

お使いになる前に必ずお読み下さい

ソフトウェアの使用条件と保障

著作権

本ソフトウェアは著作権法により保護されています。有限会社イエローソフトが本ソフトウェアの著作権を保有します。

したがって、本ソフトウェアを弊社に無断で、売却、譲渡、賃貸またはその他いかなる方法であっても第三者に使用させることはできません。

使用の制限

お客様は、本ソフトウェアを1台のコンピュータに限ってインストールすることができます。また、バックアップを目的とする複製のみ許可します。

保障

弊社は、本ソフトウェアのディスクおよびマニュアルに物理的欠陥がある場合、ご購入日から30日以内であれば無料で交換いたします。

免責

弊社は、前項に定める場合を除き、本ソフトウェアの使用、あるいは使用結果に対していかなる責任も負いません。

弊社は、本ソフトウェアによって生成される2次的ソフトウェアに関して、いかなる制限も与えません。またいかなる保障もいたしません。

もくじ

第1章	はじめに-----	3
第2章	USB メモリ IO モジュールと CPU ボードとの接続-----	10
第3章	デバイスドライバの移植と動作テスト-----	17
第4章	YS-FILE の構築とテストプログラムの実行-----	32
第5章	YS-FILE のライブラリ化-----	40
第6章	アプリケーションプログラムの作成-----	42
第7章	YS-FILE のコンフィグレーション-----	48

第1章 はじめに

1 概要

USB メモリ開発セットは、USB メモリ I/O モジュール、デバイスドライバ、ファイルシステムからなり、イエローソフト製の C コンパイラを使って簡単に USB メモリアプリケーションを組むことができるセットです。

2 内容物

USB メモリ I/O モジュール (コネクタ付き)

デバイスドライバ

ファイルシステム YS-FILE

YS-FILE ユーザーズマニュアル

このマニュアル

デバイスドライバとファイルシステムの YS-FILE は CD-ROM に格納されています。

3 必要なシステム

ソフトウェア

イエローソフトの C コンパイラ YCH8 または YCSH が必要です。

YellowIDE Ver.6 以上

ハードウェア

条件 1 H8/300H、H8S、SH1、SH2 シリーズの CPU を搭載した CPU ボードで外部にバス接続ができるもの。Tiny シリーズなどは外部にバス接続できないため使用できません。

条件 2 RAM など他のデバイスと外部バスを共有する場合は、CPU がバスコントローラを内蔵しておりかつ、複数のメモリ空間に分かれている必要があります。

一つのメモリ空間を RAM などのデバイスと USB メモリ IO モジュールで分けて使うことはできません。これはアイドルサイクルの挿入ができないため、デバイス間で干渉を起こす可能性があるためです。弊社の CPU ボードでこれに該当するのは H8S2144 です。

16K バイトの RAM が必要です。

4 動作電圧

5Vのみ

5 ファイルシステム YS-FILE の特徴

イエローソフトのファイルシステムである YS-FILE には以下の特徴があります。

Windows 互換 FAT12/FAT16/FAT32 をサポート
階層ディレクトリ、複数ドライブ、相対パス対応
ANSI 準拠のファイル入出力関数をサポート
バッファサイズなどコンフィギュレーション可能
電源断によるファイルシステムの修復機能
サンプルとして簡単なディスク OS [miniDOS]が付属
ソースコード付属
ライセンスフリー

電源断によるファイルシステムの修復機能は、ファイルシステム(FAT)を修復するのであって、アクセス途中のデータを修復するものではありません。FAT 領域の破壊によってすべてのファイルがアクセスできなくなってしまうことを避けます。

以下の機能はサポートしていませんので使用には注意して下さい。

NTFS

ロングファイル名(VFAT)

ただし、ロングファイル名を含むディスクであっても問題なくアクセスできます。この場合、ロングファイル名はショートファイル名として認識されます。

フォーマット機能

フォーマット機能はありません。フォーマットはパソコンで行って下さい。

リアルタイム OS への対応

リアルタイム OS へは特に対応していません。したがってリアルタイムシステムで使う場合は、ファイルシステムをコールするタスクを一つに統一する、あるいは排他制御を行うなどする必要があります。

FAT フォーマットの制限

FAT12/FAT16/FAT32 を使用できますが、使用できるフォーマット形式に一部制限があります。制限は以下の通りですが、通常のディスクを Windows でフォーマットした場合は問題ありません。

- 1 セクタサイズは 512 バイト
- 2 FAT 領域が 2 つあること

6 ファイルシステムの大きさ

下記はあくまでも目安です。ファイルシステムのバージョン、コンパイルオプションなどによって変わる場合があります。

最小構成 ANSI 関数を使いファイルコピープログラムを作った場合。FTA16 だけ有効、セーフシステム OFF の設定、バッファ最小。

(使用した関数、yfopen,yfclose,yfread,yfwrite)

	H8	SH
ROM	約 44K バイト	約 36K バイト
RAM	約 9K バイト	約 9K バイト

上記プログラムをシステムコールだけで作った場合

(使用した関数、yfOpen,yfCreate,yfClose,yfRead,yfWrite)

	H8	SH
ROM	約 37K バイト	約 30K バイト
RAM	約 8K バイト	約 8K バイト

サンプルプログラムの miniDOS のプログラムサイズ (多くの ANSI 関数を使用した場合、すべての FAT を有効、セーフシステム ON の設定、バッファ最小)

	H8	SH
ROM	約 90K バイト	約 74K バイト
RAM	約 11K バイト	約 12K バイト

7 インストール

CD-ROM を CD-ROM ドライブに挿入して下さい。自動的にインストール画面が開きます。インストール画面が開かない場合は「マイコンピュータ」の中の CD-ROM ドライブのアイコンをクリックして下さい。

CD-ROM の中には YS-FILE の他、最新版の YellowIDE6 も含まれています。バージョンが古い場合は最新版の YellowIDE6 もインストールします。

YS-FILE(USB メモリ版)のインストール

YS-FILE(USB メモリ版)のボタンをクリックするとインストールが開始されます。

YS-FILE は必ず、YellowIDE6 をインストールしたフォルダにインストールして下さい。デフォルトでは Windows のあるドライブの¥YellowIDE6 というフォルダにインストールされます。YellowIDE6 がこれ以外のフォルダにある場合は、インストールの途中でフォルダを変更します。



上図の画面で（変更）のボタンをクリックすることによってインストール先のフォルダを変更できます。必ず、YellowIDE6 をインストールしたフォルダにインストールして下さい。

YS-FILE(SD カード版)のインストール

SD カード IO モジュールを同時に購入された場合や将来購入予定の場合は、YS-FILE の SD 版をインストールして構いません。USB 版と SD 版では違うフォルダにインストールされます。

USB 版 YS-FILE-USB というフォルダにすべてがインストールされます。

SD 版 YS-FILE というフォルダにすべてがインストールされます。

重要！！

既に SD 版の YS-FILE がインストールされている場合

以前に SD カード開発セットを購入して、既に SD 版の YS-FILE がインストールされている場合は、この CD-ROM から SD 版の YS-FILE をインストールすると古いファイルと新しいファイルが混在してしまいます。古い YS-FILE のフォルダをどこかに保存した後、削除してインストールするか、違う名前のフォルダ名に変えてインストールして下さい。

インストール後のフォルダ構造

インストールが完了すると YellowIDE のフォルダの中に YS-FILE のフォルダが以下のような構造でできます。

¥YellowIDE6

-----YS-FILE-USB

-----Device-USB USB メモリデバイスドライバプロジェクト

-----SRC デバイスドライバソース

-----H83048 H8/3048 用プロジェクト

-----H83064 H8/3064 用プロジェクト

-----H83069 H8/3069 用プロジェクト

-----H8S2238 H8S2238 用プロジェクト

-----H8S2367 H8S2367 用プロジェクト

-----H8S2633 H8S2633 用プロジェクト

-----SH7044 SH7044 用プロジェクト

-----SH7045 SH7045 用プロジェクト

-----SH7047 SH7047 用プロジェクト

-----SH7050 SH7050 用プロジェクト

-----System

-----prj YS-FILE&miniDOS プロジェクト

-----Sample

-----Sample1 サンプルプログラム

8 開発の手順

次のような流れで開発を進めます。

第2章 USB メモリ IO モジュールと CPU ボードとの接続

第3章 デバイスドライバの移植とテストプログラムの実行

第4章 YS-FILE の構築とテストプログラムの実行

第5章 YS-FILE のライブラリ化

第6章 アプリケーションプログラムの作成

9 注意事項（必ず読んで下さい）

アプリケーションプログラムのデバッグが終了して開発が完了するまでは USB メモリは中のデータが破壊されても構わない USB メモリを使用して下さい。

また、アプリケーションプログラム運用中も大切なデータは必ず定期的にバックアップしておくようにして下さい。弊社はデータの損失による責任を一切負いません。

USB ハブや USB メモリカードリーダーライターには対応していません。USB メモリを USB コネクタに直接挿入して使用して下さい。

第2章 USB メモリ IO モジュールと CPU ボードとの接続

1 電源電圧およびインターフェース電圧

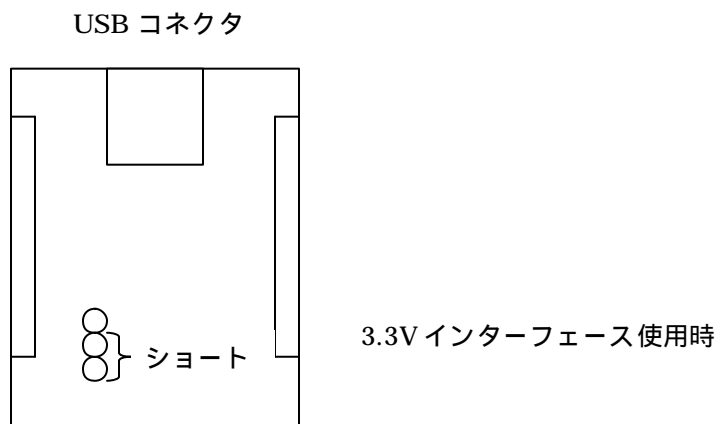
電源電圧は 5V のみです。

CPU とのインターフェース電圧は 5V と 3.3V のどちらかを選択できます。出荷時の設定は 5V になっています。

3.3V インターフェースにするにはジャンパの JP1 において、真ん中のピンと下側のピンをショートさせます。

インターフェース電圧が 3.3V であっても USB メモリ IO モジュールに供給する電圧は必ず 5V にして下さい。

イエローソフトの CPU ボードでは 3.3V インターフェースで動作させることはできません。



2 CPU との接続

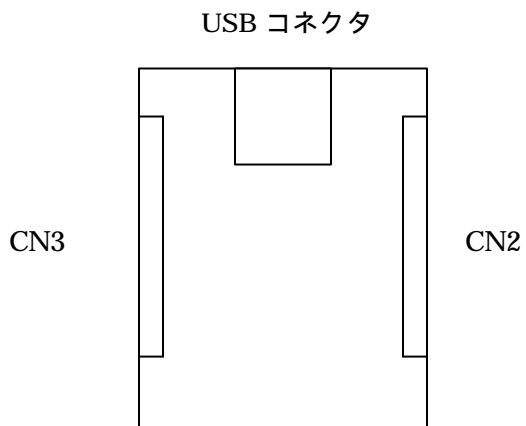
USB メモリ IO モジュールはデータ線(D0~D7)、アドレス線(A0)、リード信号線(RD)、ライト信号線(WR)、チップセレクト信号線(CS)を備えおり、ほぼ SRAM と同様な接続方法で CPU とバス接続できます。

回路例を参考に接続して下さい。

3 ENA 端子

USB メモリ IO モジュールの ENA 端子に CPU の出力ポートをつなげて下さい。出力ポートなら何でもかまいません。この端子が LOW のとき USB メモリ IO モジュールが動作します。

4 USB メモリ IO モジュールのピン配置



CN2(USB コネクタを上側に向けたとき右側のコネクタ)

ピン	信号名	説明
1	GND	グラウンド
2	GND	グラウンド
3	_RES	リセット(L=リセット)
4	_ENA	イネーブル(L=動作)
5	A0	アドレス 0
6	N.C	無接続
7	N.C	無接続
8	N.C	無接続
9	N.C	無接続
10	_CS	チップセレクト
11	_RD	リード
12	_WR	ライト
13	N.C.	無接続
14	N.C.	無接続

CN3(USB コネクタを上側に向けたとき左側のコネクタ)

ピン	信号名	説明
1	D0	データ 0
2	D1	データ 1
3	D2	データ 2
4	D3	データ 3
5	D4	データ 4
6	D5	データ 5
7	D6	データ 6
8	D7	データ 7
9	N.C.	無接続
10	N.C.	無接続(割り込み端子として予約)
11	VCC	電源(5V のみ)
12	VCC	電源(5V のみ)
13	GND	グラウンド
14	GND	グラウンド

