TrueSTUDIO から Eclipse への乗り換えガイド
Rev.2
（花岡ちゃんに花束を http://cyberworks.cocolog-nifty.com/ ）

お世話になったサイト　http://ameblo.jp/henachokox/entry-11207507403.html
ありがとうございます。

目的:
TrueSTUDIO の無償版で作った STM32F4 のプロジェクトを Eclipse を使って構築した開発環境に持ってくる。コードサイズの制限なしにソフト開発ができる。

概要:
TrueSTDIO のライブラリ、リンカスクリプト、ST-Link gdb サーバなどを拝借し、IDE とコンパイラをフリーソフトであつらえる。とりあえず浮動小数点演算をハードでやる設定はやらない。（ちゃんと設定されたかどうかがわからなかったので）

STEP 1　開発環境のインストール

概要:
Eclipse を核にした STM32F4 の開発環境をインストールする。

準備:
ワークスペースになるフォルダを作っておく。XP なら c 直下、Vista 以降ならマイドキュメントがなにかとわかりやすい。
Eclipse でワークスペースを求められたらコレを指定する。

1・下記から Eclipse IDE for C/C++ Developers(Eclipse CDT) をインストール
解凍したフォルダを適当な場所に。どこでもいいが XP なら C 直下、Vista 以降ならマイドキュメントがなにかとわかりやすい。
http://www.eclipse.org/downloads/

2・Sourcery CodeBench Lite Edition を下記ページの ARM processors から"Download the EABI Release"をクリックでダウンロードしてインストール。
インストーラ形式なので起動して指示に従う。
http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/

3・以上をインストールしたら Eclipse を起動

4・Eclipse のメニュー[Help]-[Install New Software]の"Work with"に下記 URL を入れると下の大きなボックスに"Zylin Embedded CDT"と表示されるのでコレにチェックを入れた上、NEXT ボタンをクリックしてインストール
http://opensource.zylin.com/zylincdt

5・上と同じ手順で ARM GCC Toolchain をインストール。
http://gnuarmeclipse.sourceforge.net/updates

STEP 2　プロジェクトの設定

概要:
TrueSTUDIO 無償版で作ったプロジェクトを新しい環境に持ってくる。TrueSTUDIO で作ったプロジェクフォルダ内のファイルを流用する。
TrueSTUDIO 無償版がインストールされ、動くプロジェクトがあることが前提なので、ゼロからやる場合は LED 点滅な

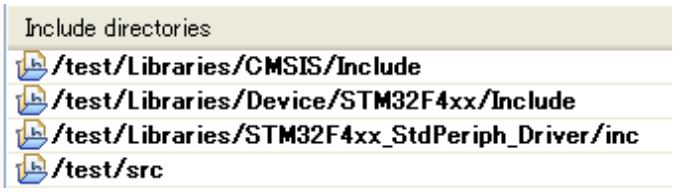
どの簡単なプロジェクトを作っておく必要がある。

1・新しいプロジェクトを作る
1-1:[File]-[New]-[C Project] で C Project ウィンドウを表示。
1-2:下記の設定をする。
Project name →好きな名前を英文でつける
Project type →[ARM Cross Target Application]の＋タブをクリックすると表示される[Empty Project] を選択。
Toolchains → [ARM Windows GCC (Sourcery G++ Lite)] を選択。
1-3:NEXT ボタンをクリックして設定完了(最後はモチロン FINISH ボタン)
1-4:画面左の Project Exploler にプロジェクトフォルダが見えていることを確認。
＊ソースコードの日本語のコメントが文字化けしているときは、Eclipse の画面左 Project Explorer で、文字化けしているファイルを右クリックし、表示されるメニューから Properties を選択。ダイアログが表示されるのでこれで文字コードを変更する。

