
CodeScape チュートリアル
for 2.1.0 Build 88 BETA



Dreamcast™

はじめに

CodeScape は常にバージョンアップが続けられています。

また SDK のバージョンによっては添付のサンプルプログラムが同一でない可能性もあり、そのためお手元の画面が本書掲載のスクリーンショットと異なる、本書の記述の一部が無効であるなど、本書との相違がある場合もあり得ます。あらかじめご了承ください。

ご不明の点につきましては、弊社テクニカルサポートにお尋ねください。

本書に掲載されている URL は、予告なく変更されることがあります。

商標

- ・ 本書中の固有名詞は、一般に各社の商標または登録商標です。
- ・ CodeScape は、Cross Products Limited の登録商標です。
- ・ Dreamcast は、株式会社セガ・エンタープライゼスの登録商標です。
- ・ MS、Microsoft、MS-DOS、Windows、Windows NT は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。

改版履歴

CodeScape チュートリアル for 2.1.0 Build 88 BETA

1998 年 8 月 31 日 初版発行

Copyright (C) 1998 株式会社セガ・エンタープライゼス

編集・製作 株式会社アスキー AAP 書籍編集部

本書の目的

本書は、多機能 GUI デバッガ“ CodeScape ”導入のための解説書です。

CodeScape 自体は複数プラットフォーム、また SEGA Saturn 向けなどの複数ターゲットも想定していますが、本書では特に Windows 上の KATANA (Dreamcast) 開発環境に絞ったチュートリアルを行います。

本書はあくまで CodeScape 導入までの支援を行うもので、すべての機能の解説を行うものではありません。より深い理解については『CodeScape マニュアル』をご覧ください。

本書で必要とする環境

本書では次のような環境を前提としています。

また、それぞれのハードウェアおよびソフトウェアは適切にセットアップが行われている必要があります。その詳細は、それぞれの機種 (あるいはパッケージ) のマニュアルをご覧ください。

ハードウェア

- ・ Intel CPU を載せた IBM-PC 互換機^{* 1} (Pentium II 233MHz 以上を推奨)
- ・ 64MB 以上の RAM
- ・ SCSI カード (Adaptec AHA-2940U or AHA-2940AU^{* 2})
- ・ SET5

ソフトウェア

- ・ Windows 95 (or WindowsNT4.0 Workstation)
- ・ SET5 SDK
- ・ スクリーンエディタ (秀丸エディタ、WZ エディタなど)
- ・ nmake^{* 3}
- ・ ASPI マネージャ^{* 4}

* 1

PC98-NX シリーズは使用できません。

* 2

アダプテックジャパンの Web ページ上に、動作確認機種の一覧が掲載されています (<http://www.adaptec.co.jp/>)。ご確認ください。

* 3

Microsoft の make です。VisualC ++ などに添付されています。

* 4

バージョン 4.00 以降が必要です。

contents

目次

はじめに	2
------------	---

第1章

プログラム作成の開始

7

1.1 CodeScape の設定	7
1.1.1 エディタの指定	7
1.1.2 プロジェクト設定	8
1.2 Make	11
1.3 SET5 へのロードと実行	12

第2章

CodeScape のデバッグ機能

15

2.1 サンプルプログラムについて	15
2.2 デバッグの実際	15
2.2.1 プロジェクトの設定	16
2.2.2 エラー発生からソースエディット	16
2.2.3 Breakpointの設定	20
2.2.4 値の内容チェック	27
2.2.5 プログラムの流れを調べる	30
2.2.6 トレース	33
2.2.7 ソースリスト中の、特定の範囲をチェックする	40

第3章

トラブルシューティング

45

3.1	SET5	45
3.1.1	プログラムをロードしても動作しない(ケース1).....	45
3.1.2	プログラムをロードしても動作しない(ケース2).....	46
3.1.3	ホストマシンとの接続について	46
3.2	CodeScape	47
3.2.1	バージョンを上げたら起動しなくなった	47
3.2.2	Autoexec.batにset_kt.bat が記述できない	47
3.3	プログラム作成に関わる注意点	48
3.3.1	グラフィックデータの利用について	48
3.3.2	その他	50
3.3.3	CodeScape 開発環境に併用すると便利なツール類	50

付録 A

参考文献

51

付録 B

参考 URL

53

第 1 章

プログラム作成の開始

この章では、CodeScape を使用するための設定から、make されたオブジェクトの実行までを解説します。

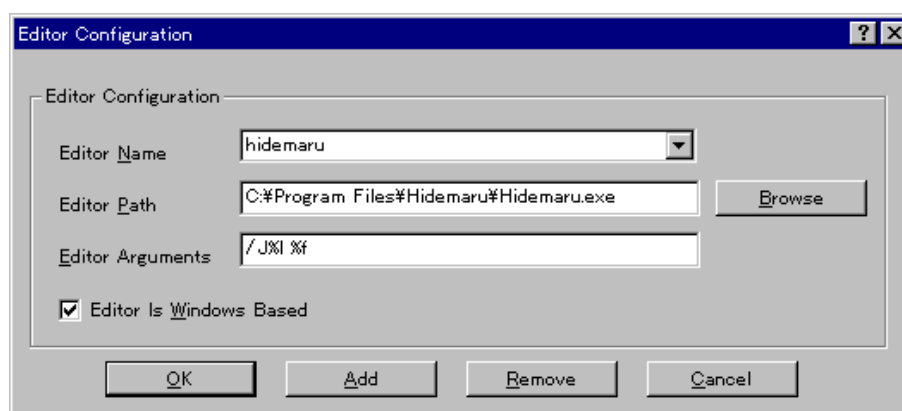
1.1 CodeScape の設定

CodeScape で開発を始める時には、いくつかの設定を行わなければなりません。
ここでは、その手順を説明します。

1.1.1 エディタの指定

まず使用するスクリーンエディタの指定を行います。

[Project] メニューの [Setup Editor] をクリックすると、次のようなダイアログボックスが表示されます (ここでは、すでに設定を行った画面を掲載しています)。



[Editor Name]

現在指定されているエディタ名称です。あらかじめいくつかのものは登録されていますので、それらの中から選択することで使用可能になります。

既存のものを使用する場合、選択してプロジェクト設定に進んでください。

登録されていないエディタを使用する場合には、以下の項目を新たに設定して登録する必要があります。

[Editor Path]

そのエディタプログラムを、full-path で指定します。

[Editor Arguments]

CodeScape がエディタを起動する時、エディタに渡されるコマンド列です。

ここでは /J%i %f を渡していますが、%i は CodeScape によって行番号に置換されるものです (また /j は、秀丸エディタの場合行番号ジャンプです*¹)。また %f はファイル名に置換されます。これを設定しておくことによって、エラー発生時にタグジャンプすることが可能です。

[Editor Is Windows Based]

指定したエディタが Windows アプリケーションである場合にチェックします。

以上を記述し、[Add] ボタンをクリックします。

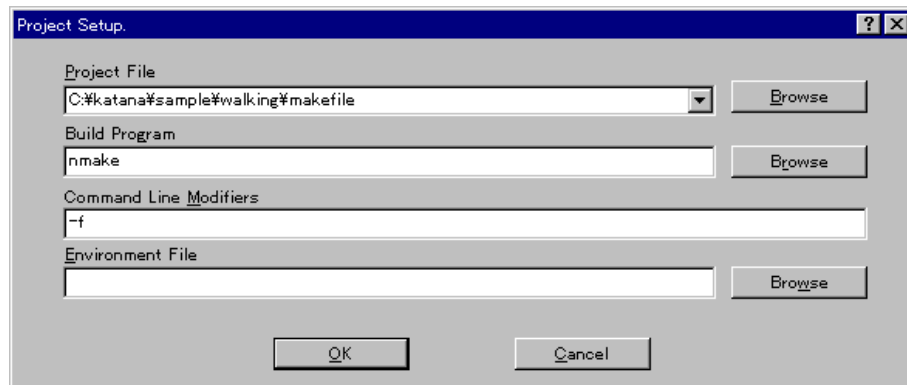
これ以降は、エディタ名称を指定するだけで、そのエディタが使用可能になります。

1.1.2 プロジェクト設定

続いてプロジェクト設定を行います。

ここで言う「プロジェクト設定」とは make 環境を設定するものですが、この設定は新規プロジェクトを始める度に設定します。

[Project] メニューの [Project Setup] をクリックすると、次のようなダイアログボックスが表示されます。



[Project File]

makefile ファイル名を (full-path で) 指定します。

ここでは、次章で使用するサンプルプログラムの makefile を指定しています。

以降は、通常ここ以外のフィールドを変更することはありません。

* 1

ただし、Build65 以前の CodeScape では秀丸で行番号ジャンプを行わせることができません。詳細はトラブルシューティングを参照ください。

[Build Program]