5 回路例

注意

電源部分は省略しています。USB メモリ IO モジュールの電源端子はすべて接続して下さい。

CPU の_CS2 は一例です。空いているチップセレクト端子なら何でもかまいません。

USB メモリ IO モジュールの_RES は、CPU ボードのリセット出力端子に接続します。例えばイエローソフトの CPU ボードの場合は_RES591 と表示されている端子です。

はプルアップです。数 K ~ 10K で必ずプルアップして下さい。

H8 系の場合は D8 ~ D15 を使用しますので注意して下さい。

配線はできるだけ短くして下さい。電源ラインはなるべく太くして下さい。

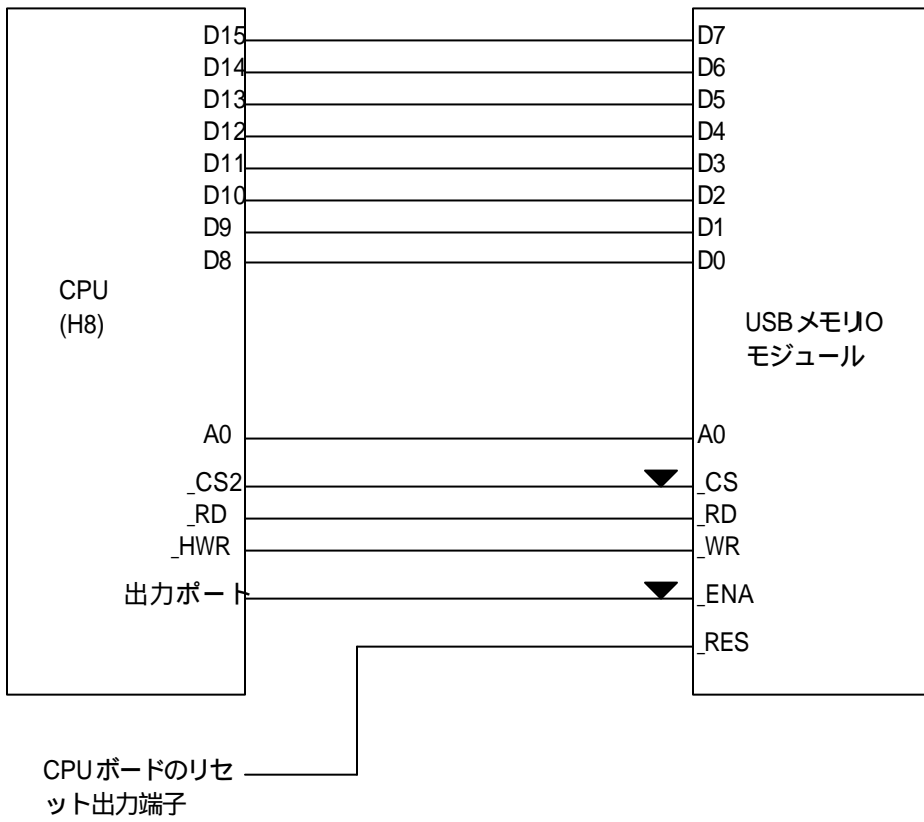
また GND 線は、CPU ボードの 1 カ所だけからとるのではなく、2 カ所、3 カ所からとると良いです。

6 USB メモリと SD カードの両方を使用できる回路にするには

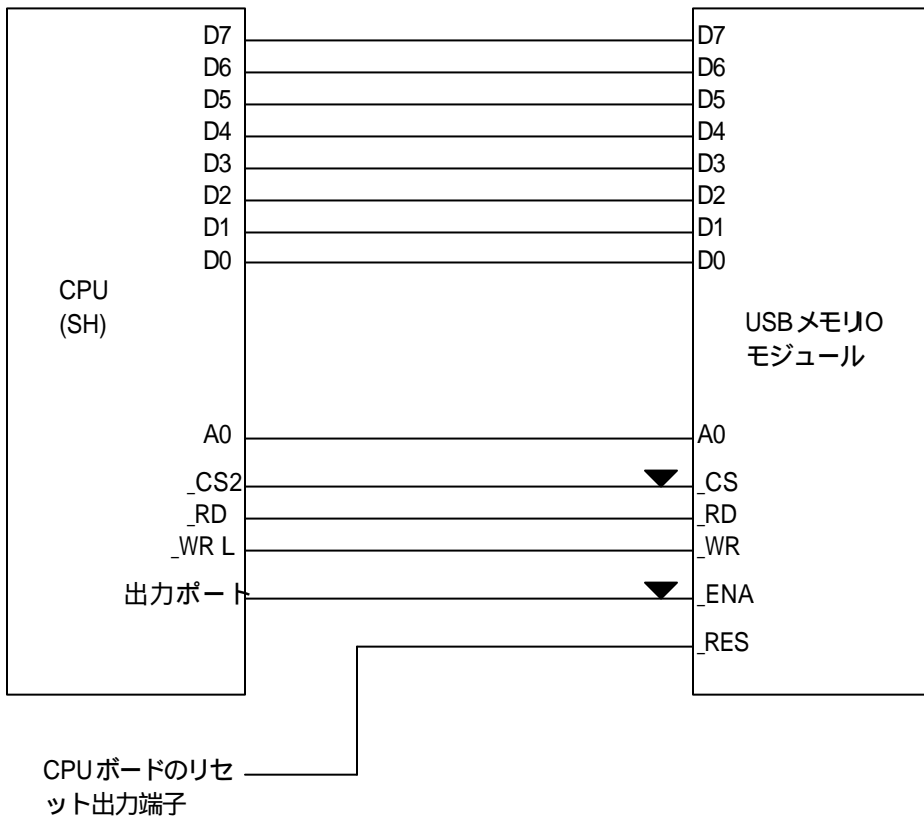
USB メモリと SD カードの違いは、アドレス線の本数です。SD カードの方がアドレス線が多いです。(A4 まで) また SD カードの場合はバスバッファを挿入します。

したがって、SD カード用の配線にしておけば、SD カードも USB メモリも両方使用できるようになります。SD カード用の配線方法は SD カード開発セットのマニュアルをご覧ください。

H8 系



SH 系



配線の確認

配線が済んだら間違いがないか確認して、電源を投入します。最低限以下の事を確認して下さい。

USB メモリ IO モジュールの電源端子をテストで調べて正常か確認する。

Vcc CN3-11 CN3-12

GND CN2-1 CN2-2 CN3-13 CN3-14

USB メモリ IO モジュールの以下の端子電圧が Vcc に等しいか確認する。

_RES CN2-3

_ENA CN2-4

_CS CN2-10

確認が済んだら、次にデバイスドライバの移植と動作テストに進みます。

第3章 デバイスドライバの移植と動作テスト

デバイスドライバを CPU ボードに合わせて移植するためにはユーザは以下のことをしなければなりません。

バスコントローラの初期設定

約 10m 秒で起動するインターバルタイマ割込みの設定

_ENA 端子の制御プログラム

イエローソフトの CPU ボードの場合はサンプルプログラムがあります。他の CPU ボードの場合はサンプルの中から一番近いと思われるものを選んで修正して使って下さい。

1 デバイスドライバのサンプルプロジェクトを開く

YellowIDE6 を起動して、(ファイル) (プロジェクトを開く) でプロジェクトを開きます。

サンプルプロジェクトは¥YellowIDE6¥YS-FILE-USB¥DEVICE-USB の中に各 CPU ごとにあります。該当する CPU がない場合は一番近いと思われるものを選んで下さい。

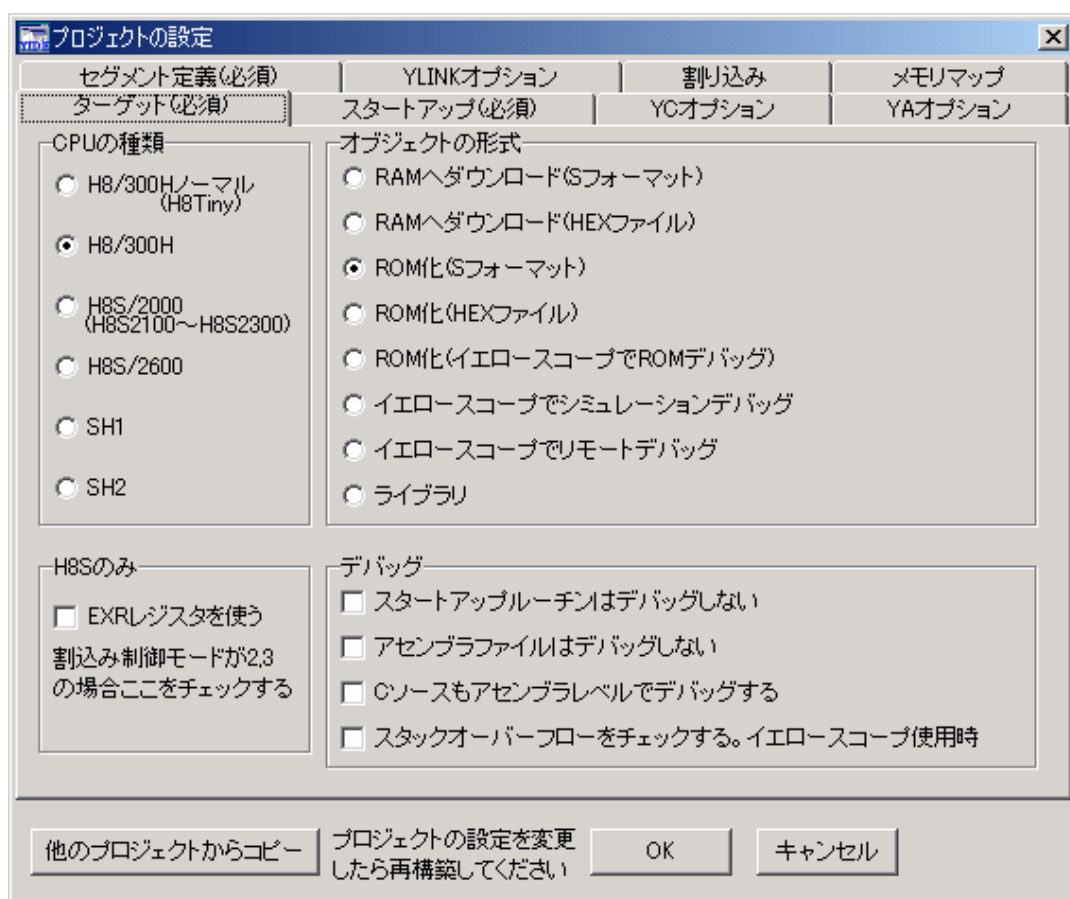
プロジェクトを開いたら、プロジェクトウィンドウを見て下さい。プロジェクトに必要なファイルが表示されています。(最後の usbdrv.h は表示されない場合もありますが影響はありません)



このうち、変更が必要なファイルは DEVICE.C と UsbConfig.h だけです。他のファイルは変更しないで下さい。

2 プロジェクトの変更

開いたプロジェクトは一例にしかすぎません。必要に応じてプロジェクトを変更する必要があります。プロジェクトウインドウの（設定）ボタンをクリックしてダイアログを開きます。



CPUの種類を必要に応じて変更して下さい。

プロジェクトの設定

セグメント定義(必須)	YLINKオプション	割り込み	メモリマップ
ターゲット(必須)	スタートアップ(必須)	YCオプション	YAオプション

スタートアップルーチン

\$.\$.*.STARTUP*CS3048.ASM

メモリ関係(すべて10進で入力して下さい)

