

Wonder Swan
Total Sound Driver

Version 0.xx

Developer's Manual

Digitalis

始めに

WTD Professional Edition 又は Developer's Edition をお買いあげ頂き、まことに有り難うございます。

本製品は、WTD を使用したアプリケーション開発を行う方々の為のエディションです。また、本書の内容に関するご質問、技術サポートは、FSPNET でのみ受け付けております。どうぞご利用下さい。

本製品のアセンブルソースリストは、Microsoft 16bit Device Driver Kit (16bitDDK) に含まれる Microsoft Macro Assembler (MASM) Version 5.10 を用いることでアセンブルすることが可能です。MASM Ver,5.10 は、Microsoft Developer's Network (msdn) Professional Subscription 以上のグレードを購読することで入手可能です。詳細は、msdn 事務局にお問い合わせ下さい。

本書に記載されている技術情報および、FSPNET で得られた技術情報は、如何なる手段を用いても公開・配布を禁止します。但し、取扱説明書の「制約事項」の条文による行為はこの限りではありません。

Microsoft macro assembler は、米国 Microsoft 社およびその他の登録商標です。

Microsoft Developer's Network は、米国 Microsoft 社およびその他の登録商標です。

Microsoft 16bitDDK は、米国 Microsoft 社およびその他の登録商標です。

1. 目次

1. 目次	p. 2
2. 制御用ワークエリア構造	p. 3
2-1. ユーザ用ワーク	p. 4
2-2. WS 内蔵 PCM 音色データ	p. 5
2-3. エンベロープ制御用ワーク	p. 5
2-4. パート共通構造体	p. 7
2-5. パート個別構造体	p. 9
3. IL ファンクション	p. 14
4. 曲・効果音データフォーマット	p. 25
4-1. ヘッダーフォーマット	p. 25
4-2. 拡張ヘッダー	p. 26
4-3. 定義データ	p. 26
4-4. 演奏データ	p. 27
4-4-1. 演奏データ書式	p. 27
4-4-2. 演奏データコマンド	p. 28
5. PCM Voice データフォーマット	p. 34
5-1. ヘッダーフォーマット	p. 34
5-2. データフォーマット	p. 34
6. 音色ファイルデータフォーマット	p. 35
6-1. ヘッダーフォーマット	p. 35
6-2. データフォーマット	p. 35
7. 技術サポートについて	p. 35

2. 制御用ワークエリア構造

本製品は、ユーザプロセス用 SRAM 領域のヒープエリアの先頭から 4096[byte]を制御用ワークエリアとして使用します。

下記の表は、制御用ワークのメモリマップである。

変更の可能性は無いとは限らないので、アドレスは本製品付属のヘッダーファイルにて定義されている定数を使用することが望ましい。

0000h	ユーザ用ワーク
0100h	音色格納バッファ
0200h	エンベロープ格納バッファ
0380h	共通ワーク構造体
0400h	個別ワーク構造体 (効果音 、 Pcm Voice)
0600h	個別ワーク構造体 (演奏)
0FFFh	

図 2－1 制御用ワークエリア・メモリマップ

2-1. ユーザ用ワーク

主にゲームなどのフラグ管理を目的とする領域である。

この領域は、ユーザプログラム及び、曲データからのランダムアクセスが可能である。

曲データ中からは、以下の機能を有す。

- ・ コマンドの引数をワークから参照する。この時、引数は 8bit のアドレスを指定し、データは 8bit である。引数が 16bit の命令ではワーク参照はできない。
- ・ 任意のアドレスにデータを代入(mov,LD)することができる。
- ・ 任意のアドレスのデータを加算(add,ADD)することができる。
- ・ 任意のアドレスのデータを減算(sub,SUB)することができる。
- ・ 任意のアドレスのデータを論理積(and,AND)することができる。
- ・ 任意のアドレスのデータを論理和(or,OR)することができる。
- ・ 任意のアドレスのデータを排他的論理和(xor,XOR)することができる。
- ・ 任意のアドレスのデータをビットセット(SET)することができる。
- ・ 任意のアドレスのデータをビットリセット(RST)することができる。
- ・ 任意のアドレスのデータを比較(cmp,CP)することができる。
- ・ 任意のアドレスのデータをテスト(test)することができる。
- ・ 以上の演算結果による条件ループジャンプをすることができる。フラグは、Cy(オーバーフロー)、Ze(演算結果 0)が用意されている。

()中の記号は、小文字が 8086 系の命令、大文字が Z80 系の命令コードに相当する。

2-2. WS 内蔵 PCM 音色データ

WS 内蔵の PCM 波形データを記憶する領域である。

WTD 内部に 0～15 番までの 16 個の音色を登録することが可能である。

それぞれの音色は、16byte 1 組みのデータでありそれぞれが領域の先頭から順に 16 個格納されている。計 256byte の領域を有す。

2-3. エンベロープ制御用ワーク

本ドライバーは、ソフトウェアレベルでのエンベロープに対応しており、音程、音量ともにその機能を有している。

エンベロープは、0～23 番まで登録することが可能であり、音程、音量また、演奏・効果音で共通のワークエリアを使用している。

それぞれの音色は、16byte 1 組みのデータでありそれぞれが領域の先頭から順に 24 個格納されている。計 384byte の領域を有す。

パラメータについて (次ページの表・図を参照)

- ・傾き : 1 回で変化するレベルの度合い
- ・速度 : 1 回の変化に要する時間 (単位[step])。上位 1bit は、傾き方向。
- ・レベル : 音程・音量の度合い。
 - 音程 0(低い)～128(中央)～255(高い)
 - 音量 0(無音)～255(最大)