どの make ツールを使用するのかを指定します (勿論、上で指定した makefile を理解できる make でなければなりません)。ここでは nmake を指定しています。ファイル名に path を含めていませんが、path が通っていないディレクトリに make が置かれている場合には、path を付加して記述する必要があります (例 : `c:\program files\devstudio\vc\bin\make`)。

[Command Line Modifiers]

make に渡すパラメータを指定します。これが複数必要な場合、その中で特に最後のパラメータには、上の Project File ファイル名を make に渡すためのパラメータを指定します。nmake を使用しますので、ここでは `-f` を指定しています。

まとめると、ここまでで

```
nmake -f C:\katana\Shinobi\sample\walking\makefile
```

という make 指定を行ったことになります。

[Environment File]

make が使用する環境変数ファイルを指定します。

SET5 SDK、及び nmake を正しくインストールしている場合、ここには特に記述する必要がありません^{* 2}。

また、次の方法で環境ファイルを作成して、そのファイル名を指定することもできます。

環境ファイルの作成方法

環境変数の設定を行うために、DOS プロンプト上から `C:\KATANA\BIN\SET_KT.BAT` (デフォルトインストール時) を実行してください。必要な環境変数がすべて設定されますので、この状態から `SET > Codescape.env` + [Enter] を実行します。こうして作成されたファイルを環境ファイルとして指定します。

* 2

具体的には、`set_kt.bat` と `vcvars32.bat` (VisualC++ の `nmake.exe` を使用する場合) が、`autoexec.bat` などの中で事前に実行されている状態を指します。

環境ファイル例

```
TMP=C:\WINDOWS\TEMP
TEMP=C:\WINDOWS\TEMP
PROMPT= $ p $ g
winbootdir=C:\WINDOWS
COMSPEC=C:\COMMAND.COM
PATH=C:\KATANA\SHC\BIN;C:\KATANA\BIN;C:\FW;C:\BIN;C:\PROGRAM~1\DEVSTUDIO\
VC\BIN;C:\WINDOWS;C:\WINDOWS\COMMAND
windir=C:\WINDOWS
SHC_VER=50010
KATANA_SDK_VER=0502
KATANA_VER=505
SHINOBI_VER=0502
KATANA_SDK_DIR=C:\KATANA
SHC_DIR=C:\KATANA\SHC
SHINOBI_DIR=C:\KATANA\SHINOBI
SHC_BIN=C:\KATANA\SHC\BIN
SHC_INC=C:\KATANA\SHC\INCLUDE,C:\KATANA\SHINOBI\INCLUDE
SHC_LIB=C:\KATANA\SHC\BIN
SHC_TMP=C:\WINDOWS\TEMP
HLNK_LIBRARY1=C:\KATANA\SHC\LIB\SH4NLFZZ.LIB
HLNK_LIBRARY2=C:\KATANA\SHINOBI\LIB\SHINOBI.LIB
HLNK_LIBRARY3=C:\KATANA\SHINOBI\LIB\NINJA.LIB
SHCPP_VER=50010
SHCPP_BIN=C:\KATANA\SHC\BIN
SHCPP_INC=C:\KATANA\SHC\INCLUDE,C:\KATANA\SHINOBI\INCLUDE
SHCPP_LIB=C:\KATANA\SHC\BIN
SHCPP_TMP=C:\WINDOWS\TEMP
SET_KT_VER=05101
PATH_HED=C:\KATANA
```

注意

CodeScape Build92 以前では、環境ファイル中にエントリ CMDLINE が存在すると問題が起きます。エントリ CMDLINE を削除してください。

1.2 Make

make は、CodeScape 上からは次のいずれかの手順で行います。

- ・ [Project] メニューの [Make] をクリック
- ・ [Ctrl] + [M] キー
- ・ [Project Build] ウィンドウを右クリックして、[Make] をクリック

前項までの設定が行われている場合これで make が行われ、その結果が [Project Build] ウィンドウに表示されます。

make が終了すると、たとえば次のような表示が出ます。



```

C:\Katana\shinobi\sample\walking>echo input C:\KATANA\SHINOBI\lib\sg_ini.obj
C:\Katana\shinobi\sample\walking>echo input C:\KATANA\SHINOBI\lib\ap.obj
C:\Katana\shinobi\sample\walking>echo input C:\KATANA\SHINOBI\lib\zero.obj
C:\Katana\shinobi\sample\walking>echo input sinit.obj
C:\Katana\shinobi\sample\walking>echo input main.obj
C:\Katana\shinobi\sample\walking>echo input pattern.obj
C:\KATANA\SHINOBI\lib\link.exe -sub=link.sub
H SERIES LINKAGE EDITOR Ver. 6.01'98.05.20
Copyright (C) Hitachi, Ltd.1989,1996
Copyright (C) HITACHI MICROCOMPUTER SYSTEM (TD. 1990,1996
Licensed Material of Hitachi, Ltd.

: #
: debug
: #
: output walking.elt
: input C:\KATANA\SHINOBI\lib\ehf1.obj
: input C:\KATANA\SHINOBI\lib\ehf2.obj
: input C:\KATANA\SHINOBI\lib\systemid.obj
: input C:\KATANA\SHINOBI\lib\roc.obj
: input C:\KATANA\SHINOBI\lib\sg_sec.obj
: input C:\KATANA\SHINOBI\lib\sg_areus.obj
: input C:\KATANA\SHINOBI\lib\sg_arec.obj
: input C:\KATANA\SHINOBI\lib\sg_are00.obj
: input C:\KATANA\SHINOBI\lib\sg_are01.obj
: input C:\KATANA\SHINOBI\lib\sg_are02.obj
: input C:\KATANA\SHINOBI\lib\sg_are03.obj
: input C:\KATANA\SHINOBI\lib\sg_are04.obj
: input C:\KATANA\SHINOBI\lib\sg_ini.obj
: input C:\KATANA\SHINOBI\lib\ap.obj
: input C:\KATANA\SHINOBI\lib\zero.obj
: input sinit.obj
: input main.obj
: input pattern.obj
: print walking.map
: form a
: entry SG_SEC
: start (P16C008000),D63LH(6C010000)
: exit

LINKAGE EDITOR COMPLETED
*****
Completed.
*****
Build Complete, Error Code: 00000000