スタックサイズ
4096 バイト

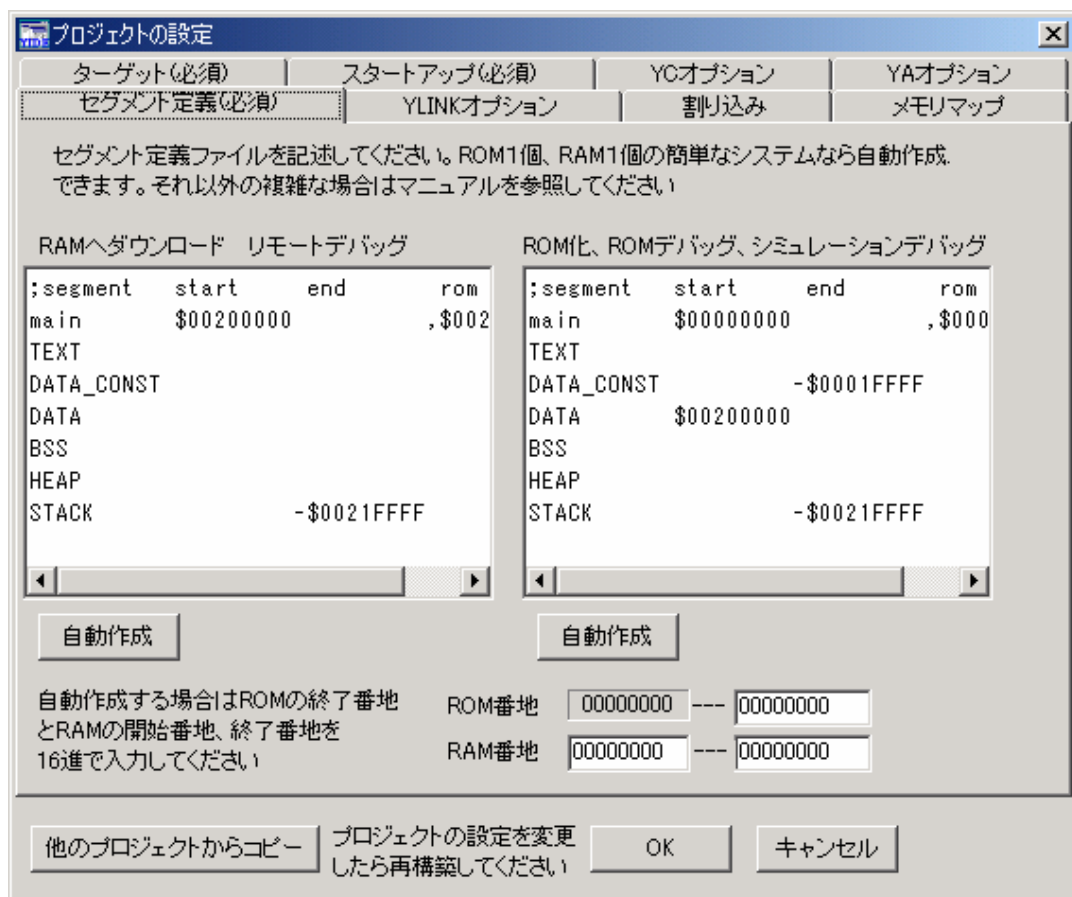
ヒープサイズ(使用しない場合は0)
0 バイト

関数の戻り値となる構造体のサイズ
(構造体を戻り値とする関数がない場合は0)
0 バイト

リンク
 スタートアップルーチンをリンクしない

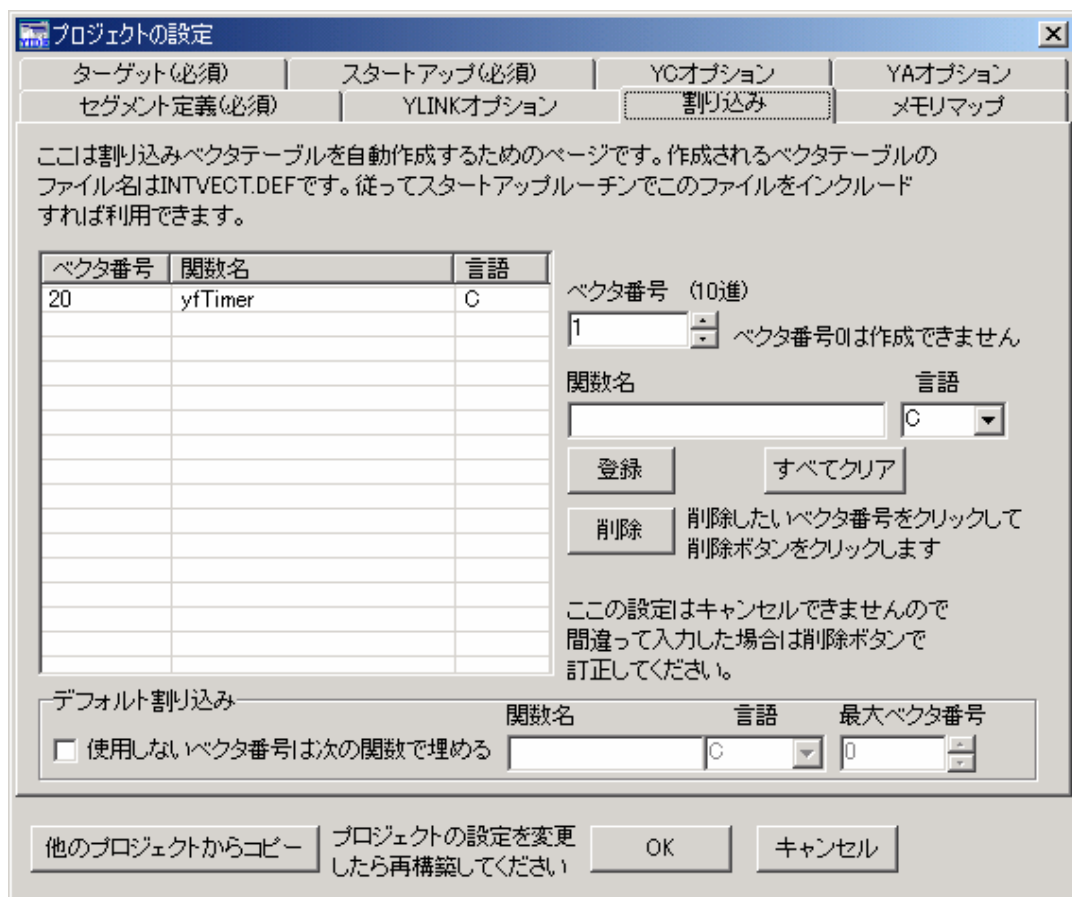
登録をクリックすると現在の設定が保存されます
読み込みをクリックすると保存した設定が読み込まれます

スタートアップルーチンを必要に応じて変更して下さい。
スタックサイズは変更しないで下さい。



セグメント定義を必要に応じて変更して下さい。

RAMのアドレスが違う場合や内蔵RAMを使う場合はRAM番地を変更する必要があります。なお、RAMは最低限16Kバイト必要です。



デバイスドライバは、約 10ms 間隔のタイマ割り込みを必要とします。タイマは CPU が内蔵している 8 ビットタイマや 16 ビットタイマ、ウォッチドッグタイマなど 10ms 間隔で割り込みが起動できるものなら何でもかまいません。またチャンネルもどれでも構いません。

使用するタイマの割り込みベクタ番号をハードウェアマニュアルで調べて変更します。関数名(yfTimer)は変更しないで下さい。

変更するには、ベクタ番号をクリックしていったん削除してから再度登録し直します。

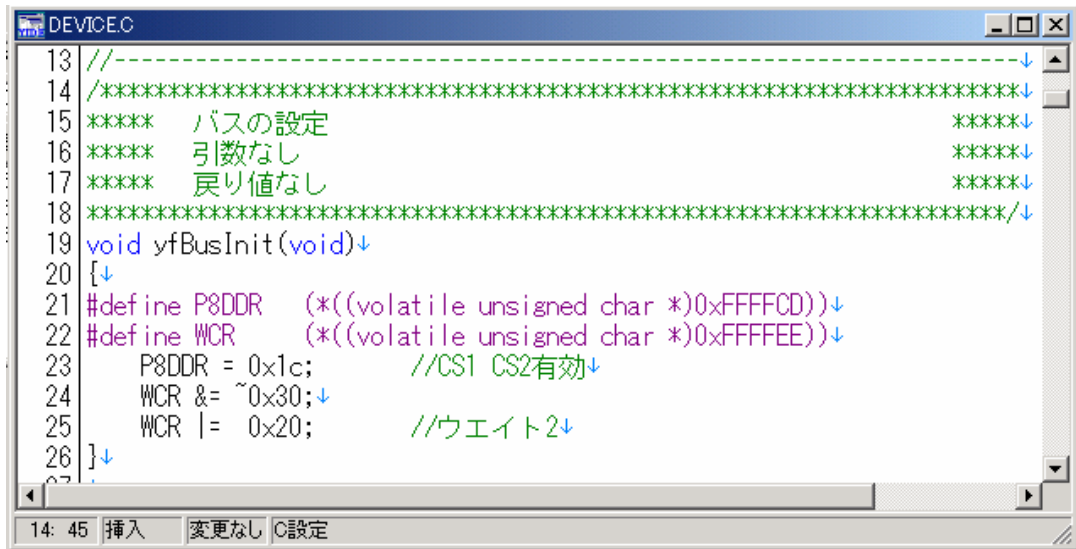
どのタイマを使用するかまだ決めていない場合は、後で変更しても構いません。以上でプロジェクトの変更は終わりです。次にソースファイルを変更します。

3 DEVICE.C の変更

プロジェクトウィンドウの DEVICE.C をダブルクリックして DEVICE.C を開きます。

最初に USB メモリ IO モジュールが接続されている外部バスの初期設定をします。関数 yfBusInit を変更します。

H8 の例



```
13 //-----↓
14 /*****↓
15 *****/ バスの設定 *****/↓
16 *****/ 引数なし *****/↓
17 *****/ 戻り値なし *****/↓
18 *****/↓
19 void yfBusInit(void)↓
20 {↓
21 #define P8DDR (*(volatile unsigned char *)0xFFFFCD)↓
22 #define WCR (*(volatile unsigned char *)0xFFFFEE)↓
23     P8DDR = 0x1c; //CS1 CS2有効↓
24     WCR &= ~0x30;↓
25     WCR |= 0x20; //ウエイト2↓
26 }↓
```

これは H8 の代表的な H8/3048F の例です。P8DDR=0x1c で CS1 端子(RAM 接続)と CS2 端子(USB メモリ IO モジュール接続)を有効にします。

P8DDR のようなポート入出力レジスタは書き込み専用レジスタです。したがって、|=や&=のような演算代入はできません。必ず単純代入文で設定して下さい。

そのため他のビットにも影響してしまいますので、他に RAM などを接続している CS 端子がある場合はそれらすべてを一度に有効にするような値を設定します。

また、この例ではポートをアドレス線として機能させる設定が記述されていません。というのは、この設定はスタートアップルーチンでされているからです。

もし、スタートアップルーチンでポートをアドレス線として機能させる設定がされていない場合はその記述も必要です。以下に例を示します。

```

P1DDR=0xFF;          //ポートをアドレスバスとして使うため
P2DDR=0xFF;          //P1,P2,P5 を出力設定する
P5DDR=0xFF;

```

ウェイトの設定

ウェイトの設定ですが、ウェイトはアクセスタイム 200ns で計算して下さい。例えば 20MHz の場合は 1 クロック=50ns ですから、200ns/50ns=4 サイクル必要です。ノーウェイトで 3 サイクルですから、1 サイクルのウェイトが必要です。

同様に 25MHz の場合は 2 ウェイト必要です。

なお、最初はウェイト最大（デフォルト）で試して下さい。動作確認後にウェイトの設定をすると良いでしょう。

SH の例

```

13 //-----↓
14 /***** ↓
15 *****/ バスの設定 *****/ ↓
16 *****/ 引数なし *****/ ↓
17 *****/ 戻り値なし *****/ ↓
18 *****/ ↓
19 void yfBusInit(void)↓
20 {↓
21 #define PACRL2 (*(volatile unsigned short *)0xFFFF838e)↓
22 #define BCR1 (*(volatile unsigned short *)0xFFFF8620)↓
23 #define WCR1 (*(volatile unsigned short *)0xFFFF8624)↓
24 //CS2を有効にする命令↓
25 PACRL2 |= 0x2000;↓
26 PACRL2 &= ~0x1000;↓
27 BCR1 &= ~0x0004; //8ビットバス↓
28 WCR1 &= ~0x0f00;↓
29 WCR1 |= 0x0300;↓
30 }↓
31 ↓

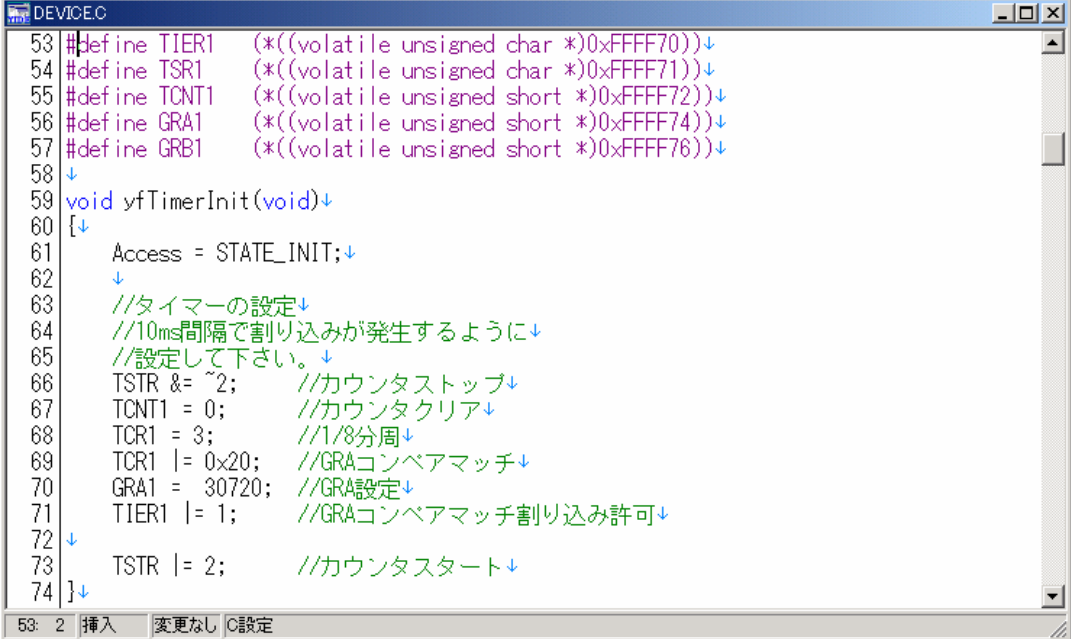
```