以下に、エンベロープデータの構造を示す。

表 2 - 3 エンベロープパラメータ

引数	名前	内容	範囲	初期値
<i>no</i>	number	音色番号	0～15	255
<i>fl</i>	First Level	Key On 時の出力レベル	0(小)～255(大)	-
<i>ar</i>	Attack Rate	Key On 時立ち上がりの早さ	0(遅)～255(早)	-
<i>as</i>	Attack Speed	Key On 時立ち上がりの早さ	※ 1	1
<i>al</i>	Attack Level	Key On 時立ち上がりのレベル	0(小)～255(大)	255
<i>dr</i>	Decay Rate	1 回目立下りの速さ	0(遅)～255(早)	-
<i>ds</i>	Decay Speed	1 回目立下りの速さ	※ 1	-1
<i>dl</i>	Decay Level	1 回目立下りのレベル	0(小)～255(大)	-
<i>sr</i>	Sustain Rate	2 回目立下りの速さ	0(遅)～255(早)	-
<i>ss</i>	Sustain Speed	2 回目立下りの速さ	※ 1	-1
<i>sl</i>	Sustain Level	2 回目立下りのレベル	0(小)～255(大)	0
<i>rr</i>	Release Rate	Key Off 後の立下りの速さ	0(遅)～255(早)	-
<i>rs</i>	Release Speed	Key Off 後の立下りの速さ	※ 1	-1
<i>rl</i>	Release Level	Key Off 後の立下りのレベル	0(小)～255(大)	0
※ 1	Speed について	増加方向	1(早)～127(遅)	
		減衰方向	-1(早)～-128(遅)	

以下に、エンベロープ形状を示す。

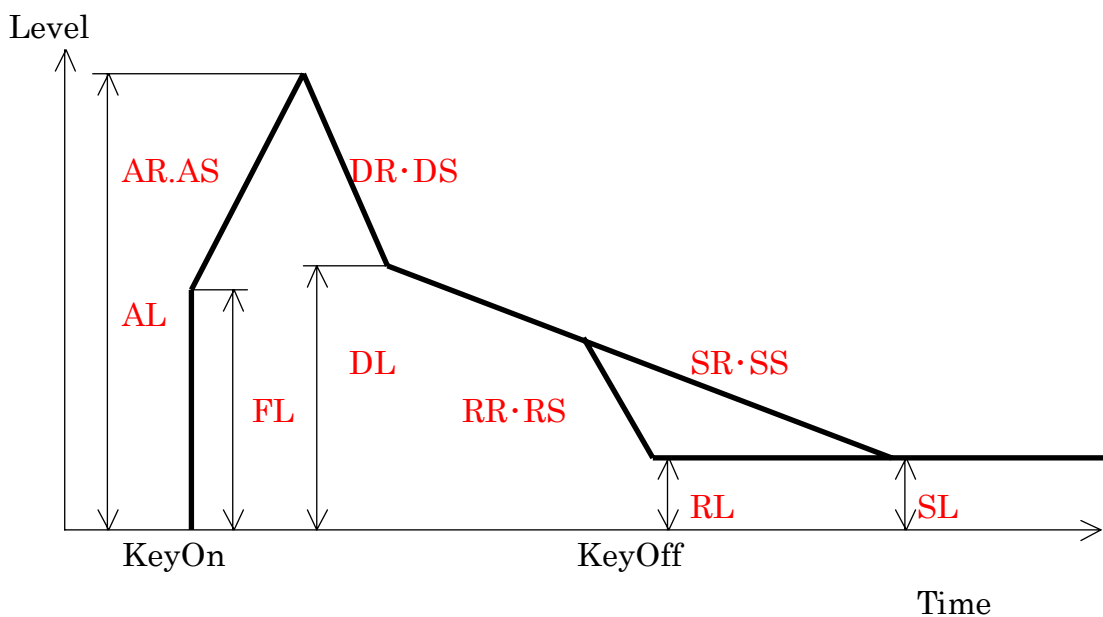


図 2 - 3 エンベロープ形状

2-4. パート共通構造体

WTDでは、システム全体の制御として「パート共通構造体」を有している。
以下にその内容を示す。

●構造体定義

```
typedef struct {  
    int      Flag;                /* Flag */  
    int      ProgramOffset;       /* Program Offset Address */  
    int      ProgramSegment;      /* Program Segment Address */  
    int      MusicOffset;         /* Music Offset Address */  
    int      MusicSegment;        /* Music Segment Address */  
    int      EffectOffset;        /* Effect Offset Address */  
    int      EffectSegment;       /* Effect Segment Address */  
    int      PcmOffset;           /* Pcm Offset Address */  
    int      PcmSegment;          /* Pcm Segment Address */  
    int      StayOutMask;         /* 常駐解除禁止 */  
    int      OldIntvector[2];     /* 前の割り込みベクタ */  
    int      OldCommBaudrate;     /* 元のボーレート */  
    int      OldCommCansel;       /* 元のキャンセルパターン */  
    char      OldSoundMode;       /* 元のサウンドモード設定 */  
    char      OldSoundOut;        /* 元のサウンド出力設定 */  
    int      OldSoundSweep;       /* 元のサウンド・sweep */  
    char      OldSoundNoise;      /* 元のサウンド・noise */  
    char      VolumeMusic;        /* ソフトウェア音量 演奏 */  
    char      VolumeEffect;       /* ソフトウェア音量 効果音 */  
    char      VolumePCM;          /* ソフトウェア音量 PCM */  
    int      Tempo;               /* テンポ */  
    int      TempoCounter;        /* テンポカウンタ */  
    char      MusicPart;          /* 演奏のパート数(最大 20) */  
    char      EffectPart;         /* 効果音のパート数(最大 3) */  
} Wtd_Sys;
```


