<
83 ^ ^ db ^ ^ 12<
84 ^ ^ db ^ ^ BGM_SE_FREQ<
85 ^ ^ dw ^ ^ 30<
86 ^ ^ db ^ ^ BGM_SE_WAIT<
87 ^ ^ db ^ ^ 1<
88 ^ ^ db ^ ^ BGM_SE_FREQ<
```

これでリンクの記述は完了です。bgm_pdriver が組み込まれました。

次に、bgm_driver の初期化です。

2.1. bgm_driver 初期化処理

bgm_driver をリンクしただけでは、何も起こりません。bgm_driver というプログラムがくっつくだけです。

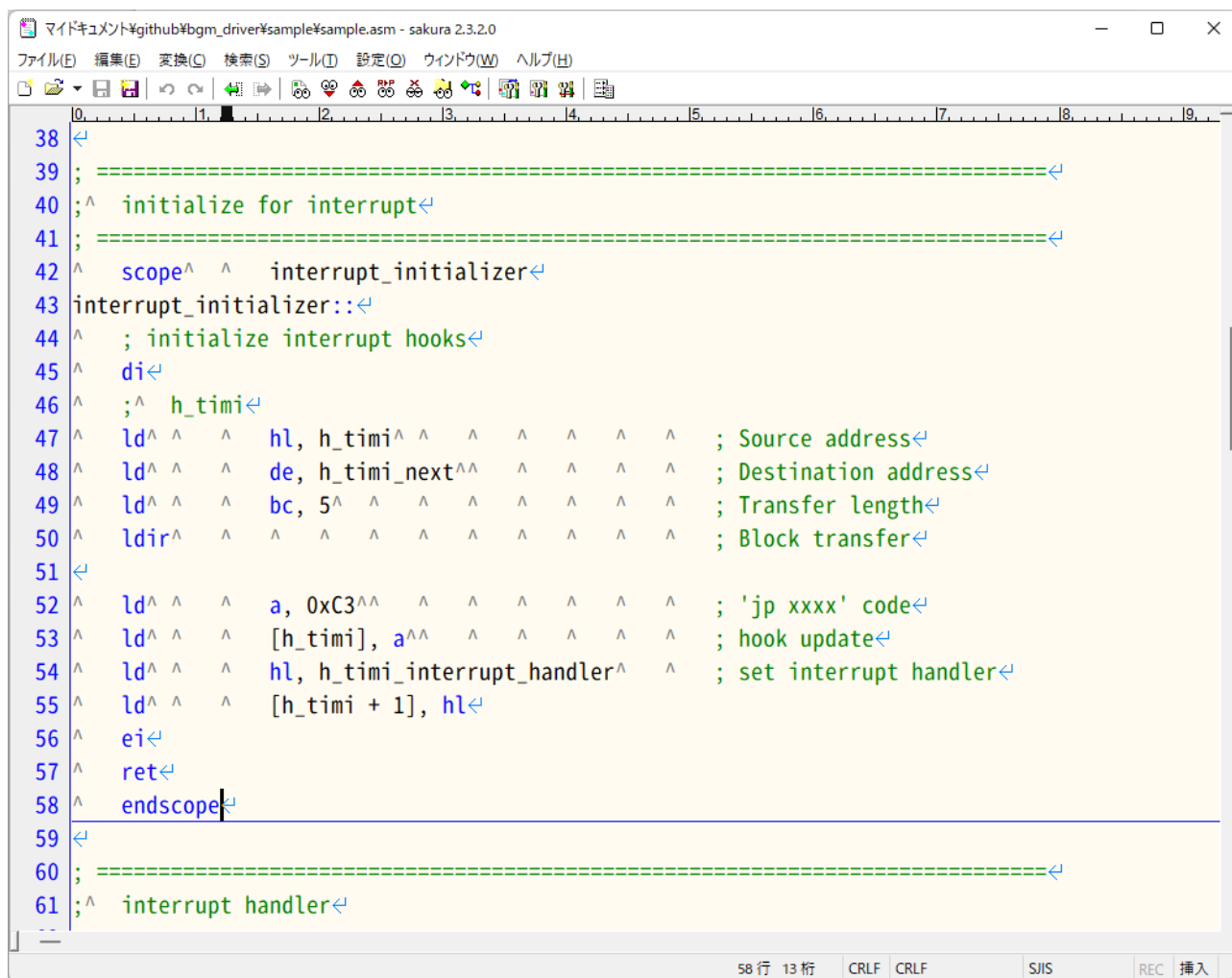
これを使える状態にするのが初期化処理です。具体的には「1/60 秒間隔の割り込み処理ルーチンから、bgm_driver の割り込み処理ルーチンを call する状態にすること」です。

MSX では、基本的に H.TIMI と呼ばれるフックに割り込み処理ルーチンを登録することで、その登録した処理ルーチンが 1/60 秒毎に呼ばれる仕組みになっています。

MSX の割り込みは、ほぼこれだけです。そのため、いろいろな処理で利用することにな

ります。bgm_driver だけが占有して使って良いことは滅多にないので、この登録処理は bgm_driver の中に組み込みませんでした。ここは自分でコーディングする必要があります。

再び sample を見てみましょう。interrupt_initializer というサブルーチンが、オーソドックスな H.TIMI フック登録方法になります。