2・ソースコードを持ってくる
2-1:Eclipse の画面左 Project Explorer でプロジェクトのフォルダを右クリックし[New]-[Source folder]を選択。名前は"src"にする。
2-2:TrueSTUDIO 無償版で作ったプロジェクトフォルダを Windows 上で開く。(つまりパソコン上で普通にフォルダを開く)。ここの src フォルダを開いておく。
2-3:TrueSTUDIO 無償版のプロジェクトの src フォルダの中身をすべて選択、ドラッグして、Eclipse の画面左 Project Explorer の今作ったプロジェクトフォルダの src フォルダにドロップする。(ややこしいので注意) これでソースファイルを持ってくる。(表示されるダイアログで Copy を選ぶ)
＊コピー後、拡張子が.s のファイルは大文字で.S にする。Project Explorer で変更したいファイルを表示させ、そのファイル上でマウスの右ボタンをクリックして表示されるメニューから[Rename..]を選ぶと変更できる。

3・ライブラリとリンカスクリプトを持ってくる
3-1・2の方法で TrueSTUDIO 無償版で作ったプロジェクトフォルダから Libraries フォルダをフォルダごと Eclipse の画面左 Project Explorer の今作ったプロジェクトフォルダにドラッグしてコピーする。
3-2・同じく stm32_flash.ld をコピーする。
3-3・Eclipse を一度終了、再起動し Project Explorer でプロジェクトフォルダを開き、追加したファイルを(念のため)確認する。

4・インクルードを設定
4-1:[Project]-[Proparties]で c/c++ General の＋タブをクリック、表示される項目から Path and Symbols を選択、表示される画面の Includes タブをクリック。
(Proparties がグレーになっていて選べないときはプロジェクトフォルダをクリックしてハイライトする)
4-2:Languages の GNU C をクリック。(重要)
4-3:Add..ボタンをクリックし開いた画面から Workspece ボタンをクリック、Libraries フォルダから下記画像のフォルダを選択する。(これはプロジェクト名が test の場合)



4-4:すべてのフォルダを選択したら右下の Apply ボタンをクリック、ウィンドウはまだ閉じない。

5・シンボルの定義
5-1:4で開いたウィンドウの Symbols タブをクリック。
5-2:Languages の GNU C をクリック。(重要)
5-3:Add..ボタンで下記のシンボルを順次入力。HSE_VALUE だけ value に 8000000 を入力、あとは空白のまま。
(8000000 は STM32F4 の外付け水晶周波数。Discovery では 8MHz なので 8,000,000Hz、コレを間違えるとシリアルの

ボーレートなどが狂ってくる）

HSE_VALUE 8000000
STM32F4XX
USE_STDPERIPH_DRIVER
USE_STM32F4_DISCOVERY

5-4：すべてを入力したら Apply ボタンをクリック、まだウィンドウは閉じない。

6・リンカスクリプトの設定
6-1：4で開いたウィンドウの左側で c/c++ Build の＋タブをクリックし、表示された項目から Settings をクリック。
6-2：右側の画面で Tool Settings タブをクリックして表示。（すでにこれが表示されていると思います）
6-3：ARM soucery Windows GCC C Linker のすぐ下の General をクリックし表示される画面の Script File に下記を入力。（他の General と間違えないように。リンカです。）
../stm32_flash.ld
6-4：Do not use standerd start files のチェックをはずし、Remove unused sections にチェックを入れる。
6-5：すべてを入力したら Apply ボタンをクリック、OK でウィンドウを閉じる。

＃＃＃＃＃ここまでできたら[Project]-[Build All]でビルドしてみる＃＃＃＃＃＃

（念のため一度 Eclipse を終了、再起動したほうがいい）
画面左側 Project Explorer ウィンドウで現在のプロジェクトフォルダを開き、Debug フォルダの中にプロジェクト名.elf のファイルができていることを確認する。

＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃

STEP 3　デバッグ環境の構築

概要：
TureSTUDIO のデバッグサーバを借用して ST-LINK 経由で STM32F4 Discovery などにプログラムを書き込み、デバッグができるようにする。

1・デバッグサーバの設定

1-1：[Run]-[External Tools]-[External Tools Configuration] でウィンドウを開き、Main タブをクリック。
1-2：Name に ST-LINK gdb Server などと入力。（表示される名前なのでなんでもいい）
1-3：Location の Browse_File_System ボタンをクリックして TrueSTUDIO 無償版の ST-LINK_gdbserver.exe を選択する。（例えば、"C:¥Program Files¥Atollic¥TrueSTUDIO for STMicroelectronics STM32 Lite 3.0.0¥Servers¥ST-LINK_gdbserver¥ST-LINK_gdbserver.exe"などになる）
1-4：Working Directory の Browse_Workspace をクリックして現在のプロジェクトフォルダを選択。
1-5：Arguments に "-e -d" を入力。（""はいらない）
1-6：Common タブをクリックし、External Tools にチェックを入れる。（これでメニューに出るようになる）