表 2-4-1 パート共通構造体

型	Name	内容
int	Flag	bit 1 曲データを演奏中
		bit 2 効果音を発生中
		bit 3 PCM Voice を発声中
		bit 4 PCM Voice 終了
		bit 5 MIDI マスク(現在未使用)
		bit 6 未使用
		bit 7 未使用
		bit 8 未使用
		bit 9 ch1 のチャンネルモードの状態
		bit 10 ch2 のチャンネルモードの状態
		bit 11 ch3 のチャンネルモードの状態
		bit 12 ch4 のチャンネルモードの状態
		bit 13 効果音で ch1 を使用中
		bit 14 効果音で ch2 を使用中
		bit 15 効果音で ch3 を使用中
		bit 16 効果音で ch4 を使用中
int	ProgramOffset	ドライバのあるオフセットアドレス
int	ProgramSegment	ドライバのあるセグメントアドレス
int	MusicOffset	曲データのあるオフセットアドレス
int	MusicSegment	曲データのあるセグメントアドレス
int	EffectOffset	効果音データのあるオフセットアドレス
int	EffectSegment	効果音データのあるセグメントアドレス
int	PcmOffset	PCM Voice データのあるオフセットアドレス
int	PcmSegment	PCM Voice データのあるセグメントアドレス
int	StayOutMask	常駐解除の禁止状態(0:常駐解除可)(未使用)
int	OldIntvector[2]	常駐前の割り込みベクトル
int	OldCommBaudrate	常駐前の通信速度(未使用)
int	OldCommCansel	常駐前の通信のキャンセルキーパターン(未使用)
char	OldSoundMode	常駐前の WS 内蔵 PCM のチャンネルモード(未使用)
char	OldSoundOut	常駐前の WS 内蔵 PCM の出力モード(未使用)
int	OldSoundSweep	常駐前の WS 内蔵 PCM のスウィープ値(未使用)
char	OldSoundNoise	常駐前の WS 内蔵 PCM のノイズ値(未使用)
char	VolumeMusic	演奏の音量 (0(無音)~255(最大))
char	VolumeEffect	効果音の音量 (0(無音)~255(最大))
char	VolumePCM	PCM Voice の音量(bit0,1:右の音量/bit2,3:左の音量)

int	Tempo	現在の割り込み周期(テンポに起因)
int	TempoCounter	PCM Voice 発声時のタイマーエミュレート用
char	MusicPart	演奏している曲のパート数。
char	EffectPart	発生している効果音のパート数。

2-5. パート個別構造体

演奏処理用 20[ch]、効果音処理用 3[ch]、PCM Voice 処理用 1[ch]のそれぞれのトラックは、パート共通構造体とは別に個別の構造体「パート個別構造体」を有している。

以下にその内容を示す。

●構造体定義

```
typedef struct {
    int    FlagControl;           /* 00-01   フラグ */
    char   FlagTai;              /* 02      タイフラグ */
    char   FlagSharp;           /* 03      #フラグ */
    char   FlagFlat;            /* 04      bフラグ */
    char   Channel;             /* 05      チャンネルトラック */
    int    Address;              /* 06-07   演奏中のアドレス */
    char   LoopCount[8];        /* 08-0F   ループ回数/PCM Adr */
    char   LoopCountPointer;     /* 10      上記ポインタ */
    char   Program;              /* 11      音色 */
    char   SweepLevel;          /* 12      スウィープ値/ノイズモード */
    char   SweepTime;           /* 13      スウィープ時間 */
    int    Leng;                /* 14-15   音長 */
    int    LengCounter;         /* 16-17   音長カウンター */
    int    LengDefault;         /* 18-19   デフォルトの音長 */
    int    KeyOffTime;          /* 1A-1B   ゲートタイム制御値 */
    int    KeyOnDelay;          /* 1C-1D   キーオンデレイ */
    char   KeyShift;            /* 1E      移調値 */
    char   Key;                 /* 1F      音程 */
    char   KeySet[8];           /* 20-27   出力した音程 */
    char   KeySetPointer;       /* 28      上記ポインタ */
    char   AcsentVelocity;      /* 29      アクセント値 */
    int    BendSet;             /* 2A-2B   出力した周波数 */
    int    Bend;                /* 2C-2D   指定周波数 */
    int    BendDetune;          /* 2E-2F   ディチューン */
}
```

char	BendLfoDecayRate;	/* 30	LFO ディケイレート */
char	BendLfoSpeedRate;	/* 31	LFO スピード */
int	BendLfoLevel;	/* 32-33	LFO 変位レベル */
char	BendLfoCount;	/* 34	LFO 変化回数 */
char	BendLfoRateCounter;	/* 35	LFO レイト用カウンタ */
int	BendLfoLevelSet;	/* 36-37	LFO 現在の変位レベル */
char	BendLfoCountCounter;	/* 38	LFO 回数用カウンタ */
char	BendEmbAddress;	/* 39	EMB 番号 (アドレス) */
char	BendEmbRate;	/* 3A	EMB レイト */
char	BendEmbCounter;	/* 3B	EMB カウンター */
char	BendEmbMaxMin;	/* 3C	EMB 最大／最小値 */
char	BendEmbLevelSet;	/* 3D	EMB 変位レベル */
char	Pan;	/* 3E	パンポット */
char	ExprPanSet;	/* 3F	出力ゲイン係数 */
char	ExprSet;	/* 40	出力音量/エクスプレッション */
char	Expr;	/* 41	音量/エクスプレッション */
int	ExprDetune;	/* 42-43	音量増減値 */
char	ExprLfoDecayRate;	/* 44	LFO ディケイレート */
char	ExprLfoSpeedRate;	/* 45	LFO スピード */
int	ExprLfoLevel;	/* 46-47	LFO 変位レベル */
char	ExprLfoCount;	/* 48	LFO 変化回数 */
char	ExprLfoRateCounter;	/* 49	LFO レイト用カウンタ */
int	ExprLfoLevelSet;	/* 4A-4B	LFO 現在の変位レベル */
char	ExprLfoCountCounter;	/* 4C	LFO 回数用カウンタ */
char	ExprEmbAddress;	/* 4D	EMB 番号 (アドレス) */
char	ExprEmbRate;	/* 4E	EMB レイト */
char	ExprEmbCounter;	/* 4F	EMB カウンター */
char	ExprEmbMaxMin;	/* 50	EMB 最大／最小値 */
char	ExprEmbLevelSet;	/* 51	EMB 変位レベル */
char	WorkAddress;	/* 52	ワークアドレス上位 8BIT */
char	OctaveSet;	/* 53	出力したオクターブ */
char	Octave;	/* 54	指定オクターブ */
char	GateTime8;	/* 55	ゲートタイム (Q,U) */
int	GateTimeStepLast;	/* 56-57	ゲートタイム(q) */
int	GateTimeStepFirst;	/* 58-59	ゲートタイム(u) */
char	VolumeUpDown	/*5A	相対音量の増減値 */