```
38 <
39 ; =====<
40 ; initialize for interrupt<
41 ; =====<
42 ^ scope^ ^ interrupt_initializer<
43 interrupt_initializer::<
44 ^ ; initialize interrupt hooks<
45 ^ di<
46 ^ ;^ h_timi<
47 ^ ld^ ^ ^ hl, h_timi^ ^ ^ ^ ^ ^ ^ ; Source address<
48 ^ ld^ ^ ^ de, h_timi_next^^ ^ ^ ^ ^ ^ ; Destination address<
49 ^ ld^ ^ ^ bc, 5^ ^ ^ ^ ^ ^ ^ ^ ; Transfer length<
50 ^ ldir^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ; Block transfer<
51 <
52 ^ ld^ ^ ^ a, 0xC3^^ ^ ^ ^ ^ ^ ^ ^ ; 'jp xxxx' code<
53 ^ ld^ ^ ^ [h_timi], a^^ ^ ^ ^ ^ ^ ^ ^ ; hook update<
54 ^ ld^ ^ ^ hl, h_timi_interrupt_handler^ ^ ^ ; set interrupt handler<
55 ^ ld^ ^ ^ [h_timi + 1], hl<
56 ^ ei<
57 ^ ret<
58 ^ endscope<
59 <
60 ; =====<
61 ^ interrupt handler<
```

H.TIMI は、5byte あるので、それをまるごとどこかへバックアップします。

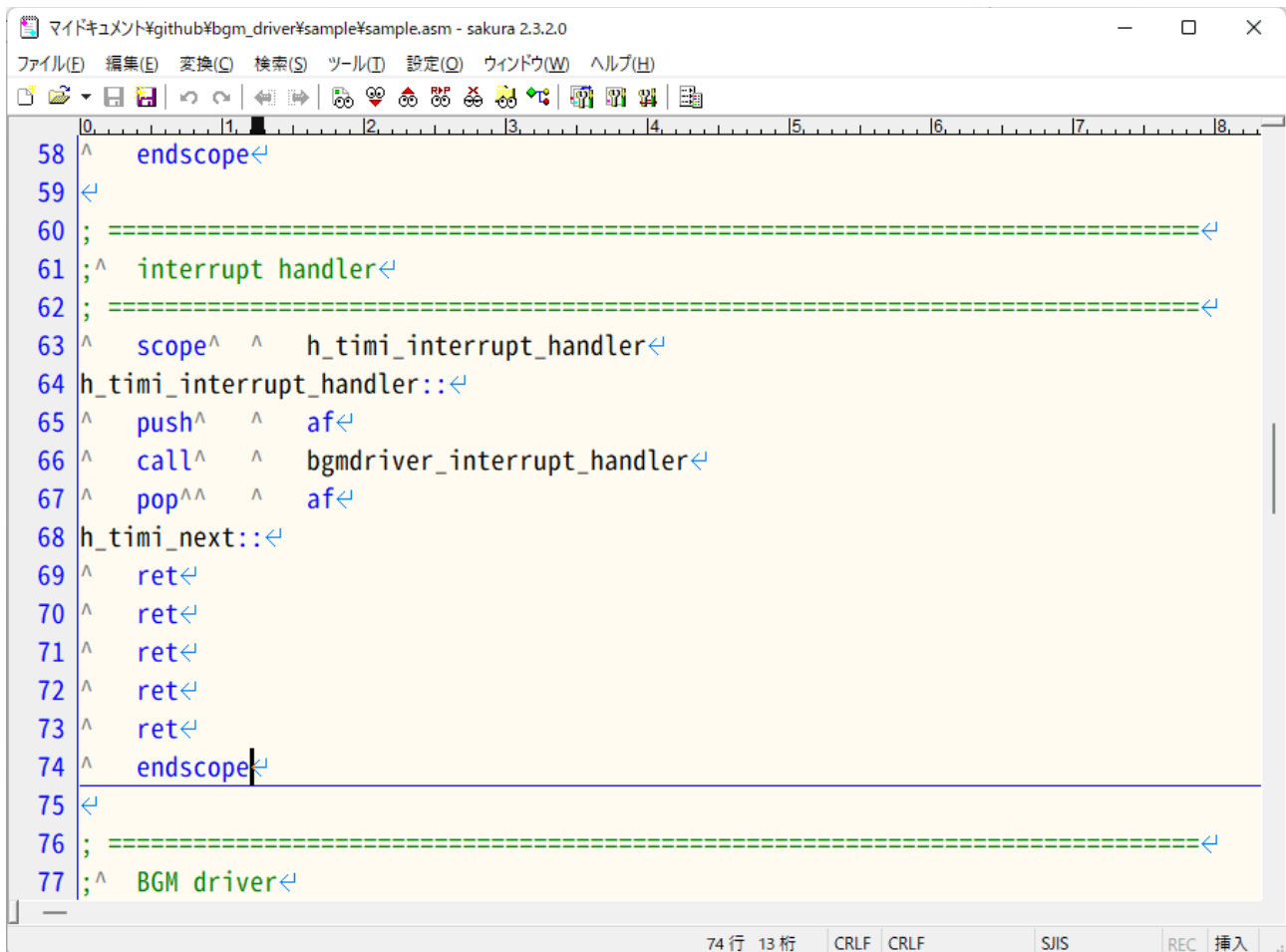
そして、代わりに「5byte 以下のサイズで、自分のコード内の割り込み処理ルーチン呼び出すコード」を H.TIMI に書き込みます。LDIR で転送しても良いし、sample のように書き込んで良いです。

いずれの場合も、書き替えている途中の中途半端な状態で割り込みが発生してしまうと、暴走の原因になりますので、書き替え中は DI して割り込みを禁止してください。

sample の場合、「JP h_timi_interrupt_handler」を書き込んでいますね。

書き替え終えたところで、EI するのを忘れずに。

そして、1/60 秒割り込みが発生すると、H.TIMI から h_timi_interrupt_handler へ飛んできます。中身を見てみましょう。