SH の場合も H8 と考え方は同じです。H8 の説明を読んで下さい。ただし、SH の場合は CS 端子の設定で演算代入ができます。スタートアップルーチンで RAM などの CS 端子の設定がしてある場合は USB メモリ IO モジュールに接続された CS 端子の設定だけすれば十分です。

また、SH の場合はバス幅がデフォルトで 16 ビットですからバス幅を 8 ビットにする設定を記述します。

約 10m 秒間隔で起動するインターバルタイマ割り込みの設定

DEVICE.C の関数 yfTimerInit を変更します。



```
53 #define TIER1 (*(volatile unsigned char *)0xFFFF70)↓
54 #define TSR1 (*(volatile unsigned char *)0xFFFF71)↓
55 #define TCNT1 (*(volatile unsigned short *)0xFFFF72)↓
56 #define GRA1 (*(volatile unsigned short *)0xFFFF74)↓
57 #define GRB1 (*(volatile unsigned short *)0xFFFF76)↓
58 ↓
59 void yfTimerInit(void)↓
60 {↓
61     Access = STATE_INIT;↓
62     ↓
63     //タイマーの設定↓
64     //10ms間隔で割り込みが発生するように↓
65     //設定して下さい。↓
66     TSTR &= ~2; //カウンタストップ↓
67     TCNT1 = 0; //カウンタクリア↓
68     TCR1 = 3; //1/8分周↓
69     TCR1 |= 0x20; //GRAコンペアマッチ↓
70     GRA1 = 30720; //GRA設定↓
71     TIER1 |= 1; //GRAコンペアマッチ割り込み許可↓
72 ↓
73     TSTR |= 2; //カウンタスタート↓
74 }↓
```

デバイスドライバは時間管理のため約 10ms 間隔で起動する割り込みが必要です。そのための設定を記述します。タイマは内蔵のタイマのうちどれを使っても構いません。ただし、10ms 間隔というのは比較的長い時間なのでカウンタがオーバーフローしないためには、16 ビットタイマが良いでしょう。

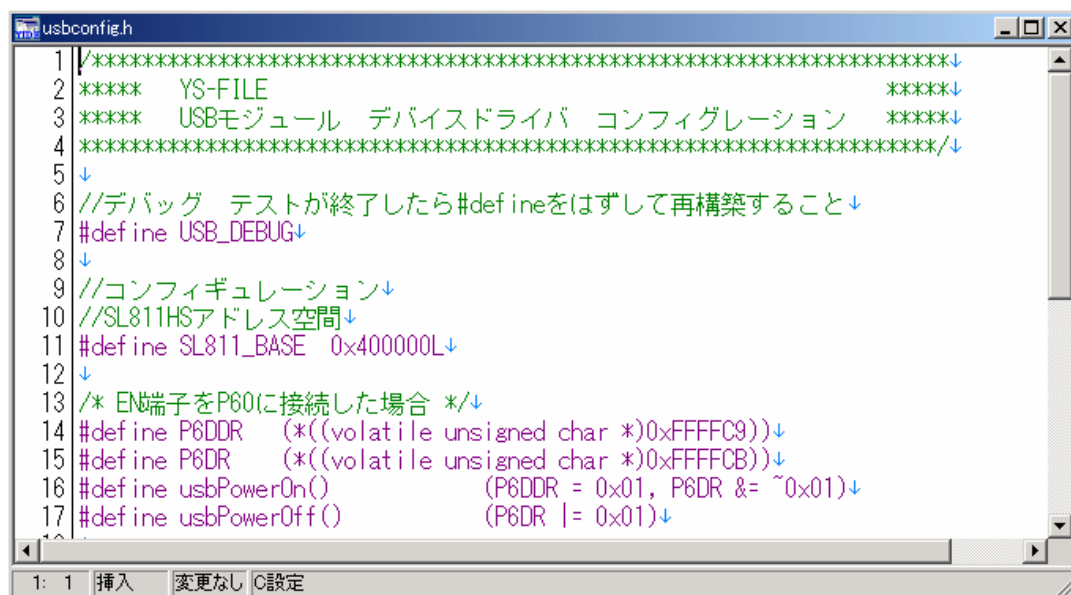
時間間隔の 10ms というのはだいたい値で、9ms でも 11ms でも構いません。

タイマの設定は、このデバイスドライバのプロジェクトとは別に、タイマを使った小さなプログラムを作成し、動作を確認してから、そこからコピーするようにすると良いでしょう。

以上で DEVICE.C の変更は終わりです。次に UsbConfig.h の変更です。

UsbConfig.h の変更

プロジェクトウィンドウの UsbConfig.h をダブルクリックして UsbConfig.h を開きます。



```
1 | /*****↓
2 | ***** YS-FILE *****↓
3 | ***** USBモジュール デバイスドライバ コンフィグレーション *****↓
4 | *****/↓
5 | ↓
6 | //デバッグ テストが終了したら#defineをはずして再構築すること↓
7 | #define USB_DEBUG↓
8 | ↓
9 | //コンフィギュレーション↓
10 | //SL811HSアドレス空間↓
11 | #define SL811_BASE 0x400000L↓
12 | ↓
13 | /* EN端子をP60に接続した場合 */↓
14 | #define P6DDR *((volatile unsigned char *)0xFFFFC9)↓
15 | #define P6DR *((volatile unsigned char *)0xFFFFCB)↓
16 | #define usbPowerOn() (P6DDR = 0x01, P6DR &= ~0x01)↓
17 | #define usbPowerOff() (P6DR |= 0x01)↓
18 | ↓
19 | ↓
20 | ↓
21 | ↓
22 | ↓
23 | ↓
24 | ↓
25 | ↓
26 | ↓
27 | ↓
28 | ↓
29 | ↓
30 | ↓
31 | ↓
32 | ↓
33 | ↓
34 | ↓
35 | ↓
36 | ↓
37 | ↓
38 | ↓
39 | ↓
40 | ↓
41 | ↓
42 | ↓
43 | ↓
44 | ↓
45 | ↓
46 | ↓
47 | ↓
48 | ↓
49 | ↓
50 | ↓
51 | ↓
52 | ↓
53 | ↓
54 | ↓
55 | ↓
56 | ↓
57 | ↓
58 | ↓
59 | ↓
60 | ↓
61 | ↓
62 | ↓
63 | ↓
64 | ↓
65 | ↓
66 | ↓
67 | ↓
68 | ↓
69 | ↓
70 | ↓
71 | ↓
72 | ↓
73 | ↓
74 | ↓
75 | ↓
76 | ↓
77 | ↓
78 | ↓
79 | ↓
80 | ↓
81 | ↓
82 | ↓
83 | ↓
84 | ↓
85 | ↓
86 | ↓
87 | ↓
88 | ↓
89 | ↓
90 | ↓
91 | ↓
92 | ↓
93 | ↓
94 | ↓
95 | ↓
96 | ↓
97 | ↓
98 | ↓
99 | ↓
100 | ↓
```

最初に SL811_BASE の値を変更します。

SL811_BASE は、USB メモリ IO モジュールが接続されているメモリ空間の開始アドレスです。

```
//コンフィギュレーション
#define SL811_BASE 0x400000L //USBメモリコントローラICレジスタアドレス
```

次に USB メモリ IO モジュールの _ENA 端子に接続したポートの入出力と出力設定を変更します。

```
/* EN 端子を P60 に接続した場合 */
#define P6DDR (*(volatile unsigned char *)0xFFFFC9)
#define P6DR  (*(volatile unsigned char *)0xFFFFCB)
#define usbPowerOn()          (P6DDR = 0x01, P6DR &= ~0x01)
#define usbPowerOff()         (P6DR |= 0x01)
```

usbPowerOn()は、ポートを出力に設定して、0(L)を出力する命令を記述します。なお H8 の場合はポートの入出力方向設定レジスタは書き込み専用レジスタなので |= などの演算代入命令は使えません。必ず単純代入を使って設定します。このため他のビットにも影響を与えますから、ポートに他の装置がつながっている場合は十分注意して下さい。

例えば上記の例で P6DDR=0x01 とありますが、ポート 6 のビット 0 は出力ポートになりますが、ビット 1 から 7 までは 0 になってしまいます。つまり入力ポートになってしまいます。ポート 6 の他のビットを出力ポートとして使っている場合は注意が必要です。

usbPowerOff()は、ポートに 1(H)を出力する命令を記述します。

最初の

```
#define USB_DEBUG
```

ですが、この USB_DEBUG が定義されていると、起動時に USB メモリの情報の表示とエラーが起こったときのメッセージを表示します。デバッグが終了するまで、この機能を有効にしておくといいでしょう。

プログラムが完成した段階でこの#define 定義をコメントアウトすればメッセージは表示されません。

```
//#define USB_DEBUG
```

以上でデバイスドライバの移植はすべて終了です。
メイクしてエラーのないことを確かめて下さい。

次にテストプログラムの実行と動作確認に進みます。

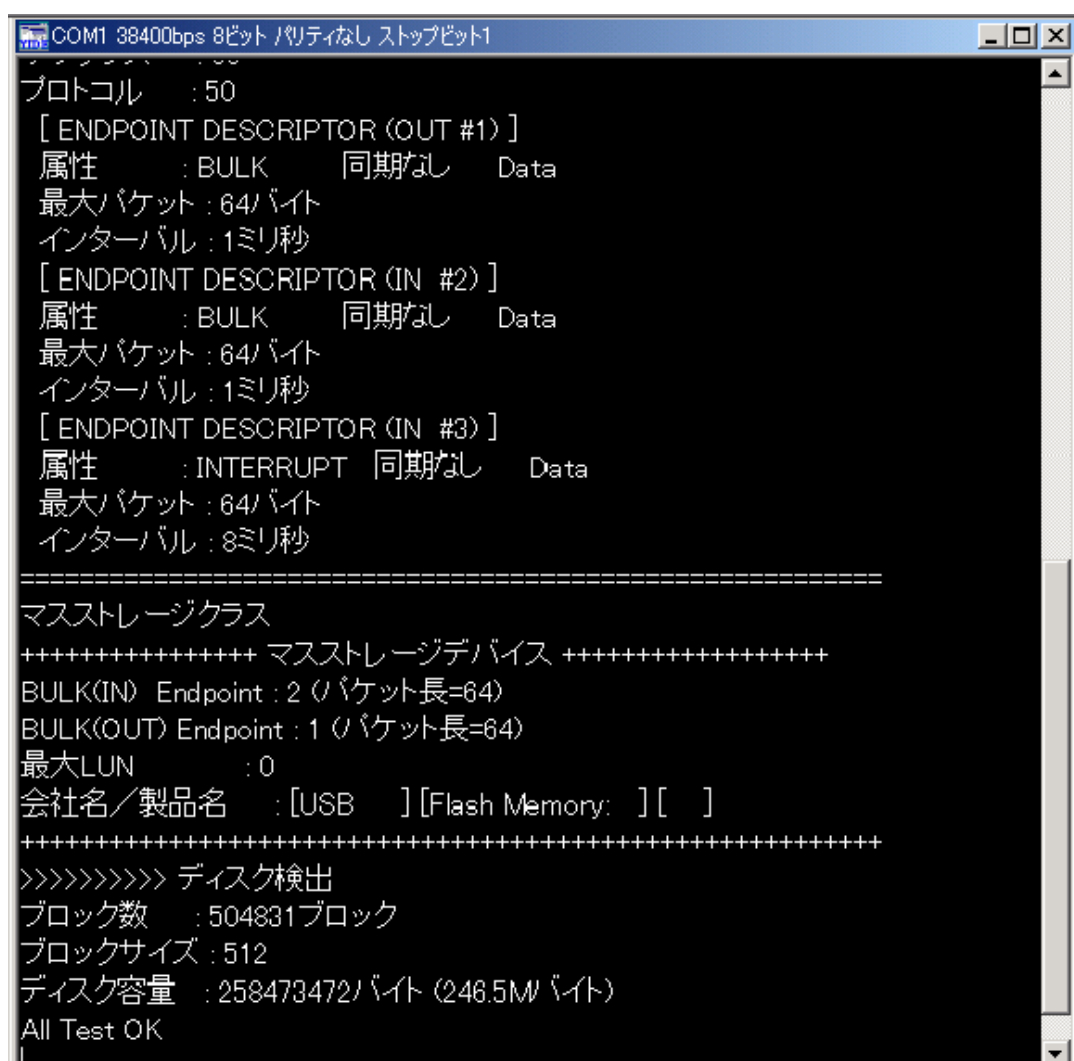
テストプログラムの実行

データが壊れてもよい USB メモリを用意して下さい。電源を入れる前に USB コネクタに USB メモリを挿入して下さい。

テストプログラムを ROM に書き込み後、電源を投入するとテストプログラムが実行されます。

テストプログラムは最初 USB メモリを認識して、情報を表示します。次に実際にテストデータを書き込み、読み込みをしてテストします。

正常なら以下の画面が出ます。



```
COM1 38400bps 8ビット パリティなし ストップビット1
プロトコル : 50
[ ENDPOINT DESCRIPTOR (OUT #1) ]
属性 : BULK 同期なし Data
最大パケット : 64 バイト
インターバル : 1ミリ秒
[ ENDPOINT DESCRIPTOR (IN #2) ]
属性 : BULK 同期なし Data
最大パケット : 64 バイト
インターバル : 1ミリ秒
[ ENDPOINT DESCRIPTOR (IN #3) ]
属性 : INTERRUPT 同期なし Data
最大パケット : 64 バイト
インターバル : 8ミリ秒
=====
マストレージクラス
+++++++ マストレージデバイス ++++++
BULK(IN) Endpoint : 2 (パケット長=64)
BULK(OUT) Endpoint : 1 (パケット長=64)
最大LUN : 0
会社名/製品名 : [USB ] [Flash Memory: ] [ ]
+++++++
>>>>>>>>> ディスク検出
ブロック数 : 504831 ブロック
ブロックサイズ : 512
ディスク容量 : 258473472 バイト (246.5M バイト)
All Test OK
```