} Wtd;

表 2-4-2 パート個別構造体

型	Name	内容
int	FlagControl	bit 1 音程 LFO スイッチ bit 2 音程エンベロープスイッチ bit 3 音量 LFO スイッチ bit 4 音量エンベロープスイッチ bit 5 パン LFO スイッチ bit 6 パンエンベロープスイッチ bit 7 次のキーでアクセント(MIDI) bit 8 音程エンベロープ上位 1bit アドレス bit 9 音量エンベロープ上位 1bit アドレス bit 10 パンエンベロープ上位 1bit アドレス bit 11 音量が 0 になった(未使用) bit 12 次のコマンドでワークを参照する bit 13 以前のワーク演算で、オーバーフローした bit 14 以前のワーク演算で、0 になった bit 15 このパートは効果音であるか? bit 16 演奏・発生終了
char	FlagTai	bit 0 タイフラグ
char	FlagSharp	bit 0,1,~6 と順にド、レ、ミに対して、半音上げる。
char	FlagFlat	bit 0,1,~6 と順にド、レ、ミに対して、半音下げる。
char	Channel	bit 0~3 チャンネル・トラック番号 bit 7 0 : MIDI / 1 : WS 内蔵 PCM
int	Address	現在演奏しているアドレス
char	LoopCount[8]	演奏・効果音 : 残りループ回数(ネストは 8 回分)
long	PcmAddress	PCM Voice : 発生中のアドレス
long	PcmSize	PCM Voice : データのサイズ
char	LoopCountPointer	ループのネスト用のスタックポインタ
char	Program	MIDI : プログラム番号
char	Voice	PCM : 音色番号
char	SweepLevel	PCM ch2 : Sweep レベル値
char	NoiseMode	PCM ch3 : ノイズモード値
char	SweepTime	PCM : Sweep タイム値
int	Leng	現在の音長
int	LengCounter	音長制御用のカウンタ
int	LengDefalut	音長省略時の音長
int	KeyOffTime	ゲートタイム制御用
int	KeyOnDelay	音量のキーオンの遅延時間

char	KeyShift	移調(トランス)	
char	Key	発生すべき KeyCode	
char	KeySet[8]	今、実際に出力した KeyCode (8 和音)	
char	KeySetPointer	和音制御用のポインタ	
char	AcsentVelocity	アクセント時の鍵盤を叩く強さ	
int	BendSet	MIDI: 出力したピッチベンド	
int	FrectionSet	PCM: 出力した周波数	
int	Bend	MIDI: 指定したピッチベンド	
int	Frection	PCM: 出力すべき周波数	
int	BendDetune	音程に対するディチューン(MIDI は、Bend を使用。)	
char	BendLfoDecayRate	音程 LFO 制御用	遅延時間
char	BendLfoSpeedRate	音程 LFO 制御用	1 回の変化に要する速度
int	BendLfoLevel	音程 LFO 制御用	1 回の変化量
char	BendLfoCount	音程 LFO 制御用	変化回数
char	BendLfoRateCounter	音程 LFO 制御用	速度制御用のカウンタ
int	BendLfoLevelSet	音程 LFO 制御用	出力した変化量
char	BendLfoCountCounter	音程 LFO 制御用	感化回数制御用のカウンタ
char	BendEmbAddress	音程エンベロープ	パターンアドレス
char	BendEmbRate	音程エンベロープ	1 回の変化に要する速度
char	BendEmbCounter	音程エンベロープ	速度制御用のカウンタ
char	BendEmbMaxMin	音程エンベロープ	変化の最大・最小値
char	BendEmbLevelSet	音程エンベロープ	出力した変化量
char	Pan	パンポット	
char	ExprPanSet	PCM: 実際に LSI に書き込んだ値	
char	Velocity	MIDI: 鍵盤をたたく強さ	
char	ExprSet	出力したエクスプレッション	
char	Expr	指定したエクスプレッション	
int	ExprDetune	エクスプレッションに対する変化量	
char	ExprLfoDecayRate	音量 LFO 制御用	遅延時間
char	ExprLfoSpeedRate	音量 LFO 制御用	1 回の変化に要する速度
int	ExprLfoLevel	音量 LFO 制御用	1 回の変化量
char	ExprLfoCount	音量 LFO 制御用	変化回数
char	ExprLfoRateCounter	音量 LFO 制御用	速度制御用のカウンタ
int	ExprLfoLevelSet	音量 LFO 制御用	出力した変化量