```
58 ^ endscope<
59 ^
60 ; =====<
61 ^ interrupt handler<
62 ; =====<
63 ^ scope^ ^ h_timi_interrupt_handler<
64 h_timi_interrupt_handler::<
65 ^ push^ ^ af<
66 ^ call^ ^ bgmdriver_interrupt_handler<
67 ^ pop^^ ^ af<
68 h_timi_next::<
69 ^ ret<
70 ^ ret<
71 ^ ret<
72 ^ ret<
73 ^ ret<
74 ^ endscope<
75 ^
76 ; =====<
77 ^ BGM driver<
```

call bgmdriver_interrupt_handler で、bgmdriver_interrupt_handler を呼んでいます。

bgmdriver_interrupt_handler は、bgm_driver の割り込み処理 API です。

その次に、h_timi_next に、ret が 5 つ並んでいますが、実は先ほど「元の H.TIMI をバックアップする」のところで、この 5 個の ret に上書きする形でバックアップしています。

そのため、ここは RAM でなければなりません。

自分に必要な「1/60 秒単位の処理」が終わったら、もともと登録してあった別の「1/60 秒単位の処理」を呼び出すことで、システムとして成り立つようになっています。

もし、ゲームなどで、1/60 秒単位の処理が必要であれば、call bgmdriver_interrupt_handler の前か後かに、その処理を挿入してください。

ここまでやって、ようやく bgm_driver が使える状態になります。

H.TIMI は、A レジスタに VDP ステータスレジスタ#0 の値が格納された状態で呼ばれます。VDP ステータスレジスタ#0 は読むことで、垂直同期割り込み(1/60 秒周期)を解除するように VDP へ通知する処理も兼ねるため、BIOS 内で「垂直同期割り込みであるか否か」の判定のために読んだステータスレジスタ#0 の値を A レジスタに保存しています。H.TIMI でステータスレジスタ#0 を読み直しても同じ値が返ってこないためです。

これを期待している H.TIMI ルーチンが存在する可能性があるので、push af, pop af で A レジスタの値を保存しています。

それ以外のレジスタは、BIOS 側の H.TIMI 呼び出し元の方で保存されているため、自分で保存する必要はありません。

これ以降は、必要な API を呼ぶことで、BGM を演奏したり停止したり、効果音を再生したりできるようになります。

3. API リファレンス

表 1 API 一覧の API を利用できます。

表 1 API 一覧

API 名	概要
3.1. bgmdriver_initialize	初期化处理
3.2. bgmdriver_play	演奏開始処理
3.3. bgmdriver_stop	演奏停止処理
3.4. bgmdriver_check_playing	演奏中チェック
3.5. bgmdriver_fadeout	フェードアウト
3.6. bgmdriver_mute_psg	音停止処理
3.7. bgmdriver_play_sound_effect	効果音開始処理
3.8. bgmdriver_interrupt_handler	演奏処理ルーチン

3.1. bgmdriver_initialize

このルーチンは、演奏用ワークエリアを初期化します。

演奏用ワークエリアは、デフォルト状態で初期化済みであるため、通常はこのルーチンと呼ぶ必要はありません。

H.TIMI フックを復元して強制的に停止させた後に、再度やり直したい場合などに使います。

このルーチンと呼ぶと、下記のレジスタの内容が破壊されます。

a, b, c, d, e, f, h, l

3.2. bgmdriver_play

HL レジスタで示される BGM データの演奏を開始します。

すでに BGM が演奏中の場合は停止し、指定の BGM データを開始します。

HL レジスタに格納する値は、BGM データの先頭アドレスです。BGMP データの途中のアドレスや、BGM データでないアドレスを指定した場合の動作は保証しません(暴走します)。

このルーチンと呼ぶと、下記のレジスタの内容が破壊されます。

a, b, c, d, e, f, h, l, ix

呼び出し例：TITLE_BGM の演奏開始