```

エラーが発生した場合、前項で設定されたエディタが自動的に起動され、該当するソースの該当行が表示されます*³。

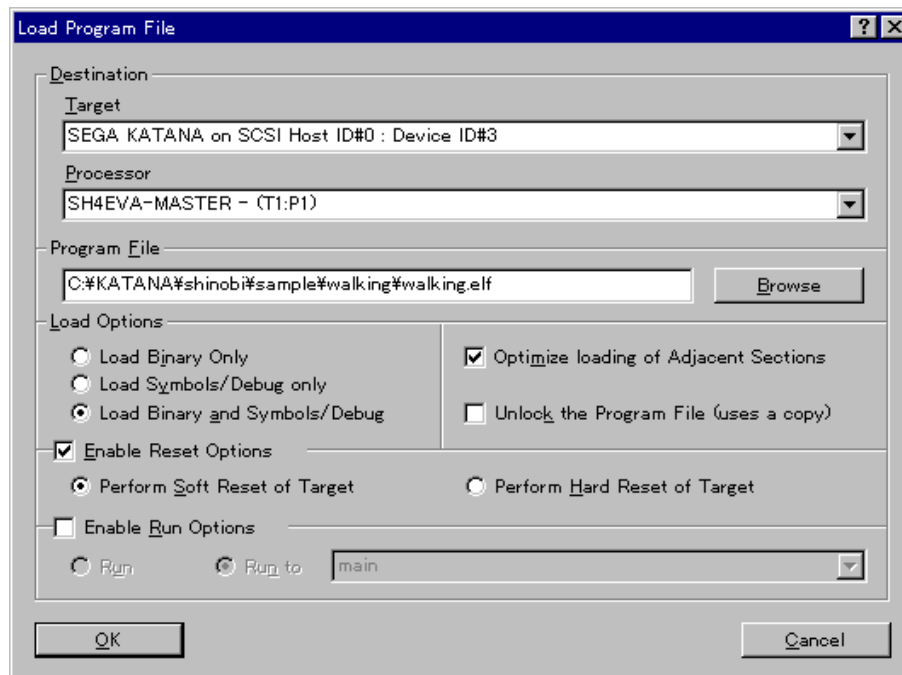
* 3

ジャンプ機能を持たないエディタを使用した場合、また Editor argument でその機能を指定していない場合、該当行へのタグジャンプは行えません。

1.3 SET5 へのロードと実行

前項で作成された実行ファイル (.elf ファイル) を SET5 にロードするには、次の手順で行います。

[File] メニューの [Load Program File] をクリックすると次のようなダイアログボックスが表示されます* 4。



[Target] [Processor]

転送先です。

SET5 を複数台接続している場合以外は、通常変更することはありません。

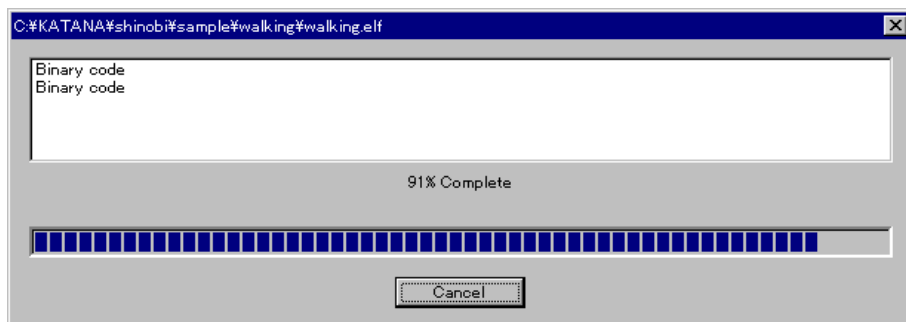
* 4

ショートカットキーは [Ctrl] + [Shift] + [C] です。

[Program File]

前項の make で作成された実行ファイル (.elf ファイル) を指定します。

以上を設定し、[OK] ボタンをクリックすると、下のようなプログレス表示が行われ、実行ファイル (.elf ファイル) が SET5 に読み込まれます。



このように一度ファイル名を指定して読み込んでおけば、以降は make 終了ごとに次のようなダイアログボックスが表示されますので、[はい] ボタンをクリックするだけで実行ファイルが読み込まれます。



読み込まれた後は、次のいずれかの手順で SET5 上で実行できます。

- ・ [Debug] メニューの [Execution] をクリックし、[Run] をクリック
- ・ [Debug] ツールバーの [Run] ボタンをクリック
- ・ [F9] キー * 5
- ・ [Editor] 以外の適当なリージョンを右クリックし、[Execution] をクリックし、[Run] をクリック

* 5

[F9] キーは、Run と Stop のトグルになっています。

第 2 章

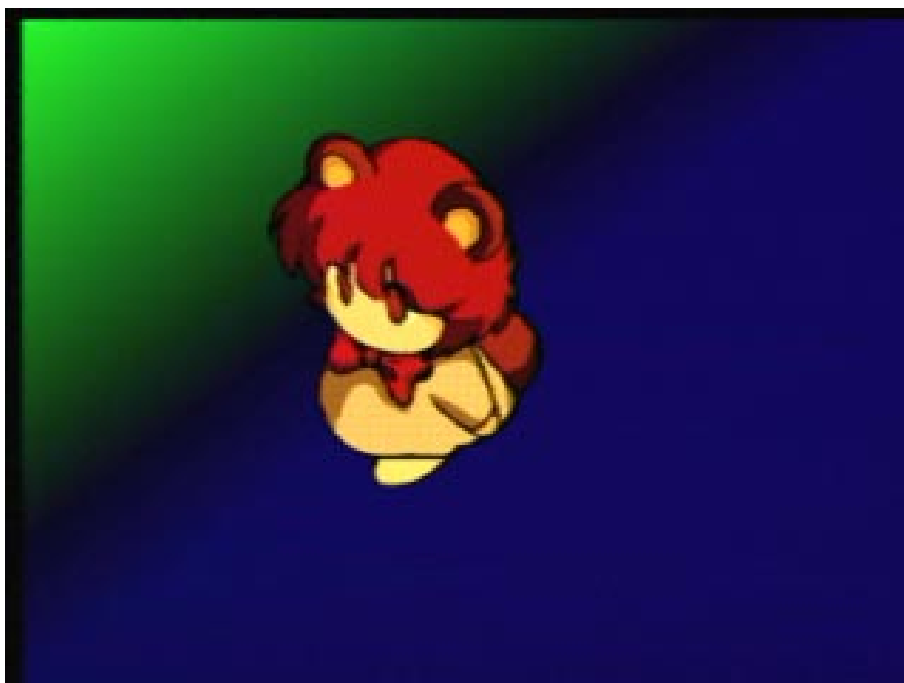
CodeScape の
デバッグ機能

ここではひとつのプログラムを例に取り、CodeScape のデバッグ機能を使ってその動作を追いかけてみます。

2.1 サンプルプログラムについて

サンプルプログラムとして、SDK にも添付されている walking を使用します^{* 1}。

walking は、動作としては以下の画面のようになります。キャラクターが画面上を歩きまわるものです。



^{* 1}

walking は、デフォルトでは c:\¥katana¥Shinobi¥sample¥walking にインストールされています。

2.2 デバッグの実際

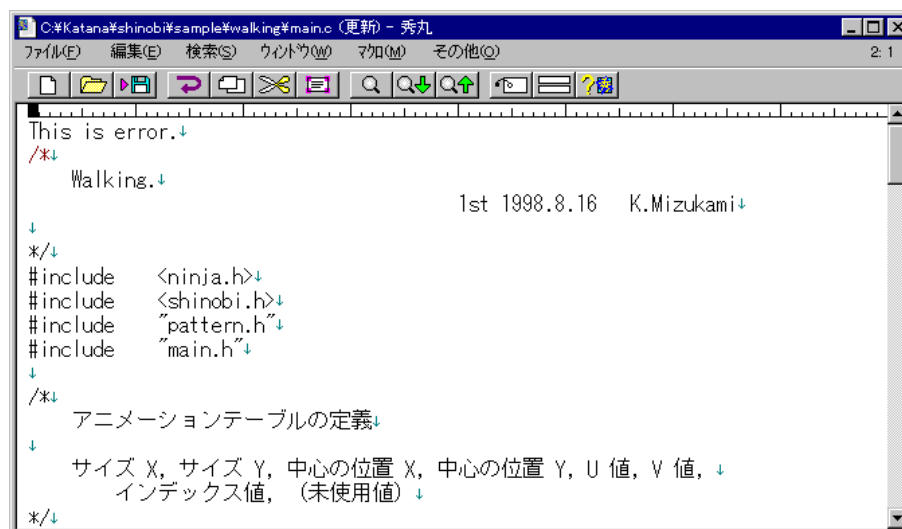
2.2.1 プロジェクトの設定

今回使用する walking にはセッションファイルが添付されています。

[File]メニューの[Session Open]で、C:\katana\shinobi\sample\walking.ssn を選択します。

2.2.2 エラー発生からソースエディット

まず最初は、意図的にコンパイルエラーになるようなものを仕込んでみましょう。



The screenshot shows a text editor window titled "C:\Katana\shinobi\sample\walking\main.c (更新) - 秀丸". The menu bar includes "ファイル(F)", "編集(E)", "検索(S)", "ウィンドウ(W)", "マクロ(M)", and "その他(O)". The toolbar contains icons for file operations and editing. The text area contains the following code:

```
This is error.↓
/*↓
    Walking.↓
                                1st 1998.8.16    K.Mizukami↓
↓
*/↓
#include    <ninja.h>↓
#include    <shinobi.h>↓
#include    "pattern.h"↓
#include    "main.h"↓
↓
/*↓
    アニメーションテーブルの定義↓
↓
    サイズ X, サイズ Y, 中心の位置 X, 中心の位置 Y, U 値, V 値, ↓
    インデックス値, (未使用値) ↓
*/↓
```