2・デバッグコンフィギュレーションの作成と設定
デバッグをスタートするときのマクロのようなもの。ツールバーの虫アイコンのとなりの▼をクリックすると表示されるのでコレを選んでデバッグを開始する。

2-1：[Run]-[Debug Configurations] で Debug Configurations ウィンドウを開く。

2-2：Zylin Embedded debug (Native) を選択し、画面左上のボタン New launch configuration （小さな書類にプラスのアイコン）をクリックして新しいデバッグコンフィギュレーションを作成。

2-3：これを「オブジェクトファイルをロードしてデバッグを開始する」という設定にする。具体的には下記の作業をする

■コンフィギュレーションに名前をつける。
Name に ″Load and Debug″などと入力する。（自分でわかればなんでもよい）

■Main タブをクリックして下記の設定。
Project の Browse ボタンをクリックして現在のプロジェクトフォルダを選択。
C/C++ Application の Browse ボタンをクリックして現在のプロジェクトフォルダ内の Dubug フォルダにある.elf ファイルを選択。
！！！終了したら Apply ボタンをクリック！！！（重要）

■Debugger タブをクリックして下記を設定。
Debugger で [Embedded GDB] を選択。
GDB debugger の Browse ボタンをクリックして下記を選択。（見つからなければ Sourcery CodeBench がインストールできてない）
″C:¥Program Files¥CodeSourcery¥Sourcery_CodeBench_Lite_for_ARM_EABI¥bin¥arm-none-eabi-gdb.exe″
GDB commad file は何も入れない。
GDB command set は [Standard]
Protocol は [mi]
！！！終了したら Apply ボタンをクリック！！！（重要）

■Commands タブをクリックして下記を設定。
’Initialize’ commands に下記をコピペ。

target remote localhost:61234
load
monitor reset init
thbreak main

’Run’ commands に下記コピペ。

continue

！！！終了したら Apply ボタンをクリック！！！（重要）

■Common タブをクリックして下記を設定。
Display in faverites menu の虫アイコンにチェック
（これでツールバーの虫アイコンの▼をクリックするとコンフィギュレーションが選択できる）

2-4：画面右下の Close ボタンをクリックで終了。

＃＃＃＃＃＃＃＃コレですべての設定終了＃＃＃＃＃＃＃＃＃

（念のため一度 Eclipse を終了、再起動しておく）

＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃＃

STEP 4　テストと使い方

STM32F4 Discovery をパソコンに接続

Ecripse を起動したら[Run]-[External Tools]-[ST-LINK gdb Server]を選択。
（サーバーの名前は STEP3 の1で設定したもの）これは起動後 1 回だけ必要。サーバが起動しているかどうかを知るには、ツールバーの小さな緑の丸に赤いかばんのアイコンにカーソルを乗せると ST-LINK gdb Server(already running..)などと表示される。

STM32F4 Discovery の USB コネクタそばの丸い LED がミドリに変わる。

BuildAll でプロジェクトをビルドする。エラーがあれば直す。
（画面下のメッセージウィンドウの Problems タブを開くと Errors として make:***なんたらというエラーが表示されるが、これは気にしなくともよい。Console タブでエラーがリポートされなくなればビルド成功）

ツールバーの虫アイコンの右どなりの小さな▼をプレスして STEP3 の2で設定したデバッグコンフィギュレーションを選択。選択できないときは[Run]-[Debug Configurations] で Debug Configurations ウィンドウを開き、設定したコンフィギュレーションを選択したうえで、ウィンドウ下の Debug ボタンをクリック。

画面下部にメッセージが流れ、しばらくするとデバッグ画面に切り替わる。このときボードの丸い LED は赤・緑が高速点滅している状態。（エラーダイアログが表示されるときはデバッグサーバが動いていないときが多い）