```
ld    hl, TITLE_BGM
```

```
call  bgmdriver_play
```

3.3. bgmdriver_stop

BGM が演奏中であれば停止します。

演奏中でない場合に呼んでも問題ありません。

このルーチンと呼ぶと、下記のレジスタの内容が破壊されます。

a, b, c, d, e, f, h, l

呼び出し例：演奏停止

```
call bgmdriver_stop
```

3.4. bgmdriver_check_playing

BGM が演奏中であるか否かを調べて Z フラグに返します。

Z フラグ 1 なら停止中、0 なら演奏中

このルーチンと呼ぶと、下記のレジスタの内容が破壊されます。

a, f, ix

呼び出し例：フェードアウトして停止するまで待機する

```
ld    a, 10
```

```
call  bgmdriver_fadeout      ; フェードアウト開始
```

wait_bgm_end:

```
call  bgmdriver_check_playing ; 演奏停止チェック
```

```
jr    nz, wait_bgm_end      ; 演奏中であれば wait_bgm_end に戻る
```

3.5. bgmdriver_fadeout

演奏中の BGM をフェードアウト後に停止します。

A レジスタにフェードアウトの速度としてウェイト時間を指定します。フェードアウトの音量が下がる間隔を、1/60 秒単位で指定するため、数字が大きいほど遅くなります。

1～255 の範囲で指定します。0 を指定するのは禁止です。

このルーチンと呼ぶと、下記のレジスタの内容が破壊されます。

a, f

呼び出し例は、3.4. bgmdriver_check_playing を参照ください。

3.6. bgmdriver_mute_psg

PSG の音声出力を停止します。

3.3. bgmdriver_stop を呼ばずに、H.TIMI を復元して 3.7.

bgmdriver_play_sound_effect が呼ばれないようにして止めた場合、音が鳴りっぱなしになる場合がある。

これを停止する処理である。

bgmdriver の内部変数は変更せず、あくまで PSG レジスタを無音に初期化する。

通常、使う必要の無いルーチンである。

このルーチンを呼ぶと、下記のレジスタの内容が破壊されます。

a, b, c, d, e, f, h, l

3.7. bgmdriver_play_sound_effect

HL レジスタで指定される効果音データを再生します。

BGM の演奏は止めずに効果音を再生します。PSG ch.1~3 のうち、ch.3 を利用して効果音を再生します。そのため、和音による効果音は実現できません。

このルーチンを呼ぶと、下記のレジスタの内容が破壊されます。

a, f

3.8. bgmdriver_interrupt_handler

演奏ルーチンの本体です。このルーチンは、1/60 秒 (厳密には、1/59.94 秒) 単位で呼び出されることを期待しています。2.1. bgm_driver 初期化処理に記載した方法などで、1/60 秒単位で呼び出されるようにしておいてください。

一方で、この関数を 1/60 秒よりも短い周期で呼び出すことで早送り、長い周期で呼び出すことでゆっくり再生させることもできます。BGM だけでなく、効果音も影響を受けるのでご注意ください。

このルーチンを呼ぶと、下記のレジスタの内容が破壊されます。

全て

4. データの構造

通常、BGM データは別途用意してある MML compiler mc.exe で生成することになるため、データ構造について知る必要はありません。

一方で、効果音についてはデータ構造を知らないと作ることができませんので、効果音の方を先に説明したいと思います。

4.1. 効果音データ

効果音データは、PSG ch.3 を使って再生されます。「ch.1～3 を同時にならして厚みのある効果音を鳴らす」というのはできませんのでご注意ください。

効果音データは、bgmdriver_d.asm に定義されている定数を羅列したものになります。その構造を、図.1 に示します。

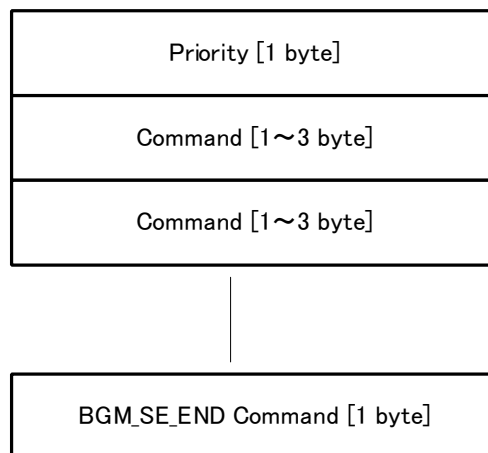


図.1 効果音データの構造

T.B.D.

4.2. BGM データの構造

通常、これを意識する必要はありませんが、下記の用途に役立つため掲載します。

(1) mc.exe よりも使いやすい MML コンパイラを自作したい人

(2) データサイズを抑えたい人

T.B.D.