最初に時刻が表示されますが、現在と合っていないで大丈夫です。なお、ファイルシステムに正しい時間を導入するには、マイコンがリアルタイムクロックなどの時計機能を持っていることと、デバイスドライバの移植が必要です。詳しくは後述します。

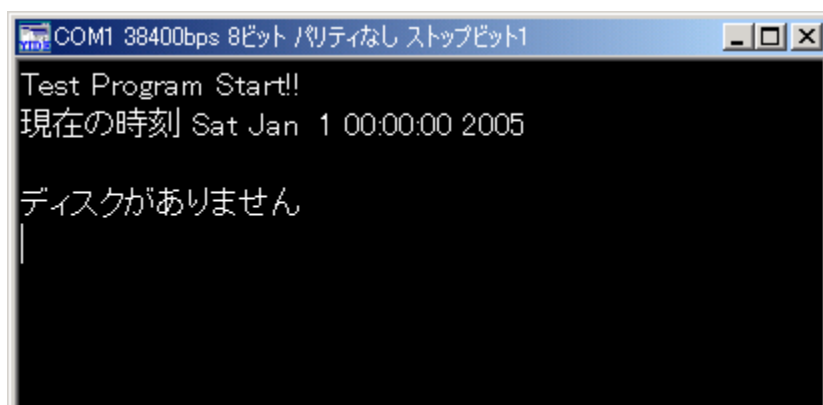
最後に"All Test OK"が表示されることを確認して下さい。

正しくないとき

前ページとは違う画面が出た場合は、配線かあるいはバスコントローラの設定が間違っています。

配線のミスなどがある場合は例えば次のような画面を表示します。

正しくない例 1

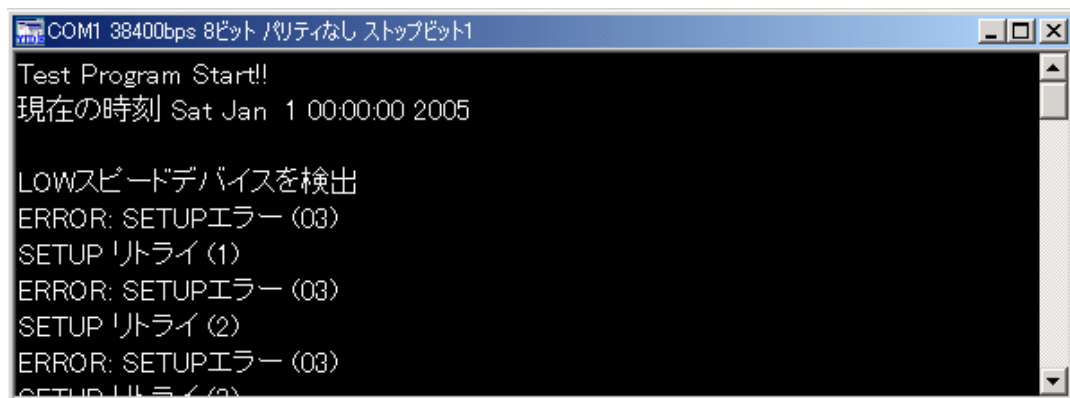


```
COM1 38400bps 8ビット パリティなし ストップビット1
Test Program Start!
現在の時刻 Sat Jan 1 00:00:00 2005
ディスクがありません
|
```

このエラーが出る場合は、_ENA 端子の配線が間違っているか、デバイスドライバの `UsbConfig.h` の変更のところでポートの設定が間違っている可能性があります。_ENA 端子が LOW になっているかテストなどで検査する必要があります。

また、USB メモリが挿入されていない場合やコネクタの接触不良の場合もこのエラーが出ます。

正しくない例 2



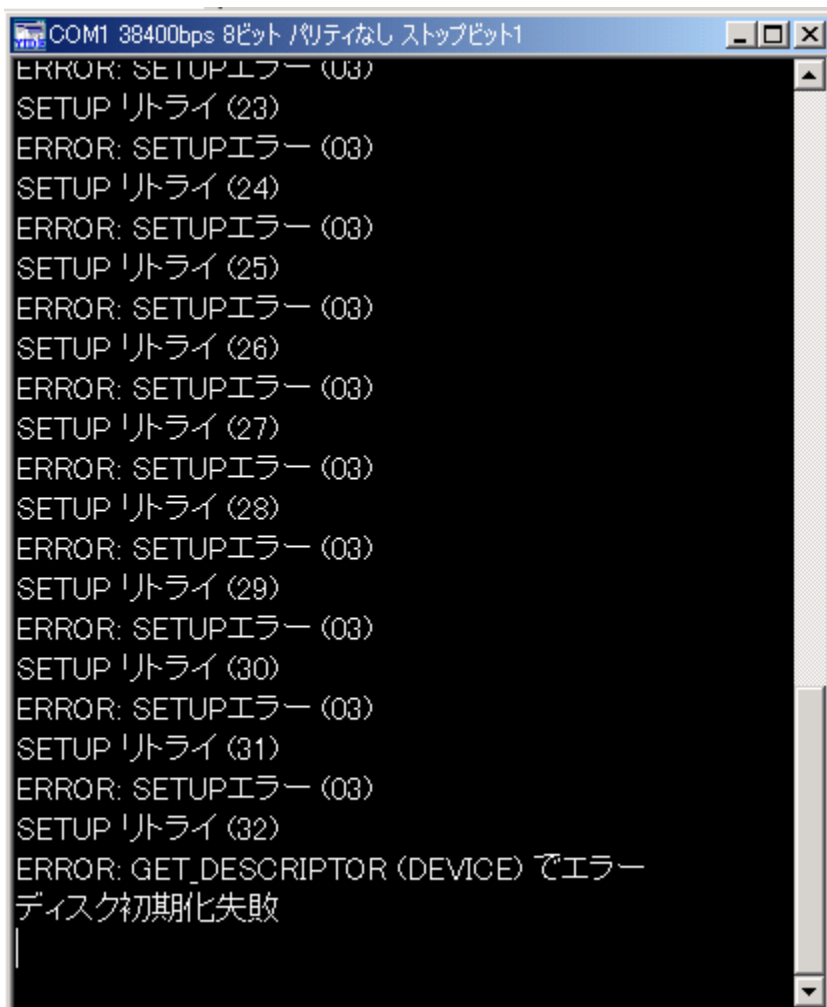
```
COM1 38400bps 8ビット パリティなし ストップビット
Test Program Start!!
現在の時刻 Sat Jan 1 00:00:00 2005

LOWスピードデバイスを検出
ERROR: SETUPエラー (03)
SETUP リトライ (1)
ERROR: SETUPエラー (03)
SETUP リトライ (2)
ERROR: SETUPエラー (03)
SETUP リトライ (3)
```

最初に「LOW スピードデバイスを検出」とあります。正常な場合は「FULL スピードデバイスを検出」となるはずです。

この場合は、CPU が USB メモリ IO モジュールとバス接続できていません。CS 端子やアドレスバス、データバスなどの配線が間違っているか、デバイスドライバでのバスの設定が間違っている可能性が高いです。

正しくない例 3



```
COM1 38400bps 8ビット パリティなし ストップビット1
ERROR: SETUPエラー (03)
SETUP リトライ (23)
ERROR: SETUPエラー (03)
SETUP リトライ (24)
ERROR: SETUPエラー (03)
SETUP リトライ (25)
ERROR: SETUPエラー (03)
SETUP リトライ (26)
ERROR: SETUPエラー (03)
SETUP リトライ (27)
ERROR: SETUPエラー (03)
SETUP リトライ (28)
ERROR: SETUPエラー (03)
SETUP リトライ (29)
ERROR: SETUPエラー (03)
SETUP リトライ (30)
ERROR: SETUPエラー (03)
SETUP リトライ (31)
ERROR: SETUPエラー (03)
SETUP リトライ (32)
ERROR: GET_DESCRIPTOR (DEVICE) でエラー
ディスク初期化失敗
```

この場合もバスまわりの配線が間違っている可能性があります。

対処方法

これらのエラーが出た場合の対処方法をいくつか示しておきます。

- 1 まず、配線に間違いがないか確認します。
- 2 デバイスドライバの設定に間違いがないか確認します。
- 3 ウェイトを最大にしてやってみて下さい。
- 4 GND ラインを強化してみてください。GND は CPU ボードの複数カ所から太い線でつなげると安定します。

第4章 YS-FILE の構築とテストプログラムの実行

デバイスドライバの移植が成功したら、ファイルシステムまであと少しです。これからの作業は機械的な作業で難しいことはありません。手順を間違えずやれば問題ありません。

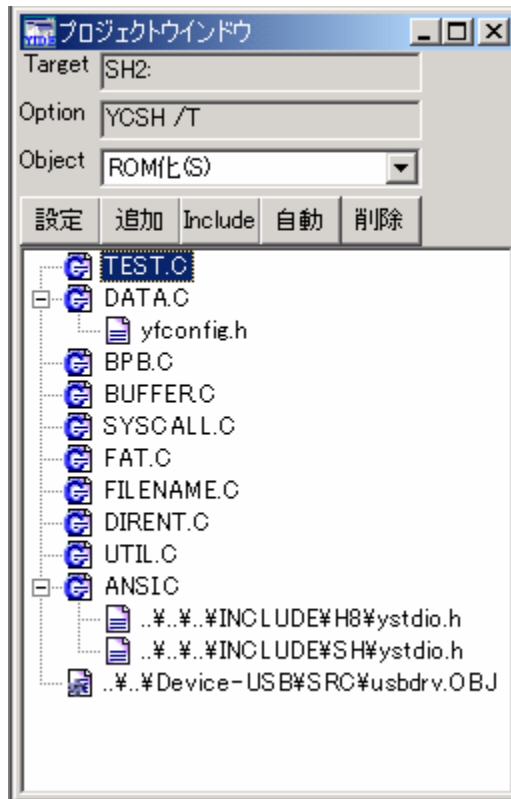
テストプログラムは miniDOS といって、小さな DOS(ディスクオペレーティングシステム)です。MS-DOS と似たコマンド体系でファイル、ディレクトリ操作ができます。

それではいよいよファイルシステムのコンフィグレーションとテストプログラムの実行に入ります。

YellowIDE のメニューの (ファイル) (プロジェクトを開く) でプロジェクトを開きます。プロジェクトは以下のフォルダにあります。

```
¥YellowIDE6¥YS-FILE-USB¥System¥prj¥test.yip
```

同じフォルダの中に test.yip と YSFILE.YIP の 2 つのプロジェクトがありますが、test.yip の方を開いて下さい。間違って YSFILE.YIP の方を開かないで下さい。