char	ExprLfoCountCounter	音量 LFO 制御用	感化回数制御用のカウンタ
char	ExprEmbAddress	音量エンベロープ	パターンアドレス
char	ExprEmbRate	音量エンベロープ	1 回の変化に要する速度
char	ExprEmbCounter	音量エンベロープ	速度制御用のカウンタ
char	ExprEmbMaxMin	音量エンベロープ	変化の最大・最小値
char	ExprEmbLevelSet	音量エンベロープ	出力した変化量
char	WorkAddress	ワークコマンド使用時の上位 8bit	
char	OctaveSet	出力したオクターブ	
char	Octave	指定したオクターブ	
char	GateTime8	'Q','U'コマンドで指定した数値	
int	GateTimeStepLast	'q'コマンドで指定した数値	
int	GateTimeStepFirst	'u'コマンドで指定した数値	
char	VolumeUpDown	相対音量の増減値	

3. IL ファンクション

wtdIL は、C 言語で呼ばれることを前提としたアセンブラで記述されています。アセンブラで呼ぶ場合は、引数はスタックに入れて呼んで下さい。返値は Wonder Witch 付属の C コンパイラの仕様で返されます。

返値レジスタ仕様

char 型 ax (ah=00h)

int 型 ax

long 型 dxax

int	wtdIL.Stay0;
-----	--------------

<引数>

無し

<返値>

int エラーコード

0x0000 正常終了

0x0001 heap 領域が足りない。

<処理>

ドライバを常駐する。

呼び出すプログラムは、ヒープ領域に 4096byte 以上の空き領域がある必要がある。ドライバはヒープ領域の先頭から 4096byte を制御用ワークエリアとして使用する。

注意： ・ 常駐後は、以下の機能に対してユーザはアクセスしてはならない。

サウンド関連

通信関連

HBLANK タイマー割り込み制御関連

int	wtdIL.StayOut();
-----	------------------

<引数>

無し

<返値>

int エラーコード

0x0000 正常終了

0x0001 常駐していない

<処理>

ドライバを常駐解除する。

現在は、常駐未チェックであるため、常駐せずにこの機能を呼ぶことは危険なので注意すること。

void	wtdIL.Init();
------	---------------

<引数・返値>

無し

<処理>

ドライバを初期化する。

char wtdIL.MusicPlay(char far *Music);

<引数>

char far *Music 演奏データのあるアドレス

<返値>

char エラーコード

0x00 正常終了

0x01 WTD の演奏データではない。

0x02 データのバージョンがドライバーより高い。

<処理>

演奏を開始する。

<処理内容>

演奏用構造体のセット & キーオフ & 演奏フラグのセット

void wtdIL.MusicStop();

<引数・返値>

無し

<処理>

演奏を終了する。

<処理内容>

演奏フラグをリセット & 音量を 0 にする。

char wtdIL.EffectPlay(char far *Effect)

<引数>

char far *Effect データのあるアドレス

<返値>

char エラーコード(wtdIL.MusicPlay と同じ)

<処理>

効果音を発生する。

<処理内容>

演奏用パートマスクのセット

発生用構造体のセット & キーオフ & 効果音フラグのセット

void wtdIL.EffectStop();

<引数・返値>

無し

<処理>

発生を止める。

<処理内容>

効果音フラグをリセット & 使っていたパートに対して以下の処理

演奏用パートマスクのリセット

現在出力されるべき数値を出力

char wtdIL.PcmPlay(char far *Pcm)

<引数>

char far *Pcm データのあるアドレス

<返値>

char エラーコード(0:正常終了／その他:異常終了)

<処理>

効果音を発生する。

<処理内容>

演奏用パートマスクのセット

発生用構造体のセット & キーオフ & 効果音フラグのセット

void wtdIL.PcmStop();

<引数・返値>

無し

<処理>

発生を止める。

<処理内容>

PCM フラグをリセット & 使っていたパートに対して以下の処理

演奏用パートマスクのリセット

現在出力されるべき数値を出力

```
char wtdIL.PcmVoiceSet(char no,char far *wave);
```

<引数>

int no 音色番号(0～15)

char far *wave 音色データ格納アドレス (FreyaBIOS 仕様)

<返値>

char エラーコード

<処理>

音色をドライバーのワークに格納する。エラーコードは以下の通り。

00h 正常終了

11h 番号が異常

```
char wtdIL.SoftEmbSet(char no,char far *emb);
```

<引数>

int no 音色番号(0～23)

char far *emb エンベロープデータ格納アドレス(WTD_Emb 構造体)

<返値>

char エラーコード(wtdIL.PcmVoiceSet と同じ)