ここまでくればあとは TureSTUDIO と似た感じでデバッグできる。
デバッガからはリセットできない。リセットはなぜか不安定なので注意。また、リセットは実行ボタンをクリックしてプログラムを走らせないと効かないので注意。

STEP 5　ハード浮動小数点演算の設定

FPU を使う設定。float の関数に double を設定するなどの場合、これを設定しないとエラーになる場合もある（？）
[Propertis]-[C/C++ Build]-[Settings]を選択し、[Tool Settings]をクリック、下記を設定。
■[ARM Sourcery Windows GCC C Linker]から[Librarys] をクリック。表示される画面の[Libraries (-l)]の Add..ボタン（書類に緑のプラスのアイコン）をクリックし 'm' を追加。
■[Target Processor]をクリックし表示される画面で下記を選択。
　　[Float ABI] で [Library with FP (-mfloat-abi=softfp)]を選択。
　　[FPU Type] で [FPv4-SP-D16 (-mfpu=fpv4-sp-d16)] を選択。
■Ok ボタンをクリックして終了。

STEP 6　sscanf と sprintf を使えるようにする

newlib の syscalls.c を改造したものをプロジェクトの src フォルダにコピーする。これだけで上の二つの関数が使えるようになる。ソースコードはこの書類の最後の付録にあるので、テキストエディタで syscalls.c というテキストファイルを作り、これにテキストとしてコピペする。
（このコードは newlib を Nemui さんが改変したものを Hanaokacyan がさらに改変したもの）

STEP 7　Tips

○デバッグ時に必ずしもボードにリセットがかからない。Hard_Fault などの例外が出る場合がある。そのため手動でリセットをかける。手順は下記のとおり
　　プログラムが停止しているかを確認（実行中なら普通にリセットボタンを押すだけ）
　　ボードのリセットボタンを押したままにする
　　デバッグ画面の実行ボタン（右向き三角）をクリック
　　リセットボタンを離す

最初の実行なら main 関数の入り口でとまる。(そのままとまらずに実行し続けるときもある)

○Double abc = 123.56;のようにダブルの変数の定義時に(0以外の)初期値を設定するようなコードは暴走する。ボーレートがおかしくなったり、ハードトラップで停止したりするといった、わけのわからないことが起こる。基本的に変数定義時の初期値設定はやらないほうが安全。

○ボードへのダウンロードがいつまでも終わらないときは、ボードの USB ケーブルを抜く。エラーになって終了する。このときでバックサーバも停止するので、再起動しておく。