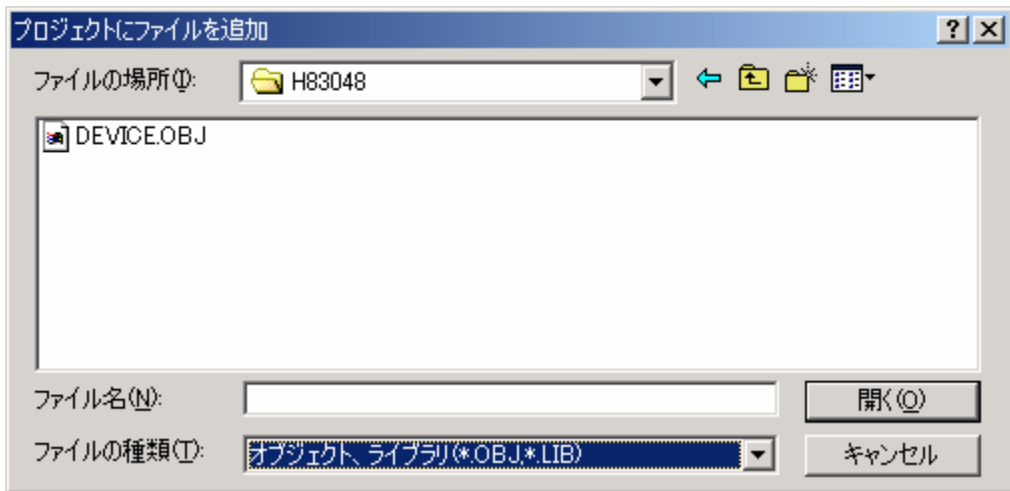


上図のようなプロジェクトウインドウが開きます。確認して下さい。これがYS-FILEのファイル群です。

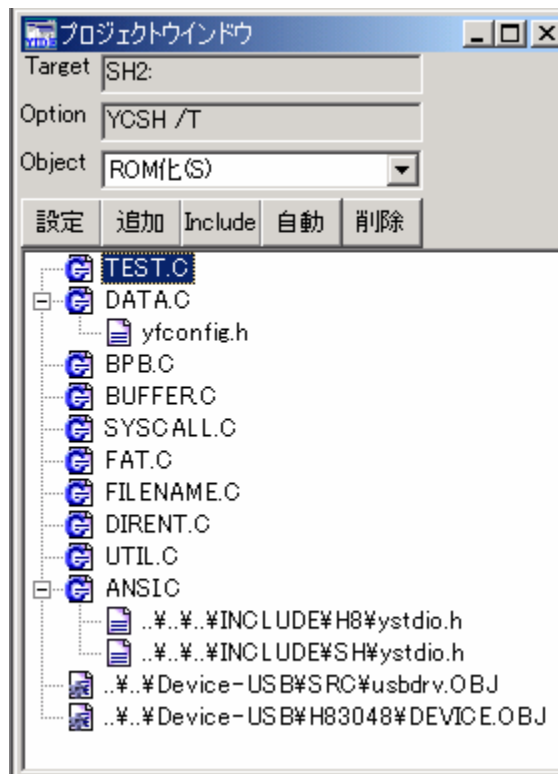
実はこのプロジェクトにはファイルが一つ足りません。先ほど移植したデバイスドライバのファイルが足りません。それを追加します。

(追加) ボタンをクリックします。ダイアログが開きます。フォルダをたどって、デバイスドライバのプロジェクトのあるフォルダまでたどります。

ここで注意するのは、追加するファイルはコンパイル済みの OBJ ファイルです。そこで、ファイルの種類を「オブジェクト、ライブラリ(*.OBJ,*.LIB)」にします。



そうすると DEVICE.OBJ が表れますので DEVICE.OBJ を選択します。



一番最後にファイル DEVICE.OBJ が追加されます。もしここが DEVICE.C になっていたら間違いです。DEVICE.C を削除してもう一度やり直します。必ず

DEVICE.OBJ を登録します。

プロジェクトの設定

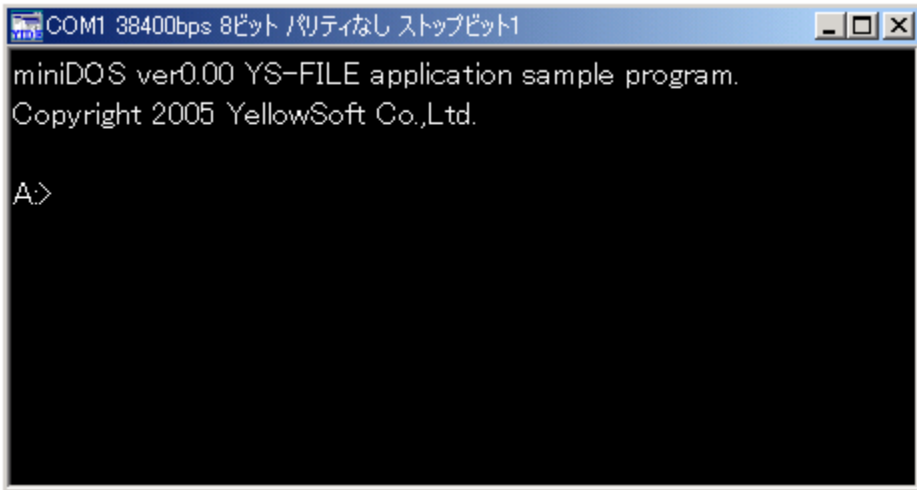
次にプロジェクトの設定を行いますが、デバイスドライバを移植したときのプロジェクトの設定をそのままコピーすることにします。

プロジェクトウインドウの(設定) ボタンをクリックして、「プロジェクトの設定」画面を開きます。左下の(他のプロジェクトからコピー) ボタンをクリックして、デバイスドライバを移植したときのプロジェクトを選択すれば、プロジェクトの設定がコピーされます。

テストプログラムの実行

プロジェクトの設定が済んだら、メイクしてすぐ実行できます。その前に フォーマットされた USB メモリを用意して中にテスト用のファイルをいくつか書き込んでおいて下さい。(もちろんパソコンで書き込みます、ファイルはどんなものでも構いませんがテキストファイルがあると確認が楽です)

ROM へ書き込み後、CPU を起動します。次のような画面が出れば正常です。



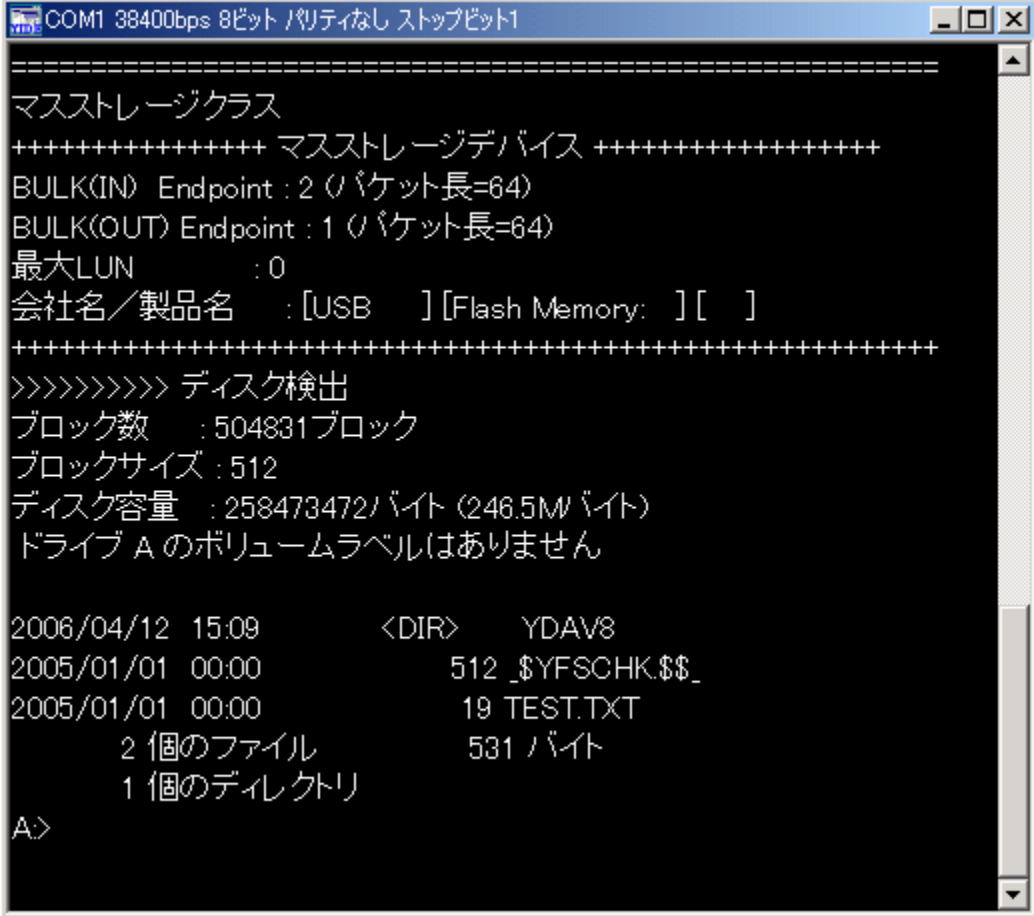
A:>はプロンプトと呼ばれるものでここからコマンドを打ち込みます。使い方はMS-DOS とほとんど同じです。コマンドの詳細については「YS-FILE プログラマーズマニュアル」をご覧ください。

MS-DOS になじみのない方にいくつかコマンドを紹介します。
まず、プロンプトから"DIR"と打ち込んで見ましょう。

```
A:>DIR
```

最後はリターンキーを押します。このコマンドは現在のディレクトリ内(カレントディレクトリと言います)のファイルおよびサブディレクトリの一覧を表示します。

注意 「ディレクトリ」という言葉になじみがない場合は「フォルダ」と置き換えてほぼ同じ意味です。



```
COM1 38400bps 8ビット パリティなし ストップビット
=====
マスストレージクラス
+++++++ マスストレージデバイス ++++++
BULK(IN) Endpoint : 2 (バケット長=64)
BULK(OUT) Endpoint : 1 (バケット長=64)
最大LUN      : 0
会社名/製品名 : [USB  ] [Flash Memory:  ] [  ]
+++++++
>>>>>>>> ディスク検出
ブロック数   : 504831ブロック
ブロックサイズ : 512
ディスク容量 : 258473472バイト (246.5Mバイト)
ドライブ A のボリュームラベルはありません

2006/04/12 15:09      <DIR>      YDAV8
2005/01/01 00:00                512 _$YFSCHK.$$_
2005/01/01 00:00                19 TEST.TXT
      2 個のファイル          531 バイト
      1 個のディレクトリ

A>
```

一番最初に USB メモリにアクセスすると、上図のようにディスク情報を表示します。その後、ファイル、サブディレクトリの一覧を表示します。

上に例ではサブディレクトリ「YDAV8」とファイル「TEST.TXT」があります。もう一つ見慣れないファイル「_\$YFSCHK.\$\$_」というファイルがあります。これは YS-FILE が勝手に作るファイルで電源断が起こった時にファイルシステムを修復するための情報が格納されています。このファイルにはアクセスしないで下さい。ただし削除することは構いませんが、YS-FILE は何度もこのファイルを作成します。(コンフィグレーションで作成しないようにすることができます。)

ファイル TEST.TXT がテキストファイルなら中身を見ることができます。テキストファイルの中身を見るコマンドは TYPE の後にファイル名を続けます。

```
A:¥>TYPE TEST.TXT
```

ファイルをコピーしたい場合は、例えば TEST.TXT を TEST2.TXT という名前でコピーしたい場合は次のようにコマンドを打ち込みます。

```
A:¥>COPY TEST.TXT TEST2.TXT
```

パソコンなどでコピーする感覚からするとだいぶ遅く感じるかも知れません。マイコンの能力と搭載メモリからすると仕方ありません。ただし、コンフィグレーションによって速くすることは可能です。

サブディレクトリの中のファイルも見たい場合は、まずカレントディレクトリを現在のディレクトリからサブディレクトリに移動します。今の例では YDAV8 に移動します。CD コマンドで移動します。

```
A:¥>CD YDAV8
```

そうするとプロンプトが A:¥>から A:¥YDAV8>に変わります。これはカレントディレクトリが YDAV8 に移動したことを表します。

ここで DIR コマンドを実行するとディレクトリの中身が見られます。

```
A:¥YDAV8>DIR
```

もとのディレクトリに戻りたい場合は、親のディレクトリ、あるいは上のディレクトリに移動するという意味で CD の後にピリオドを2つを続けます。

```
A:¥YDAV8>CD ..
```

その他のコマンドについては「YS-FILE プログラマーズマニュアル」をご覧ください。

エラーが出た場合

デバイスドライバのテストプログラムでエラーが出なかったにも関わらず、この段階でエラーが起きる場合があります。

これは他のデバイス、特に外付けの SRAM などと USB メモリ IO モジュールが干渉したことが考えられます。

デバイスドライバのテストプログラムでは SRAM と USB メモリ IO モジュールが交互にアクセスされることは少ないのですが、実際のプログラムでは頻繁に起こります。対策としては SRAM アクセス時と USB メモリ IO モジュールのアクセスの間にバスコントローラの設定でアイドルサイクルを挿入することです。(デフォルトの設定では挿入されている) しかし、一つのメモリ空間を SRAM と USB メモリ IO モジュールで分けあっている場合などはアイドルサイクルの挿入ができません。この場合の対策はなく、内蔵 RAM を使うしかありません。

その他の対策としては、GND 線の強化。外付け SRAM のウエイト数を多くするなどがあります。

画面の文字が文字化けするとき

このテストプログラム実行中に、表示される文字が文字化けを起こす場合があります。これはプログラムの問題ではなく YellowIDE のターミナルウインドウの表示の問題ですから大丈夫です。