ファイルの先頭に、このように不要な文字列を入れてみました。
これを make すると次のようになります。

```

Starting Build nmake -f maketile

Microsoft (R) Program Maintenance Utility Version 1.62.7022
Copyright (C) Microsoft Corp 1988-1997. All rights reserved.

*****
Start Make
*****
C:\VKATANA\SHC\BIN\shc.exe main.c -objectfile=main.obj -listfile=main.lst -sub=shc.sub
SH SERIES C Compiler Ver. 5.0(Relase28)
Copyright (C) 1992,1996 Hitachi,Ltd.,Hitachi Software Engineering Co.,Ltd.
Licensed Material of Hitachi,Ltd.,Hitachi Software Engineering Co.,Ltd.

*** Analyzing Source Program (p) ***
*** Analyzing Source Program (f) ***
*** Generating Code (f) ***
main.c(1) : 2500 (E) Illegal token "is"
C:\VKATANA\SHINOBI\INCLUDE\NinjaStr.h(18) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\NinjaStr.h(19) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\NinjaStr.h(20) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\NinjaStr.h(21) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\NinjaStr.h(22) : 2141 (E) Struct has no member name
C:\VKATANA\SHINOBI\INCLUDE\NinjaStr.h(27) : 2105 (E) Incomplete tag used in declaration
C:\VKATANA\SHINOBI\INCLUDE\Ysg_pad.h(155) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\Ysg_pad.h(156) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\Ysg_pad.h(157) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\Ysg_gd.h(194) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\Ysg_gd.h(195) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\Ysg_gd.h(219) : 2500 (E) Illegal token "UInt8"
C:\VKATANA\SHINOBI\INCLUDE\Ysg_gd.h(220) : 2500 (E) Illegal token "UInt8"

```

このようにエラーになりました。

続いてこのエラーを取り除きましょう。

[Editor Configuration] ウィンドウで指定したエディタが自動的に起動します* 2。

問題のエラーは、不要な文字列が入っていることが原因ですのでこれを削除します。

```

C:\Katana\shinobi\sample\walking\main.c (更新) - 秀丸
ファイル(F) 編集(E) 検索(S) ウィンドウ(W) マクロ(M) その他(O) 1: 1

/*
Walking.
1st 1998.8.16 K.Mizukami

*/
#include <ninja.h>
#include <shinobi.h>
#include "pattern.h"
#include "main.h"

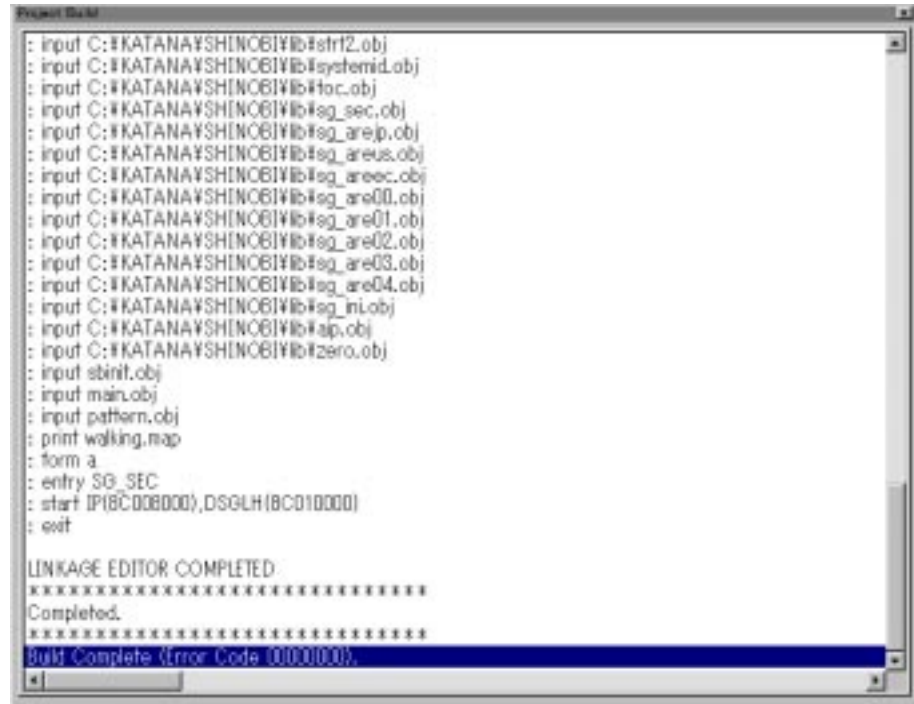
/*
アニメーションテーブルの定義
↓
サイズ X, サイズ Y, 中心の位置 X, 中心の位置 Y, U 値, V 値,
インデックス値, (未使用値)
*/
NJS_TEXANIM anim[] = {

```

* 2

この時、指定したエディタがタグジャンプ機能を持っていて、かつ [Editor Configuration] ウィンドウの Editor Argument でその機能が指定されている場合、エラー行にカーソルが移動します。

変更後セーブし、再度 [Project Build] ウィンドウから make します* 3。

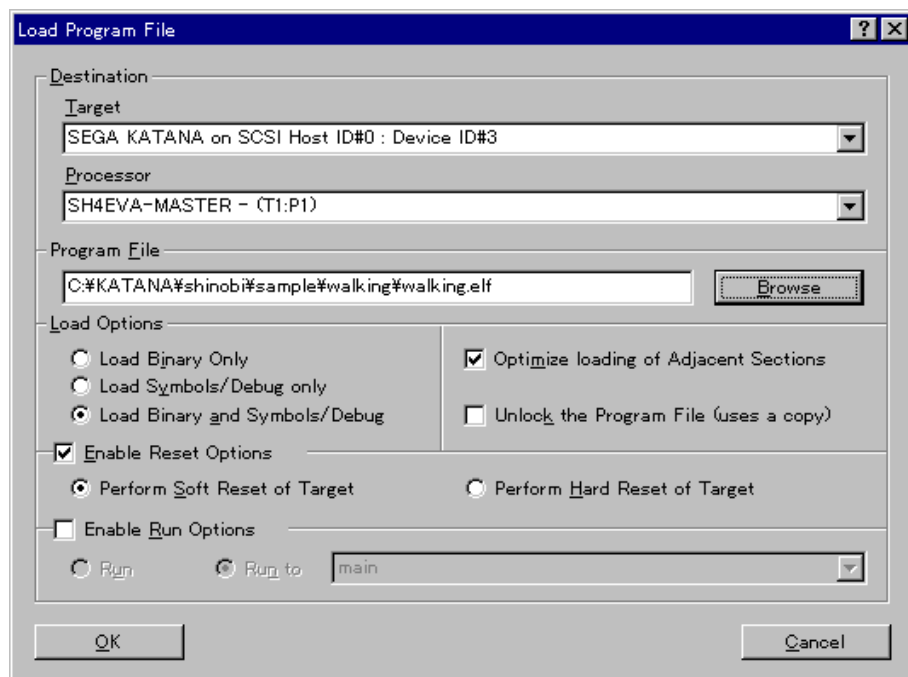


エラーがないことを確認したら、続いてこの実行ファイルをロードします。

[File] メニューの [Load Program File] をクリックすると、次のようなダイアログが表示されます* 4。

* 3
make のショートカットキーは [Ctrl] + [M] です。

* 4
ショートカットキーは [Ctrl] + [Shift] + [C] です。



一度ファイル名を指定して読み込んでおけば、以降は make 終了ごとに次のようなダイアログボックスが表示されますので、[はい] をクリックするだけで実行ファイルが自動的に再読み込みされます。