<処理>

エンベロープデータをドライバーのワークに格納する。

```
char far *wtdIL.ChangeFar(void far *add);
```

<引数>

char far *add ファーアドレス

<返値>

far * ファーアドレス

<処理>

ファーアドレスのオフセットを、0000h に変換する。

void	wtdIL.SoundSetChannel(int mode);
------	----------------------------------

<引数>

int mode サウンド BIOS ah=01h と一緒

<返値>

無し

<処理>

サウンド BIOS を利用してモードを設定する。

int	wtdIL.SoundGetChannel();
-----	--------------------------

<引数>

無し

<返値>

int サウンド BIOS ah=02h と一緒

<処理>

サウンド BIOS を利用してモードを取得する。

void	wtdIL.SoundSetOutput(int mode);
------	---------------------------------

<引数>

int mode サウンド BIOS ah=03h と一緒

<返値>

無し

<処理>

サウンド BIOS を利用してモードを設定する。

int	wtdIL.SoundGetOutput(int mode);
-----	---------------------------------

<引数>

無し

<返値>

int サウンド BIOS ah=04h と一緒

<処理>

サウンド BIOS を利用してモードを取得する。

int	wtdIL.SoundGetRandom();
-----	-------------------------

<引数>

無し

<返値>

int サウンド BIOS ah=0Eh と一緒

<処理>

サウンド BIOS を利用してチャンネル 4 の疑似乱数を取得する。

void	wtdIL.SetPlayMode(int mode);
------	------------------------------

<引数>

int	mode	演奏モード
	00h	MIDI 有りモード
	その他	MIDI 無しモード

<返値>

無し

<処理>

モードをセットする

現在、ドライバ内部ではこの数値は使われていない。

int	wtdIL.GetPlayMode();
-----	----------------------

<引数>

無し

<返値>

int	モード (wrdIL.SetPlayMode と同じ)
-----	-----------------------------

<処理>

モードを所得する

現在、ドライバ内部ではこの数値は使われていない。

void wtdIL.SetMusicVolume(char volume);

<引数>

char volume 音量（0(小)～255(大)）

<返値>

無し

<処理>

メインのソフトウェア音量を設定する。

ユーザがこのファンクションで指定する数値を管理することでフィードバック処理などがおこなえる。

int wtdIL.GetMusicVolume();

<引数>

無し

<返値>

char 音量（0(小)～255(大)）

<処理>

メインのソフトウェア音量を所得する。

void wtdIL.SetEffectVolume(char volume);

<引数>

char volume 音量（0(小)～255(大)）

<返値>

無し

<処理>

効果音のソフトウェア音量を設定する。

int wtdIL.GetEffectVolume();

<引数>

無し

<返値>

char 音量（0(小)～255(大)）

<処理>

効果音のソフトウェア音量を所得する。

void	wtdIL.SetPcmVolume(char volume);
------	----------------------------------

<引数>

char volume bit 0,1 右の音量(0～3)
bit 2,3 左の音量(0～3)

<返値>

無し

<処理>

Pcm のハードウェア音量を設定する。

int	wtdIL.GetPcmVolume();
-----	-----------------------

<引数>

無し

<返値>

char bit 0,1 右の音量(0～3)
bit 2,3 左の音量(0～3)

<処理>

Pcm のハードウェア音量を取得する。

4. 曲・効果音データフォーマット

4-1. ヘッダーフォーマット

以下にヘッダーのフォーマット及び構造体定義を示す。

●構造体定義

```
typedef struct{
    char    Name[4];          /* 00-03h      :選別子 "WTD',0" */
    char    VersionN;         /* 04h         :Version 整数部 */
    char    VersionS;         /* 05h         :Version 小数部 */
    int     Extr;             /* 06-07h      :拡張ヘッダサイズ */
    char    Emb;              /* 08h         :エンベロープ数量 */
    char    Voice;            /* 09h         :波形データ数量 */
    char    Part;             /* 0Ah         :パート数 */
    char    TimeBase;         /* 0Bh         :4部音符の分割数 */
    void    *ExtrAdr;         /* 0C-0Dh      :拡張ヘッダのアドレス */
    char    *DataAdr;         /* 0E-0Fh      :データのアドレス */
    void    *(*PartAdr);      /* 10-11h      :曲データ先頭アドレスのアドレス */
} WTD_Mus;
```

表 4-1 ヘッダーフォーマット

address	型	内容	
0000h~0003h	char	'WTD',0	;データ判別用
0004h	char	データバージョン 整数部	;バージョン違いによる暴走対策
0005h	char	データバージョン 小数部	;バージョン違いによる暴走対策
0006h	int	拡張ヘッダデータ数量	;拡張ヘッダのデータ数(byte 単位)
0008h	char	ソフトエンベロープ数量	;エンベロープの定義数(17byte 単位)
0009h	char	WS 内蔵 PCM 音色数量	;音色の定義数(17byte 単位)
000Ah	char	パート数	;パート数(現在は 20)
000Bh	char	Time base	;4 分音符の分割数
000Ch	*void	拡張ヘッダ	;拡張ヘッダ格納アドレス
000Eh	*void	音色, エンベロープ 定義の先頭	;定義データ格納アドレス
0010h	*void	トラックのアドレス	Track1 ;パート数だけ定義がある。
0012h	*void	トラックのアドレス	Track2 ;パート数だけ定義がある。
...	
????h~????h		データ領域	ポインタはこの領域の任意アドレスを指す。

4－2．拡張ヘッダー

将来の拡張性のために確保した領域である。

Address	Data
0006h word	拡張ヘッダのサイズ(単位:Byte)
000Ch word	拡張ヘッダの先頭アドレス

現在は、未使用となっている。

4－3．定義データ

ドライバー本体に転送するソフトウェア・エンベロープ形状データ及び、WS 内蔵 PCM 波形データを定義する領域である。

Address	Data
0008h byte	定義しているエンベロープの数量
0009h byte	定義している WS 内蔵 PCM 音色の波形の数量
000Eh word	上記定義データの存在しているアドレス

音色、エンベロープ共に 17Byte 1 組みの構造体であり、先頭 1 Byte は音色若しくはエンベロープの番号であり、残りの 16Byte は音色の波形データ、又は、エンベロープ形状となる。番号は、MML 中に記述する番号と同一にしなければならない。

各定義の為の構造体は、「WS 内蔵 PCM 音色波形データ」、「エンベロープ形状データ」の順に格納しなければならない。

備考：

音色についてのデータの書式は本書 2－2 項を参照。

エンベロープについてのデータの書式は本書 2－3 項を参照。

4-4. 演奏データ

演奏すべきデータを格納する領域である。

Address	Data
000Ah byte	演奏するトラック数
000Bh byte	4分音符の分割数
0010h word	Track 1 の先頭アドレス
0012h word	Track 2 の先頭アドレス
.....	Track の数量だけ、アドレスを順に記述する。