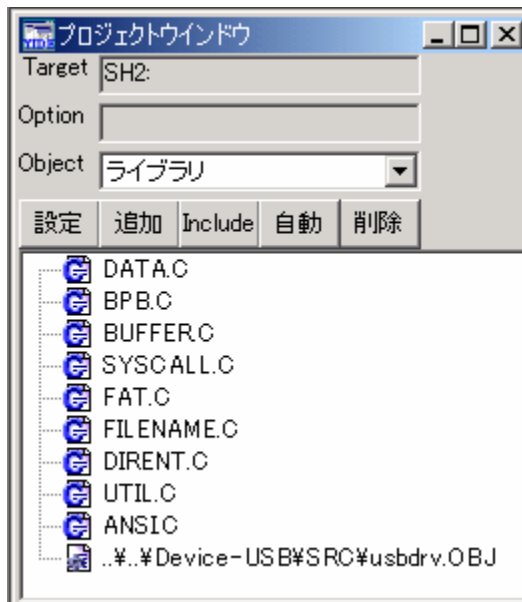
なお、YellowIDE Ver.6.42 以降では文字化けしないように改善されています。

第5章 YS-FILE のライブラリ化

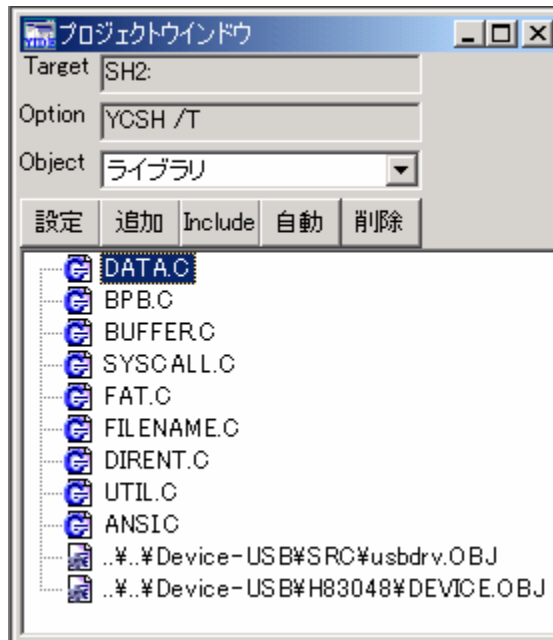
YS-FILE を使ったサンプルテストプログラムがうまく実行できたのなら、YS-FILE をライブラリ化します。

ライブラリ化することによって、YS-FILE のたくさんのファイルをプロジェクトに追加する必要がなくなり、ただ一つのファイル YSFILE.LIB を追加すれば良くなります。

YellowIDE のメニューの (ファイル) (プロジェクトを開く) で、先ほどのサンプルプログラムのプロジェクトと同じフォルダに YSFILE.yip というプロジェクトがありますから、それを開きます。



プロジェクトウインドウにはデバイスドライバの DEVICE.OBJ がまだ追加されていません。サンプルプロジェクトの時と同じように、デバイスドライバのテストプログラムのフォルダから DEVICE.OBJ を登録します。



拡張子が.OBJであることを確認して下さい。次にプロジェクトウィンドウの(設定)ボタンをクリックします。

プロジェクトの設定のターゲット画面で「CPUの種類」を選択して下さい。

以上で、プロジェクトの設定はすべて終わりです。

最後にメイクをすればライブラリが完成です。このプロジェクトフォルダの中にYSDFILE.LIBというライブラリができています。

アプリケーションプログラムからはこのライブラリファイルをプロジェクトに登録することによって使用できるようになります。

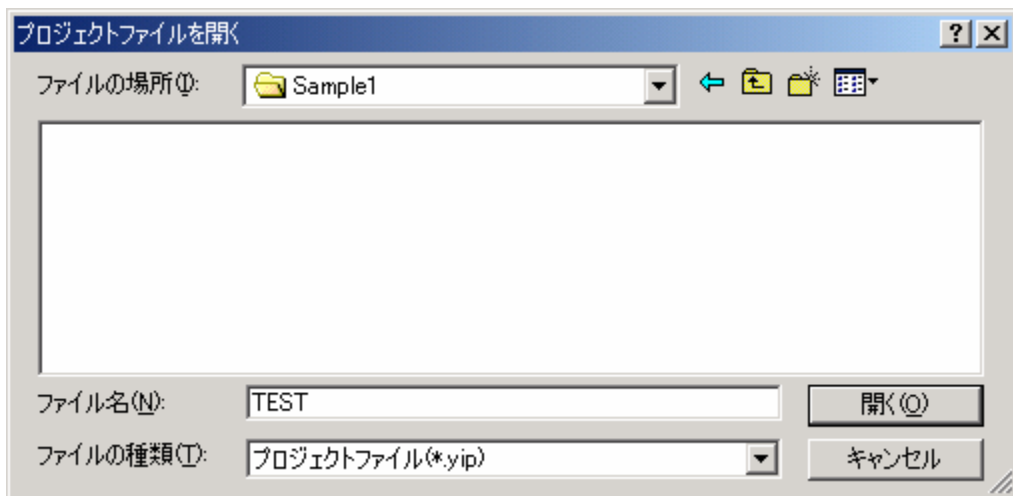
第6章 アプリケーションプログラムの作成

ライブラリができたので、それではアプリケーションプログラムから実際に利用してみましょう。

簡単なサンプルプログラムを用意しています。ただし、ソースファイルだけでプロジェクトは作成していません。プロジェクトを作成するところから始めます。

YellowIDE のメニューの (ファイル) (プロジェクトの新規作成) で、プロジェクトを作成します。

プロジェクトを作成する場所は、YS-FILE-USB フォルダの中の Sample フォルダの中の Sample1 フォルダです。プロジェクト名は何でもかまいません。例では TEST.YIP とします。



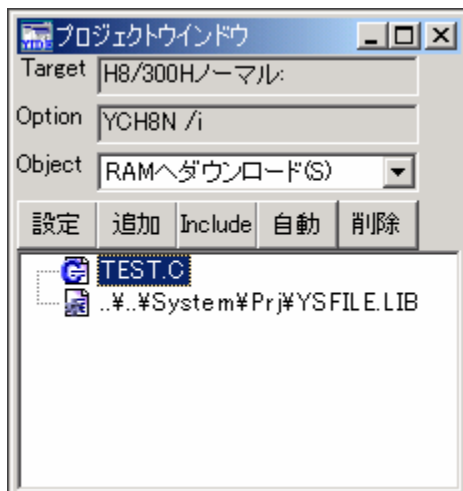
プロジェクトを作成したら、プロジェクトウインドウの (追加) ボタンをクリックしてソースファイルを登録します。ソースファイル名は TEST.C です。

次に、YS-FILE のライブラリもプロジェクトウインドウに追加します。プロジェクトウインドウの (追加) ボタンをクリックします。

ファイルの種類から「オブジェクト、ライブラリ(*.obj, *.lib)」を選択します。

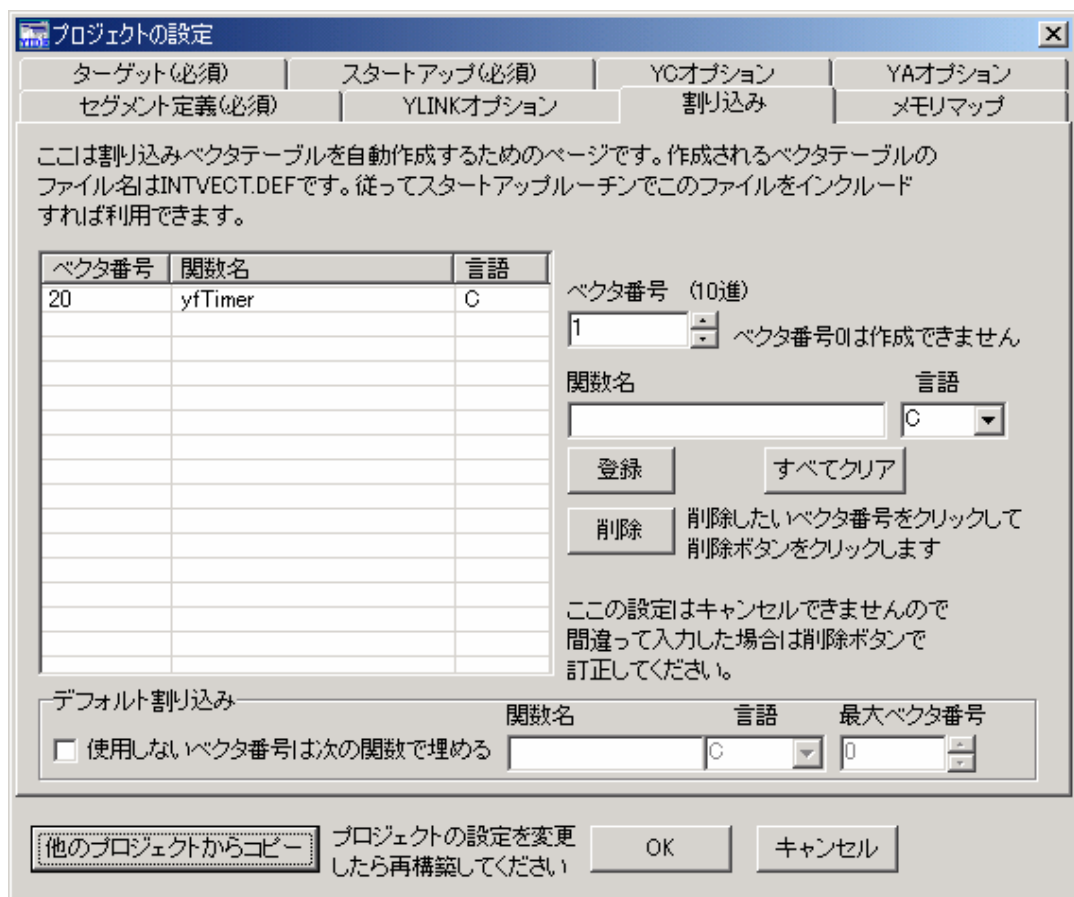
フォルダを移動して、YS-FILE-USB の中の System フォルダの中の prj フォルダの

中の YSFILE.LIB を選択します。



以上で必要なファイルの登録が終わりました。次にプロジェクトの設定を行います。プロジェクトの設定は各自、自分のシステムに合わせて行って下さい。

プロジェクトの設定で重要なのは、タイマ割り込みベクタの登録です。デバイスドライバの移植の時に使った約 10ms 間隔のインターバルタイマ割り込みのベクタ設定が必要です。これは忘れやすいので注意が必要です。



ベクタ番号はデバイスドライバの移植の時とまったく同じです。(上図の 20 は例です) 関数名は yfTimer です。

サンプルプログラムの解説

プロジェクトウィンドウの TEST.C をダブルクリックして下さい。このファイルについて少し解説します。

このサンプルプログラムは簡単なもので、USB メモリに TEST.TXT というファイルを作成して、そこに"TEST Hello, World¥n"を書き込みます。そして、もう一度このファイルを読み込んで内容を表示します。

```
#include <ysfile.h>           //YS-FILE を使う場合インクルード
#include <ystdio.h>           //ANSI 関数を使う場合インクルード
```

YS-FILE を使う場合は `ysfile.h` と `ystdio.h` をインクルードします。ANSI 関数を使わないでシステムコールだけ使う場合は `ystdio.h` のインクルードは不要ですが、とりあえず両方インクルードしておけば間違いありません。

```
void main()
{
    YFILE *fp;
    char buf[512];
    int c;
```

YFILE は従来の ANSI 関数の FILE に相当するものです。このように YS-FILE では先頭に Y がつきます。関数の場合も同様で、先頭に小文字の y がつきます。

```
#ifdef __YCH8__
    _ei();           //H8
#else
    WriteSR(0);     //SH
#endif
```

YS-FILE はデバイスドライバで割り込みを使用しますので割り込み許可命令を記述します。H8 系の場合は `_ei()` で SH 系の場合は `WriteSR(0)` です。このサンプルでは H8、SH 両方でコンパイルできるように `#ifdef` を使っています。

```
yfInitFileSystem();
```

YS-FILE を使用する場合は割り込み許可の後にこの関数を呼び出して下さい。必要な各種初期化を行います。

必ず、割り込み許可の後に `yfInitFileSystem()` を呼び出して下さい。順序が逆の場合は動作しません。この順序は SD カードとは逆になっています。

```
//ルートディレクトリにファイルを作成
if ((fp=yfopen("TEST.TXT", "w")) == NULL) {
    puts("Error ファイルが作成できません");
    return;
}
```

ファイルをオープンします。従来の fopen に相当するものが YS-FILE では yfopen になります。くれぐれも先頭の y を忘れないように。

なお、if 括弧の中の puts("Error ファイルが作成できません");には先頭に y がつきません。これはエラーメッセージは USB メモリのファイルに出力するのではなく、YellowIDE のターミナルウインドウに出力するわけですから YS-FILE とは何の関係もありません。したがって y がつきません。

先頭に y がつく場合とつかない場合が混在しますので十分注意が必要です。

```
//今作ったファイルに書き込む
yfputs("TEST Hello, World¥n", fp);
yfclose(fp);
```

ここで USB メモリの中のファイルに書き込みが行われます。

```
//そのファイルを表示する
if ((fp=yfopen("TEST.TXT", "r")) == NULL) {
    puts("Error ファイルが開けません");
    return;
}
while ((c = yfgetc(fp)) != EOF) {
    fputc(c, stdout);
}
yfclose(fp);
```