うまくコンパイルが終了していれば、次のいずれかの手順で SET5 上で実行できます。

- ・ [Debug] メニューの [Execution] をクリックし、[Run] をクリック
- ・ [Debug] ツールバーから [Run] ボタンをクリック
- ・ [F9] キー * 5
- ・ [Editor] 以外の適当なリージョンを右クリックし、[Execution] クリックして [Run] をクリック

以上が、基本的な修正から再 make への流れです。

* 5

[F9] キーは、Run と Stop のトグルになっています。

2.2.3 Breakpoint の設定

続いて Breakpoint の設定を行ってみましょう。

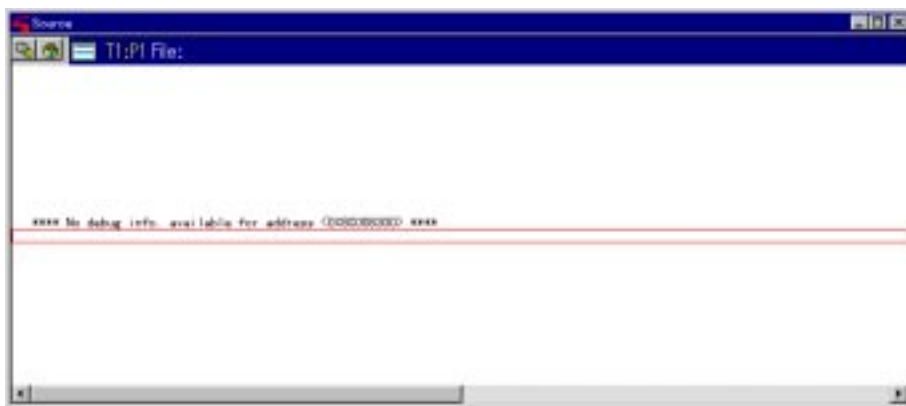
print では、実際のメインコードは test.c 中の njUserMain(); ですので、その中に Breakpoint を設定してみます。以下にその手順を示します。

[Window] メニューの [New Window] をクリックして新しいリージョン (ウィンドウ) を開きます。



このように、未定義のリージョン (No type) が開きます。

今は Breakpoint を設定するためにソースリストを表示させたいので、[Region Combo] ツールバーの [Source] をクリックします。



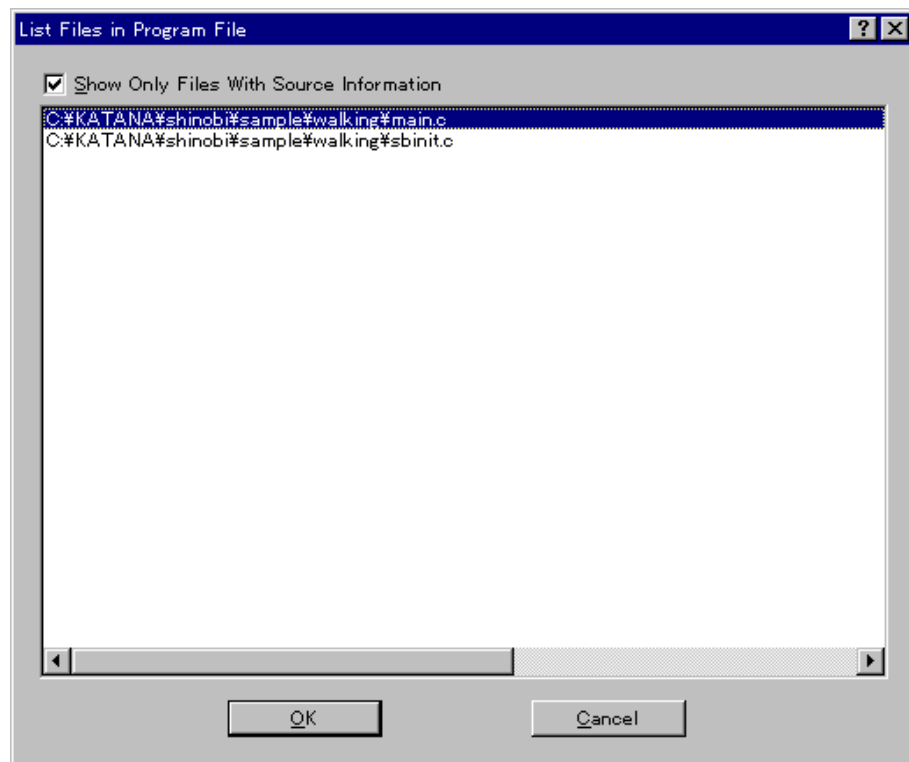
このリージョンのタイトルバーが [Source] に変わり、この [No type] リージョンは [Source] リージョンに設定されました*⁶。

* 6

本書では以降この一連の手順を、単に「～リージョンを開く」と呼びます。

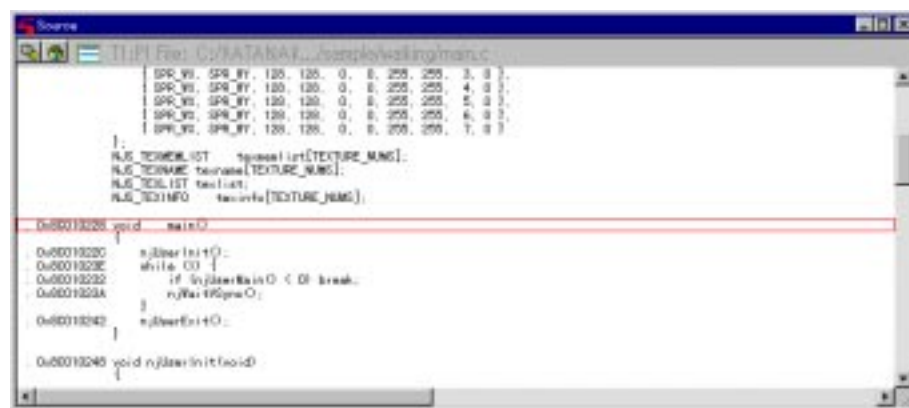
続いて、この [Source] リージョンで右クリックし、[Goto Source File] をクリックします。