○変更したソースはビルド時に自動保存されない。保存してからビルドする癖をつけましょう。

■付録■ sprintf と sscanf を使えるようにするための syscalls.c

```
/*****************************************************************************/
/*                                                                           */
/*        Syscall support functions for newlib console I/O with stdio        */
/*                                                                           */
/*****************************************************************************/

/* $Id: syscalls.c,v 1.4 2008/08/25 16:11:40 cvs Exp $ */
/* 20090508 Nemui Fixed   scanf consideration */
/* 20091220 Nemui Separated Device Independent Definitions */
/* 20101107 Nemui Added error handling in sbrk */
#include <stdlib.h>
#include <string.h>
#include <reent.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>

/* 20120701 Hanaokacyan added */
#define FALSE 0
#define TRUE  1

/* 20120701 Hanaokacyan replaced putch() and getch() by this dummy code */
void putch(char c){

}
 char getch(void){
       return(0x0d);
}
/*--------- end Hanaokacyan modification ------------------------------*/

/* new code for _read_r provided by Alexey Shusharin - Thanks */
_ssize_t _read_r(struct _reent *r, int file, void *ptr, size_t len)
{
  char c;
  int  i;
  unsigned char *p;

  p = (unsigned char*)ptr;
  for (i = 0; i < len; i++)
  {
      /* 20090521Nemui */
      do{
            c = getch();
      }while(c == FALSE);
```

```
      /* 20090521Nemui */

    *p++ = c;
      #ifdef ECHOBACK
            putch(c);
      #endif

    if (c == '\r' && i <= (len - 2)) /* 0x0D */
    {
      *p = '\n';                            /* 0x0A */
        #ifdef ECHOBACK
            putch('\n');                          /* 0x0A */
        #endif
      return i + 2;
    }
  }
  return i;
}

_ssize_t _write_r (
    struct _reent *r,
    int file,
    const void *ptr,
    size_t len)
{
      int i;
      const unsigned char *p;

      p = (const unsigned char*) ptr;

      for (i = 0; i < len; i++) {
            if (*p == '\n' ) putch('\r');
            putch(*p++);
      }

      return len;
}

int _close_r(
    struct _reent *r,
    int file)
{
      return 0;
}

_off_t _lseek_r(
    struct _reent *r,
    int file,
    _off_t ptr,
    int dir)
{
      return (_off_t)0; /*  Always indicate we are at file beginning.   */
}


int _fstat_r(
    struct _reent *r,
    int file,
    struct stat *st)
{
      /*  Always set as character device.                */
```

```
      st->st_mode = S_IFCHR;
            /* assigned to strong type with implicit */
            /* signed/unsigned conversion.  Required by    */
            /* newlib.                          */

      return 0;
}

#ifdef __GNUC__
int isatty(int file); /* avoid warning */
#endif
int isatty(int file)
{
      return 1;
}


void _exit(int n) {
label:  goto label; /* endless loop */
}

int _getpid(int file)
{
      return 1;
}

int _kill(int file)
{
      return 1;
}

/* "malloc clue function" */

      /**** Locally used variables. ****/
extern char end[];              /*  end is set in the linker command   */
                                                /* file and is the end of statica
lly   */
                                                /* allocated data (thus start of
heap).      */

static char *heap_ptr;              /* Points to current end of the heap.     */

/************************ _sbrk_r *************************************/
/*  Support function.  Adjusts end of heap to provide more memory to   */
/* memory allocator. Simple and dumb with no sanity checks.                  */
/*  struct _reent *r    -- re-entrancy structure, used by newlib to    */
/*                support multiple threads of operation.
            */
/*  ptrdiff_t nbytes    -- number of bytes to add.
      */
/*  Returns pointer to start of new heap area.
      */
/*  Note:  This implementation is not thread safe (despite taking a          */
/* _reent structure as a parameter).
            */
/*  Since _s_r is not used in the current implementation, the following      */
/* messages must be suppressed.
                  */

/* Register name faking - works in collusion with the linker.  */
register char * stack_ptr asm ("sp");
```

```
void * _sbrk_r(
    struct _reent *_s_r,
    ptrdiff_t nbytes)
{
      char  *base;            /*  errno should be set to  ENOMEM on error     */

      if (!heap_ptr) {  /*  Initialize if first time through.          */
            heap_ptr = end;
      }
      base = heap_ptr;  /*  Point to end of heap.                              */

      if (heap_ptr + nbytes > stack_ptr)
      {
                  errno = ENOMEM;
                  return (caddr_t) -1;
      }
      heap_ptr += nbytes;     /*  Increase heap.
            */

      return base;            /*  Return pointer to start of new heap area.   */
}



void * _sbrk(ptrdiff_t incr)
{
  char  *base;

/* Initialize if first time through. */

  if (!heap_ptr) heap_ptr = end;

  base = heap_ptr;      /*  Point to end of heap. */

      if (heap_ptr + incr > stack_ptr)
      {
                  errno = ENOMEM;
                  return (caddr_t) -1;
      }

  heap_ptr += incr;     /* Increase heap. */

  return base;          /* Return pointer to start of new heap area. */
}

int _open(const char *path, int flags, ...)
{
  return 1;
}

int _close(int fd)
{
  return 0;
}

int _fstat(int fd, struct stat *st)
{
  st->st_mode = S_IFCHR;
  return 0;
}
```

```
int _isatty(int fd)
{
  return 1;
}


int _lseek(int fd, off_t pos, int whence)
{
  return 0;
}

int _read(int fd, char *buf, size_t cnt)
{
  *buf = getch();

  return 1;
}

int _write(int fd, const char *buf, size_t cnt)
{
  int i;

  for (i = 0; i < cnt; i++)
    putch(buf[i]);

  return cnt;
}

char *__exidx_start;
char *__exidx_end;

/* Override fgets() in newlib with a version that does line editing */
/*
char *fgets(char *s, int bufsize, void *f)
{
  cgets(s, bufsize);
  return s;
}
*/
```