書き込んだファイルをもう一度、今度は読み込み属性("r")で開いて、1文字ずつ YellowIDE の画面に表示します。

ファイルから読み込む場合は yfgetc ですが、画面に出力する場合はファイルではあ

りませんから先頭の y は付けなくていいので fputc です。

このサンプルプログラムを実行すると、YellowIDE の画面に "TEST Hello, World¥n" と表示されます。また USB メモリの中に TEST.TXT というファイルが作成され、そこに "TEST Hello, World¥n" と書き込まれます。USB メモリをパソコンで読み込んでみて確認してみてください。

アプリケーション作成チェックシート

今までの注意点をまとめて見ました。以下のことを守って頂ければ、ファイルシステムを利用することができます。

- 1 プロジェクトに YSFILE.LIB を追加する
- 2 プロジェクトの設定でタイマ割込みのベクタ登録を忘れないように
- 3 ysfile.h と ystdio.h をインクルードする
- 5 割り込みを許可する
- 4 割り込み許可の後に yfInitFileSystem() を呼び出す
- 6 ANSI のファイル入出力関数を使う場合は先頭に [y] を付ける。
例 FILE YFILE fopen yfopen fputc yfputc

第7章 YS-FILE のコンフィグレーション

YS-FILE はさまざまなコンフィグレーションをすることができます。詳しくは「YS-FILE プログラマーズマニュアル」を見て下さい。ここでは、プログラムサイズを小さくする、使用メモリを少なくする、速度を速くする3点に絞って、コンフィグレーションの仕方を説明します。

YS-FILE をコンフィグレーションするためには、YS-FILE のサンプルプログラム、miniDOS を作ったプロジェクトまで戻る必要があります。YellowIDE のメニューの（ファイル）（プロジェクトを開く）で¥YS-FILE-USB¥System¥prj¥test.yip を開きます。

プロジェクトウィンドウのファイルの中で、コンフィグレーションに関するファイルは以下の2つです。

yfconfig.h	YS-FILE のコンフィグレーション
ystdio.h	YS-FILE のうち ANSI 関数のコンフィグレーション

1 プログラムを小さくする方法

プログラムを小さくする方法をいくつか紹介します。

ANSI 関数を使わない。

ANSI 関数を使わないで、システムコールだけ使えばプログラムは小さくなります。ただし、システムコールだけによるファイル入出力はバッファを使ったブロック単位の扱いになりますので不便ではあります。例えば ANSI 関数の yfgetc はファイルから 1 文字を読み込みますが、システムコールの場合はある程度まとまった単位、例えば 512 バイトとかを一度に読み込みます。

システムコールでも 1 バイトだけの入出力は可能です。しかし 1 バイトごとにディスクにアクセスされるため相当効率は悪くなります。

したがって、ある程度まとまった単位（512 バイト以上 512 バイト単位）の入出力でよい場合はシステムコールだけでプログラムを組みます。

使用できるフォーマットタイプを制限する。

YS-FILE は FAT12、FAT16、FAT32 のフォーマットタイプすべてに対応しています。例えば FAT16 のディスクしか扱わないのであれば FAT12 と FAT32 を使用不可とすることで、プログラムサイズを小さくできます。

yfconfig.h の以下の部分を修正します。

```
/*
***** FAT の選択 *****
*/
#define YSFILE_FAT12
#define YSFILE_FAT16
#define YSFILE_FAT32
```

使用しないフォーマットタイプをコメントアウトします。上は FAT16 しか使用しない場合の例です。

セーフシステムを使用しないようにする

YS-FILE は電源断から、ファイルシステムを修復する機能を持っていますが、この機能を無効にすることによってプログラムサイズを小さくできます。

```
/*
***** セーフシステム *****
***** 電源断からの回復システムを有効にする *****
***** 有効にすると書き込み速度が遅くなります *****
*/
#define SAFE_SYSTEM
```

#define SAFE_SYSTEM の一行をコメントアウトして下さい。

yfprintf、yprintf 関数を使う場合

yfprintf 関数や、yprintf 関数はそれだけで大きなプログラムです。もし、YS-FILE の yfprintf 関数と、ファイルではなく画面へ出力する通常の fprintf 関数の両方を使っているなら、共通のルーチンを利用させることによって、プログラムサイズを小さくできます。

```
/*  
***** コンパイラ付属の sprintf を使用 *****  
*****/  
#define PRINTF_COMLIB
```

#define PRINTF_COMLIB のコメントをはずして下さい。コメントをはずすことによつて、yfprintf はコンパイラ付属の sprintf を利用しますのでルーチンの共通化ができます。

yfprintf 関数だけを使って、通常の fprintf 関数を使っていないなら、上記のコンフィグレーションをしても無駄です。しかしもし、yfprintf 関数で浮動小数を扱わないのであれば、プログラムを小さくすることができます。

```
/*  
***** yfprintf 関数での浮動小数点の使用 *****  
*****/  
//#define PRINTF_FLOAT
```

#define PRINTF_FLOAT の 1 行をコメントアウトして下さい。

デバイスドライバのデバッグ用関数はずす。

デバイスドライバの中にはデバイス情報を表示する関数や、エラーが起こった場合にメッセージを表示する関数が含まれています。ある程度開発が進み、デバイスに関して問題ないと判断した時点でこれらの関数はずした方が良いでしょう。かなりのプログラムの節約になります。

デバイスドライバを作成するプロジェクトにおいて Usbconfig.h の中の以下の一行をコメントアウトします。

```
//デバッグ テストが終了したら#define をはずして再構築すること  
//#define USB_DEBUG
```

修正したら再構築をします。

2 使用メモリを少なくする方法

プログラムサイズを小さくする方法は同時に使用メモリも少なくします。したがって前項で説明した対策は有効です。ただし、それほど効果はありません。

一番大きな効果を得る方法はバッファを少なくすることです。

しかしながらバッファサイズを小さくすると確実にファイル処理速度は遅くなります。メモリをとるか、速度をとるか、トレードオフの関係になります。

YS-FILE のデフォルトの設定ではバッファサイズをほぼ最小にしています。したがって、事項で説明する処理速度を速くする方法をしないことが、メモリの節約になります。

3 ファイル処理の速度を速くする方法

プログラムサイズを小さくする方法で説明した ANSI 関数を使わない方法とセーフシステムを使用しない方法は有効です。処理速度は向上します。

次にメモリに余裕があれば、バッファのサイズを増やします。バッファのサイズを増やせば処理速度は向上します。扱うファイルサイズの最大を目安にメモリの許す範囲内でバッファを増やします。

```
/******  
**** セクタバッファ ****  
*****/  
//セクタバッファの数 512 バイト単位 2 以上を推奨  
#define MAX_SECTOR_BUFFER 1 //最低 1 以上  
  
//FAT バッファの数 512 バイト単位 最低 2 4 以上を推奨  
#define MAX_FAT_BUFFER 2 //最低 2 以上  
  
//ディレクトリエントリバッファの数 512 バイト単位 2 以上を推奨  
#define MAX_DIRENT_BUFFER 1 //最低 1 以上  
  
//空クラスタバッファの数 5 以上を推奨  
#define MAX_EMPTY_CLUST_BUFFER 5 //最低 1 以上
```

セクタバッファの数、FAT バッファの数、ディレクトリエントリバッファの数をそ

れぞれ増やします。1つ増やすと512バイト増えます。

大きなファイルを扱う場合はセクタバッファを多めにすると良いでしょう。

ディスク容量の大きなUSBメモリでたくさんのファイルが入っているディスクを扱う場合はFATバッファを多めにすると良いでしょう。

たくさんのファイルを扱う場合はディレクトリエントリバッファの数を多めにすると良いでしょう。

空きクラスタバッファの数を増やすと、容量の大きなUSBメモリにたくさんのファイルが入っている状態で新規にファイルを作成する処理が速くなります。ここは1つ増やしても4バイトしかメモリ消費量は増えませんが、あまり増やすと逆に速度が落ちる場合があります。頻繁にファイルを作成する場合を除き5~10くらいが良いでしょう。

上記のうちで一番効果があるのはセクタバッファの数です。(MAX_SECTOR_BUFFER) ただし、必要以上に大きくしても意味がありません。1Kバイトのファイルしか扱わないのにバッファのサイズを10Kバイトにしてもまったく意味がありません。

ANSI関数を使う場合は、ANSI関数の方のバッファも同時に増やします。(同じくらいがちょうどいい)

セクタバッファの数を増やしても、ANSI関数のバッファを増やさなければ効果はありません。

例えばセクタバッファの数を4とした場合(512×4=2Kバイト)はANSI関数のバッファも2Kバイトにします。方法は以下の通りです。

プロジェクトウィンドウのANSI.Cの下にあるystdio.hを開きます。

H8の場合は..`¥..¥..¥INCLUDE¥H8¥ystdio.h`です。

SHの場合は..`¥..¥..¥INCLUDE¥SH¥ystdio.h`です。

```
#define YBUFSIZ      512*4    //バッファサイズ 512 バイト以上 512 バイト単位  
                        で確保すると効率がよい
```

YBUFSIZのところを512*4とします。

YBUFSIZ の値は 512 バイト以上 512 バイト単位にすると最高の効率が得られます。

4 一度にオープンできるファイルの数を増やす

一度にオープンできるファイルの数は、デフォルトで 2 になっています。もっとたくさんファイルを開いておきたい場合は変更できます。

yfconfig.h の以下の部分を変更します。

```
/******  
**** 一度にオープンできるファイルの数 ****  
**** 無用に多くしないでください ****  
*****/  
#define MAX_OPEN_FILE 2
```

また、ANSI 関数を使う場合は ystdio.h も同時に変更します。

```
#define YFOPEN_MAX 2 //最大同時オープンファイル数
```

5 ライブラリの再構築

必要なコンフィグレーションが終わったら、YellowIDE のメニューの(プロジェクト) (再構築)で再構築して下さい。(メイク)ではなく(再構築)することに注意して下さい。サンプルプログラムの miniDOS を実行してみましょう。

問題なければ YS-FILE のライブラリを作成しますので、ライブラリのプロジェクト ¥YS-FILE¥System¥prj¥YSFILE.yip を開いて同様に (再構築) します。

6 その他の使い方および注意点

メッセージを非表示にする

YS-FILE のサンプルの miniDOS を実行して分かるように、起動後、最初に USB メモリにアクセスした時、USB メモリ情報を表示するようになっています。また、デバイスドライバに何らかのエラーが起こった場合もエラーメッセージを表示するようになっています。

これらのメッセージを表示させないようにするには、デバイスドライバの移植のところまで戻ります。

デバイスドライバのプロジェクトで UsbConfig.h を開きます。
一番最初の #define 文をコメントアウトします。

```
/*
*****
*****   YS-FILE                               *****
*****   USB モジュール デバイスドライバ   コンフィグレーション *****
*****/

//デバッグ テストが終了したら#define をはずして再構築すること
//#define USB_DEBUG
```

コメントアウトして、メイクします。

メイクが終了したら、今度はライブラリも再構築しなければなりません。プロジェクト ¥YS-FILE-USB¥System¥prj¥YSFILE.yip を開いて、メイクして下さい。

ファイルの日付について

YS-FILE でファイルを作成した時のファイルのタイムスタンプ(時刻、日付)はデバイスドライバ内の関数を変更することによって修正できます。

タイムスタンプを正しい時間にするにはターゲットのマイコンがリアルタイムクロック(RTC)などの機能を持っていないとできません。ほとんどのマイコンは RTC をもっていませんので、その場合、タイムスタンプは変化しませんから無視する以外にありません。

RTC を付けて、ファイルスタンプを正しい時刻にするにはデバイスドライバ内の関数を変更します。

デバイスドライバを移植したときのプロジェクトを開き、DEVICE.C を開きます。関数 yfGetNowTime()が最後の方にありますから、それを変更します。

```
/*
*****
*****   日付の取得                               *****
*****
*****/
int yfGetNowTime(struct tm *ptime)
{
    time_t t;
    time(&t);
    *ptime = *localtime(&t);
    return 0;
}
```

この関数の中で、RTC から時刻を読み出し、ANSI 関数で使われる struct tm 構造体のフォーマットに変換して、*ptime に格納します。

RTC から時刻を読み出す具体的な方法は RTC の仕様や RTC の接続方法によって異なりますからここでは説明できません。ユーザが自身で考える必要があります。

struct tm 構造体のフォーマットですが、以下のようになります。

```
struct tm {
    int tm_sec;    /* seconds [0,59] */
    int tm_min;   /* minutes [0,59] */
    int tm_hour;  /* hours [0,23] */
```

```
int tm_mday; /* day of month [1,31] */
int tm_mon; /* month of year [0,11] */
int tm_year; /* year from 1900 */
int tm_wday; /* day of week [0 - 6] */
int tm_yday; /* day of year [0 - 365] 設定しなくても OK */
int tm_isdst; /* Summer Time flag [0, 1] 設定しなくても OK */
};
```

修正が済んだら、メイクをし、さらに YSFILE のライブラリも再構築します。