すると次のようなウィンドウが開き、このプログラムで使用されているソースファイルの一覧がリストされます。



この中から開きたいソースファイルを選択します。

今開きたいソースは main.c ですから、この中から C:\katana\Shinobi\sample\walking\main.c を選択します。

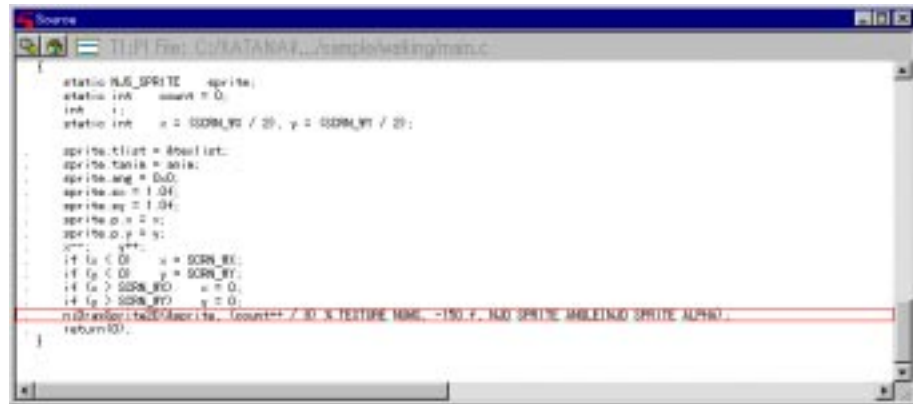


これで [Source] リージョンに main.c が表示されました。

ではこの中に Breakpoint を設定してみましょう。

特定の行に Breakpoint を設定する

画面上にスプライトを表示している行 (99 行目) に Breakpoint を設定してみます。スクロールダウンし、その行をクリックします。



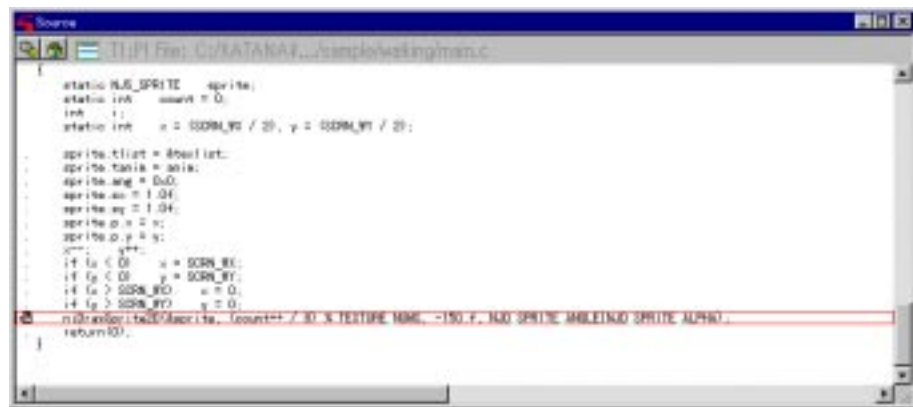
この状態で、次のいずれかの方法で設定できます。

- ・ [Debug] メニューの [Breakpoints] をクリックし、[Toggle Breakpoint] をクリック
- ・ [Breakpoint] ツールバーの [Toggle Breakpoint] ボタンをクリック
- ・ [F5] キー

または次の方法でも可能です。この場合、指定したい行をクリックする必要はありません (右クリックした時に、その時のカーソルの位置に設定されます)

- ・ マウスカーソルをその行で右クリックし、[Breakpoints] をクリックして [Toggle Breakpoint] をクリック

すると、行頭に Breakpoint が設定されたことを示すマークがつきます* 7。

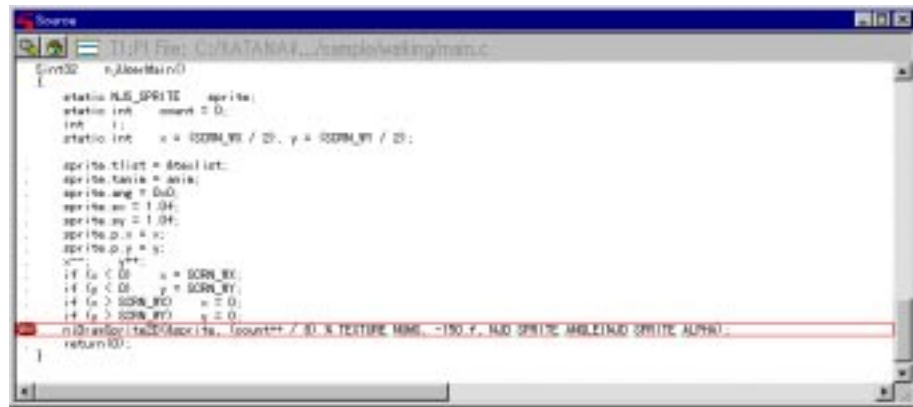


これでこの行に Breakpoint が設定されました。

* 7

これらはトグルですので、同じ行でこれらの操作を行うことでブレークポイントの解除が行えます。

それでは [F9] キーを押して実行してみましょう。



行頭のマークが STOP に変わり、また [Target] ウィンドウも



と表示され、プロセッサがここで停止したことが分かります* 8。

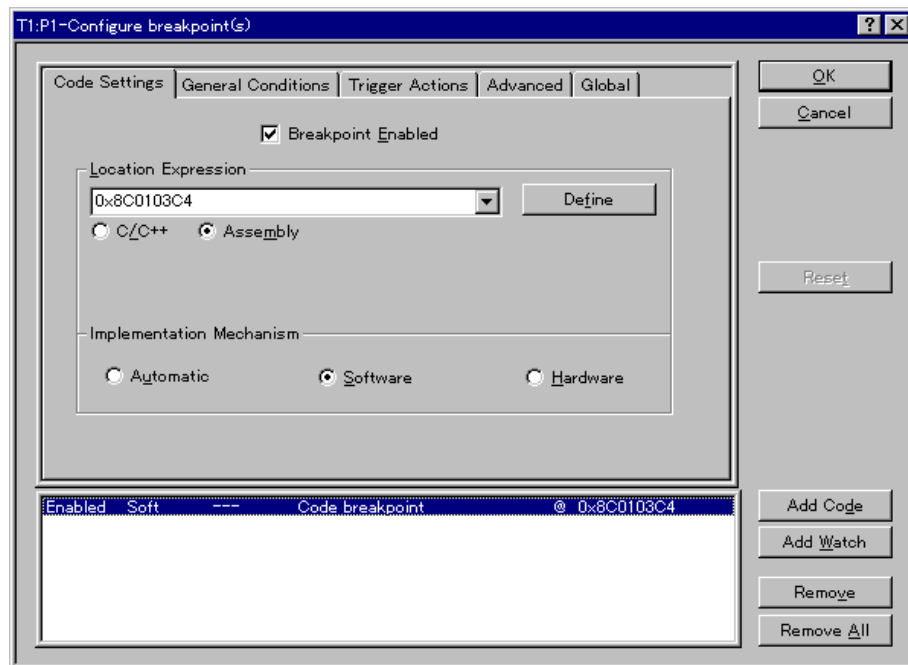
特定変数へのアクセスで停止

続いて、特定の変数にアクセスされた時に停止するような設定 (Watch access breakpoint) を行ってみましょう。

- ・ [Debug] メニューの [Breakpoints] をクリックし、[Configure Breakpoint (s) ...] をクリック
- ・ [Breakpoint] ツールバーの [Configure breakpoint (s)] ボタンをクリック
- ・ [Ctrl] + [F5] キー
- ・ [Editor] 以外のリージョンを右クリックし、[Breakpoints] をクリックして [Configure Breakpoint (s)] をクリック

* 8

ここで [Register] リージョンを開くと、停止アドレス及び現在のレジスタの値などが確認できます。

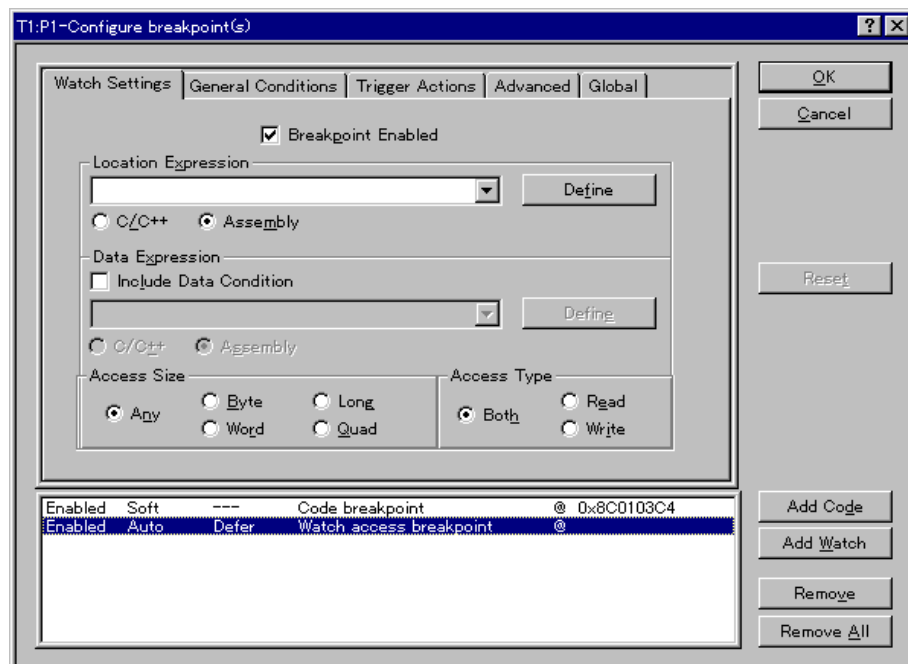


このようなダイアログが表示されました。

先ほど設定した Breakpoint が有効なままですので、その情報が表示されています。

ここで [Add Watch] ボタンをクリックします。

[Code Setting] タブが [Watch Settings] タブに変わります。



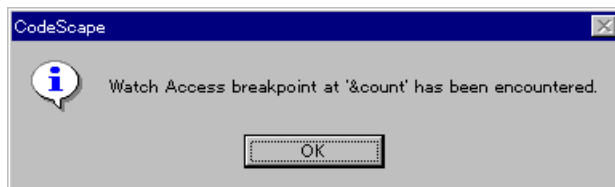
それでは、変数 count のアクセスに対して停止するようにしてみましょう。

[Local Expression] に「 & count 」と入れ、またこれは C/C + + の表記方法ですので、下のオプションボックスで [C/C + +] を選択します。そして [OK] をクリックします。

