

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.....	7
3.3. bgmdriver_stop.....	7
3.4. bgmdriver_check_playing.....	7
3.5. bgmdriver_fadeout.....	8
3.6. bgmdriver_mute_psg.....	8
3.7. bgmdriver_play_sound_effect.....	8
3.8. bgmdriver_interrupt_handler.....	8

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。

```
マイドキュメント\github\bgm_driver\sample\sample.asm - sakura 2.3.2.0
ファイル(F) 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)
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。

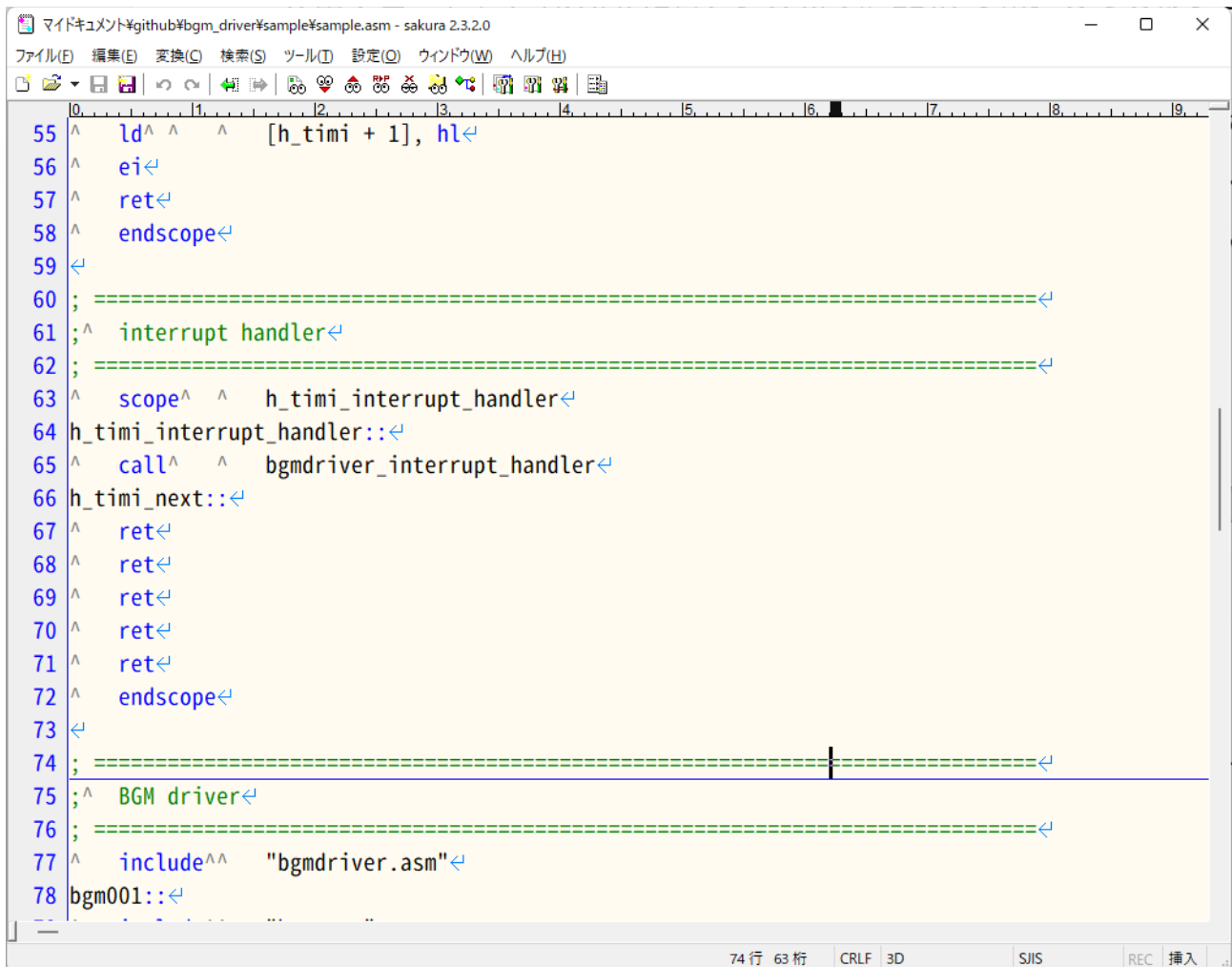
そして、代わりに「5byte 以下のサイズで、自分のコード内の割り込み処理ルーチン呼び出すコード」を H.TIMI に書き込みます。LDIR で転送しても良いし、sample のように書き込んでも良いです。

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。



```
55 ^ ld^ ^ ^ [h_timi + 1], hl^
56 ^ ei^
57 ^ ret^
58 ^ endscope^
59 ^
60 ; =====<
61 ; ^ interrupt handler<
62 ; =====<
63 ^ scope^ ^ h_timi_interrupt_handler^
64 h_timi_interrupt_handler::^
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::^
```

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。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。

あとは、必要な API を呼ぶだけです。

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
bgmdriver_initialize	初期化处理
bgmdriver_play	演奏開始処理
bgmdriver_stop	演奏停止処理
bgmdriver_check_playing	演奏中チェック
bgmdriver_fadeout	フェードアウト
bgmdriver_mute_psg	音停止処理
bgmdriver_play_sound_effect	効果音開始処理
bgmdriver_interrupt_handler	演奏処理ルーチン

3.1. bgmdriver_initialize

3.2. bgmdriver_play

3.3. bgmdriver_stop

3.4. bgmdriver_check_playing

3.5. bgmdriver_fadeout

3.6. bgmdriver_mute_psg

3.7. bgmdriver_play_sound_effect

3.8. bgmdriver_interrupt_handler