以下に演奏データのコマンドを説明する。

4-4-1. 音階データ書式

以下に、音階データの書式を示す。

表 4-4-1 音階データ書式について

bit	Data	Category
bit0-2	0	r 休符
	1	c ド
	2	d レ
	3	e ミ
	4	f ファ
	5	g ソ
	6	a ラ
	7	b シ
bit3-4	0	何も無し
	1	# シャープ
	2	b フラット
	3	ナチュラル
bit5		& 次、キーオフをしない。
bit6		1 次に来る数値は音長(byte or word)
bit7	1 固定	

bit 6 = 'H'(音長指定有り)の引数について

ドライバー内部では、まず 1[byte]のみデータ取得を行う。このとき、データが 000h(0x00) ~ 0Feh(0xFE)の場合は、この数値が音長となる。取得したデータが、0FFh(0xFF)であった場合は、次の 2[byte]のデータ(インテル方式)を読み込み、この 16[bit]の数値を音長とする。

例 080h 音長 = 1 2 8
 0FFh,000h,001h 音長 = 2 5 6
これは、'I'(0x6C,06Ch)コマンドにも相当する。

4-4-2. 演奏コマンド

Command	Version	Information
0x21 !	Ver,0,01	次のコマンドの引数をワーク参照にする。このとき、コマンドの引数はワークアドレスの下位 8bit として使われる。
0x22 "	Ver,0,01	ワークアドレスの上位 8bit を指定する。このとき、0x00 以外の値を指定した場合の動作は保証しない。
<i>byte</i>		ワークアドレス上位 8bit
0x27 '	Ver,0,01	次の音符のみに対してベロシティーを指定する。
<i>byte</i>		ベロシティー
0x28 (Ver,0,01	エクスプレッションまたは音量を減衰させる。差分は、0x78 コマンドにて指定する。
0x29)	Ver,0,01	エクスプレッションまたは音量を増加させる。差分は、0x78 コマンドにて指定する。
0x2A *	Ver,0,04	LFO の On/Off を設定する。
<i>byte</i>		0: 音程 / 1: 音量 / 2: パンポット
<i>byte</i>		0...Off / 1...On
0x2F /	Ver,0,05	エンベロープの On/Off を設定する。
<i>byte</i>		0: 音程 / 1: 音量 / 2: パンポット
<i>byte</i>		0...Off / 1...On

0x30	0	未製作	ワークに数値を代入する。
	byte		ワークアドレス
	byte		代入する数値
0x31	1	未製作	ワークに数値を加算する。
	byte		ワークアドレス
	byte		加算する数値
0x32	2	未製作	ワークに数値を減算する。
	byte		ワークアドレス
	byte		減算する数値
0x33	3	未製作	ワークに数値を論理積する。
	byte		ワークアドレス
	byte		論理積する数値
0x34	4	未製作	ワークに数値を論理和する。
	byte		ワークアドレス
	byte		論理和する数値
0x35	5	未製作	ワークに数値を排他的論理和する。
	byte		ワークアドレス
	byte		排他的論理和する数値
0x36	6	未製作	ワークをビットセットする。
	byte		ワークアドレス
	byte		ビットセットするビット
0x37	7	未製作	ワークにビットリセットする。
	byte		ワークアドレス
	byte		ビットリセットするビット
0x38	8	未製作	ワークと数値を比較する。
	byte		ワークアドレス
	byte		比較する数値
0x39	9	未製作	ワークにビットテストする。
	byte		ワークアドレス
	byte		ビットテストする数値

0x3A	:	Ver,0.03	最後にループを抜ける。
		<i>word</i>	<i>0x5D</i> コマンドのアドレス。
0x3B	;	未製作	WTD_Ctrl_Flag の下位 8bit と条件を比較し一致した場合、ループを抜ける。
		<i>byte</i>	条件
		<i>word</i>	<i>0x5D</i> コマンドのアドレス。
0x3C	<	Ver,0.01	オクターブを下げる
0x3E	>	Ver,0.01	オクターブを上げる。
0x40	@	Ver,0.01	音色を指定する。
		<i>byte</i>	音色番号(PCM 0~15 / MIDI 0~127)
0x42	B	未製作	ベンド及びベンドレンジを設定する。
		<i>word</i>	<i>bit 15</i> 0:Bend Range を省略する。 / 1:設定する。
			<i>bit 0-13</i> Bend (0(低)~8192(中央)~16383(高))
		(<i>byte</i>)	Bend Range(前引数 <i>bit15</i> ='1'のときのみ指定する。)
0x43	C	Channel/Track	チャンネルおよびトラックを設定する。
		<i>byte</i>	<i>bit 0-3</i> チャンネル／トラック
			<i>bit 4-7</i> 0000< <i>b</i> > : 外部 MIDI 音源 Port 1
			~ ~ ~ (未対応)
			0111< <i>b</i> > : 外部 MIDI 音源 Port 8 (未対応)
			1000< <i>b</i> > : 内蔵 PCM 音源
0x44	D	Ver,0.01	周波数に対して変化量を指定する。
		<i>word</i>	変化量(-16384~16383)
0x45	E	エンベロープ	エンベロープを指定する。
		<i>byte</i>	0 : 音程 / 1 : 音量 / 2 : パンポット
		<i>byte</i>	エンベロープ番号(0~23)
0x46	F	未製作	音量を設定する。
		<i>byte</i>	音量(0~127)
0x47	G	未製作	チャンネルプレッシャーを設定する。
		<i>byte</i>	プレッシャー(0~127)
0x48	H	未製作	バンクセレクトを行う。
		<i>byte</i>	Bank MSB
		<i>byte</i>	Bank LSB
0x4B	K	Ver,0.01	キーオンの遅延時間を設定する。
		<i>word</i>	遅延時間 (0~65535)
0x4C	L	Ver,0.01	演奏終了、又は無限ループ。
		<i>word</i>	0000< <i>h</i> > : 演奏終了
			Other : ループ先のアドレス