これで、以降は変数 count にアクセスが入るごとに停止するようになります* 9。

それでは [F9] キーを押して再実行してみましょう。

すぐに次のようなダイアログボックスが表示され、変数 count がアクセスされたことが分かります。



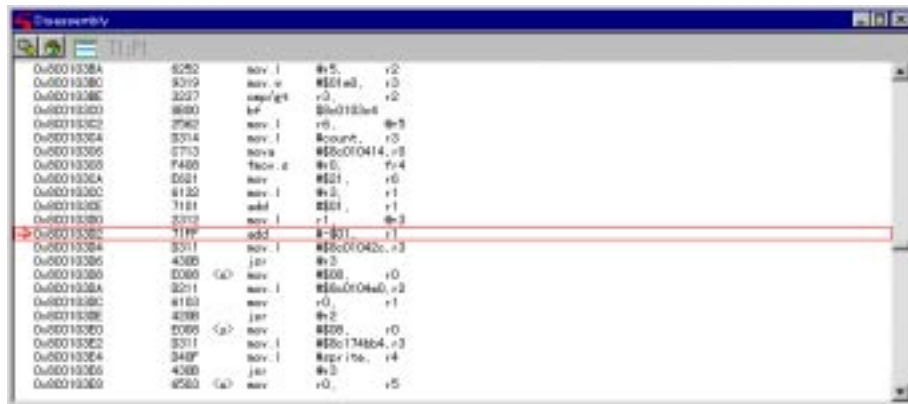
[Source] リージョンを見ると、



ソース中に変数 count が出てきているあたりで停止しています。

ただし Watch access breakpoint には注意しなければいけない点があります。

ここで [Disassemble] リージョンを開いてみます。



* 9

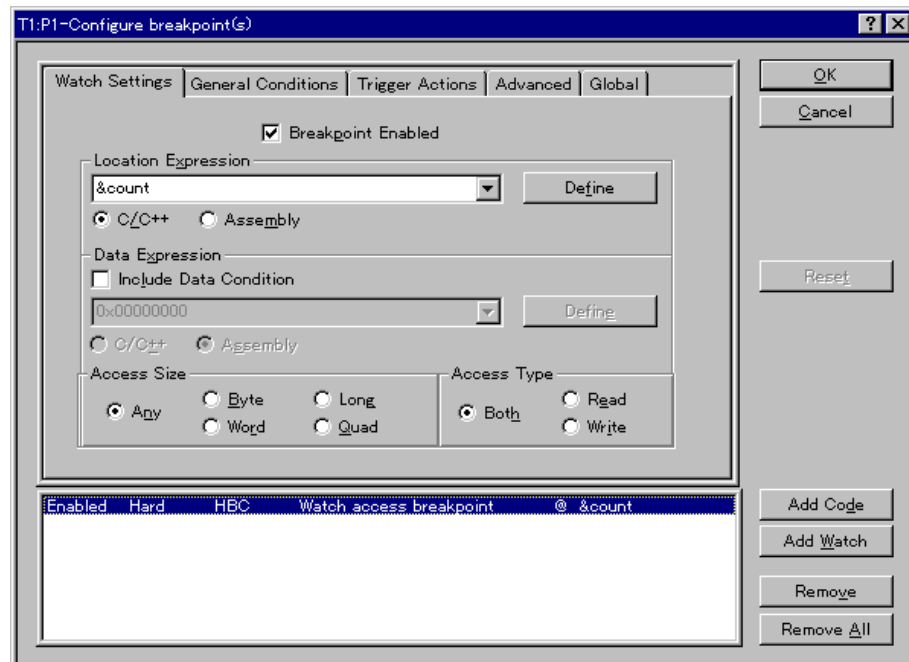
これを読み込みまたは書き込みのいずれかのみに対して行いたい場合には、下の [Access Type] オプションボタンで [Both] ではなく [Read] または [Write] を選択します。

実際に変数 count にアクセスが入ったのは、現在の PC 値より 2 行前のようです。

このように、Watch access breakpoint がトリガされる時 PC はパイプライン中で、すでに数命令先を処理しています (よってズレが生じてしまいます)。

このことに気を付けてください。

この Watch access breakpoint の解除は、上の [Configure breakpoint (s)] ダイアログで、下にリストされている Breakpoint の一覧から解除したいものをクリックして選択し、[Remove] ボタンをクリックすることで行います。



これで Watch access breakpoint が解除されました。

Breakpoint には、ここで示した以外に、色々な条件を設定しての Breakpoint 処理が可能です。詳細は『CodeScape マニュアル』をご覧ください。

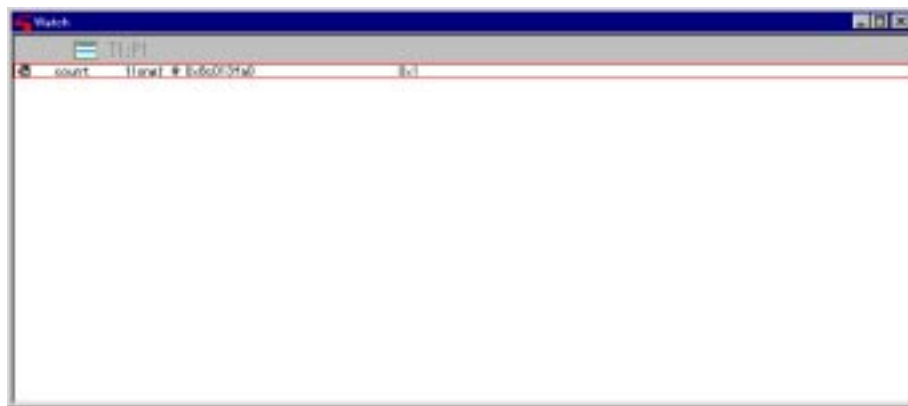
2.2.4 値の内容チェック

実行中の変数の値を見てみましょう。

[Region Combo] ツールバーで新しいウィンドウを開き、[Watch] リージョンに設定します (開き方の詳細は、前項で [Source] リージョンを開いた方法と同じです)。



では、ここでのカウンターである変数 `count` (`int count`) の値を見ることにしましょう。右クリックして [Insert] をクリックします。カーソルが点滅しますので、`count` と入力します。