0x4D	M	未製作	モジュレーションを指定する。
			モジュレーション
0x4E	N	未製作	ノン・レジスタード・パラメータの設定
	byte		NRPN MSB
	byte		NPRN LSB
	byte		NPRN Data
0x50	P	Ver,0.03	MIDI：ダンパースイッチ / PCM：チャンネルモード
	byte	bit 1	スイッチ
0x51	Q	Ver,0.01	ゲートタイム / 8 分割
	byte		ゲートタイム(1~8)
0x54	T	未製作	ローパスフィルターの設定
	byte		カット・オフ周波数
	byte		レゾナンス
0x55	U	Ver,0.01	ゲートタイム / 100 分割
	byte		ゲートタイム(1~100)
0x56	V	Ver,0.01	音量に対する変化量を指定する。
	word		変化量(-16384~16383)
0x58	X	Ver,0.05	エクスクルーシブの送信
	byte		データサイズ
	byte[size]		エクスクルーシブ・データ (サイズ分) (0xF0,0xF7 は不要)
0x5A	Z	Ver,0.05	バイナリーデータの送信
	byte		データサイズ
	byte[size]		送信するデータ(サイズ分)
0x5B	[Ver,0.03	ループ開始位置
	byte		ループ回数
0x5D]	Ver,0.03	ループの終了位置
	word		0x5B コマンドのアドレス
0x5F	_	Ver,0.01	音程のシフト量を設定する。
	byte		シフト量
0x6B	k	Ver,0.01	KeyOn 時のベロシティーを指定する。
	byte		ベロシティー
0x6C	l	Ver,0.01	音長省略時の音長を指定する。
	byte	0x00	音長=0[step] 和音発生時に使用する。
		0x01~0xFE	音長
		0xFF	2Byte で音長を指定する。
	(word)		音長 (前引数が、0xFF の時のみ指定する。)

0x6D	m	Ver,0.04	音程の揺らぎを設定する。
	byte		0: 音程 / 1: 音量 / 2: パンポット
	byte		Decay Rate
	byte		Speed
	word		Level
	byte		Count
0x6E	n	Ver,0.01	ノイズモードを指定する。
	byte		ノイズモード(0~7)
0x6F	o	Ver,0.01	オクターブを指定する。
	byte		オクターブ(-2~9)
0x70	p	Ver,0.01	パンポットを指定する。
	byte		パンポット (0(右)~64(中央)~127(左))
0x71	q	Ver,0.01	ゲートタイム 「音長一指定した数値」 [step]発音する。
	word		ゲートタイム(0~65535)
0x73	s	Sweep	スイープモードを設定する。
	byte		Level
	byte		Rate
0x74	t	Ver,0.01	テンポを設定する。
	word		割り込み周期 (H-BLANK 12[kHz])
0x75	u	Ver,0.01	ゲートタイム 「指定した数値」 [step]発音する。
	word		ゲートタイム(0(半永久),1~65535)
0x76	v	Ver,0.01	エクスプレッション/PCM 音量を指定する。
	byte		エクスプレッション/音量 (0~127)
0x78	x	Ver,0.05	相対音量の差分値を指定する。
	byte		差分値
0x79	y	Ver,0.05	コントロールチェンジをする。
	byte		レジスタ
	byte		データ
0x7B	{	未製作	臨時記号(半音)の初期値を設定する。
	byte	bit 0	ド
		bit 1	レ
		~	~ ~ ~
		bit 6	シ
		bit 7	0: Sharp set / 1: Flat set

5. PCM Voice データフォーマット

Microsoft 社の'wav'形式のファイルを採用している。対応しているフォーマットは以下の仕様のファイルのみであるので留意すること。

- ・量子化数 8 [bit]
- ・チャンネル数 1 [ch] (モノラル)

サンプリングレイトは如何なるファイルも 12 [kHz]として再生する。

本ドライバーでは、'fmt'チャンク及び'data'チャンクがファイルの先頭から 64 [kByte]以内に存在していなければならない。データチャンクのサイズは、論理的には 4 [Gbyte]まで対応しているが、80186CPU 及び、FreyaOS, BIOS がそれを許さない。ファイルサイズは、384[kByte]の壁は越えられない。

PCM Voice の制御は、12[kHz]で H-BLANK(タイマー)割り込みをかけることにより 12[kHz]のサンプリングレイトを実現している。このとき、音楽の演奏処理および効果音処理は、割り込み処理内部でタイマーをエミュレーションすることでテンポ処理を行うため、割り込み処理が非常に重たくなる。メインルーチンは、このことを留意した上でアプリケーションを開発しなければならない。

6. 音色ファイルフォーマット

6-1. ヘッダーフォーマット

現在、考案中。

6-2. データフォーマット

現在、考案中。

7. 技術サポートについて

Wonder Swan Total Sound Driver (WTD) についての技術サポートは、FSPNET のみ受け付けております。ドライバーの制御方法、内部変数の詳細、データ形式に関する技術的な質問は FSPNET でしかお答えいたしませんので、ユーザサポート BBS 等への技術的な質問はご遠慮ください。

FSPNET の詳細につきましては、アーカイブ付属の'FSPNET.txt'をご参照ください。