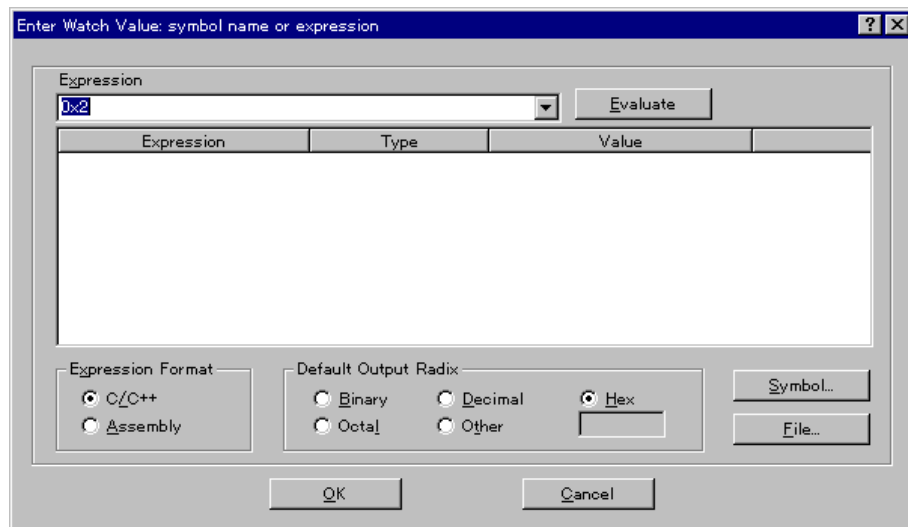


変数 `count` の、現在の値が表示されました。

先ほどの Breakpoint の設定を解除していない場合、[F9] キーを押すごとに Breakpoint で停止しますので、この値が1ずつカウントアップしていくことを確認してください。

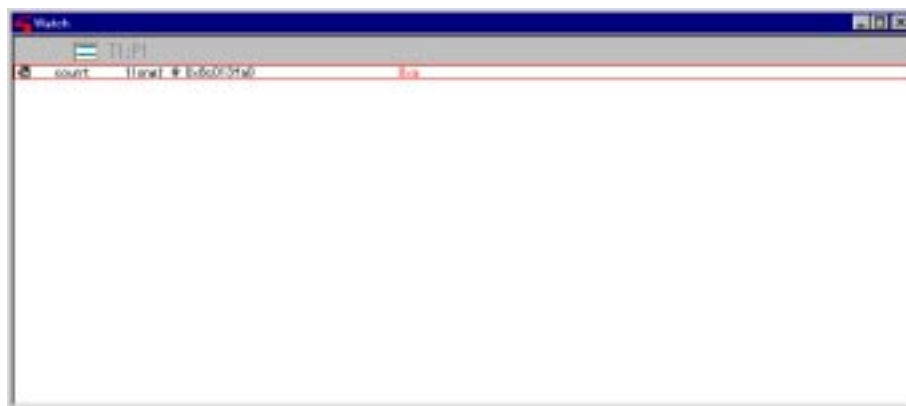
続いてこの変数の値を変更してみましょう。

[Watch] リージョンで変数 `count` を左クリックで選択し、右クリックします。そして [Edit Watch Value] をクリックすると、次のようなウィンドウが表示されます。



この [Expression] に直接値を入力することで変数の内容を変更することができます^{* 10}。

値を設定して [OK] ボタンをクリックするとこのウィンドウが閉じ、[Watch] リージョンに新しい値が入っているはずです。確認してください。



この状態で再実行を行うと、新しい値で表示が始まります。

再実行は、次のいずれかの操作で行います。

- [Debug] メニューの [Execution] をクリックし、[Run] をクリック
- [Debug] ツールバーの [Run] ボタンをクリック
- [F9] キー^{* 11}
- [Editor] 以外の適当なリージョンを右クリックし、[Execution] をクリックして [Run] をクリック

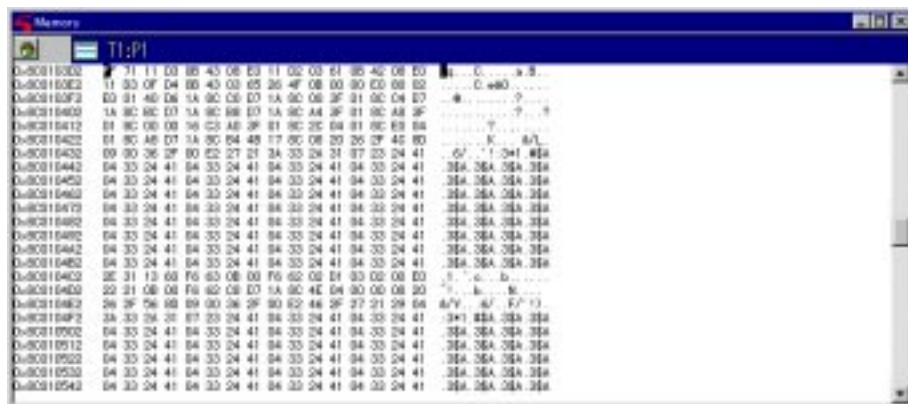
^{* 10}

ここには式を入れることも可能です。式を入れて [Evaluate] をクリックしてください。式が展開され、変数にその値が代入されます。

^{* 11}

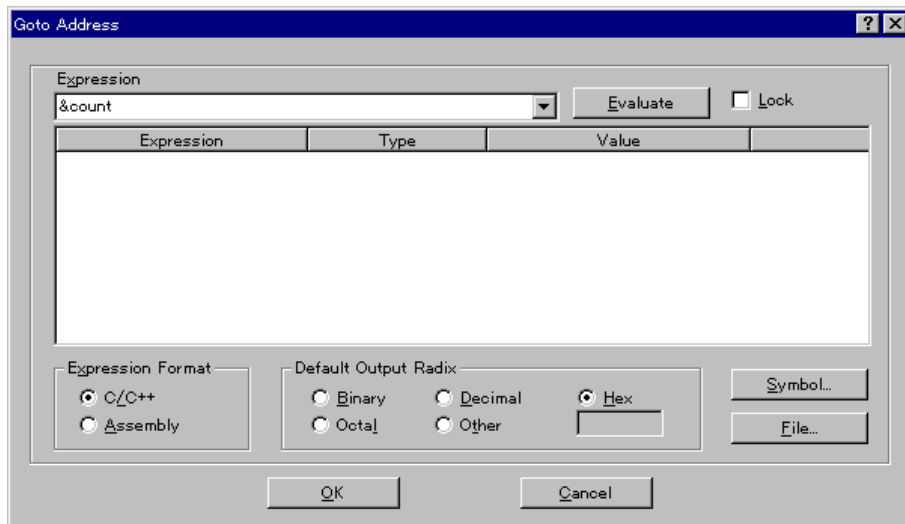
[F9] キーは、Run と Stop のトグルになっています。

ちなみに [Memory] リージョンから変数の値を見ることも可能です。
配列など、連続したアドレスの値を見るにはこちらが便利な場合もあるでしょう。
[Memory] リージョンを開きます。



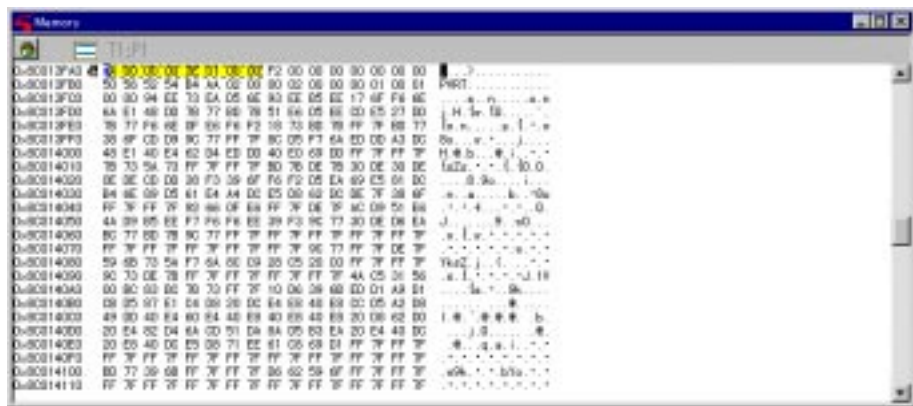
開いた直後の状態では、現在のプログラムカウンタ（ PC ）値を先頭とする表示（メモリダンプ）を行っています。

[Memory] リージョンで右クリックし、[Goto Address...] をクリックします。



ここで [Expression] に変数名を入れます。但し先頭にシンボル名であることを示す「&」を付けることを忘れないでください。

[OK] ボタンをクリックすると、変数 count の、メモリ上のアドレス（変数 count の内容が格納されているアドレス）を先頭とする表示に変わります。



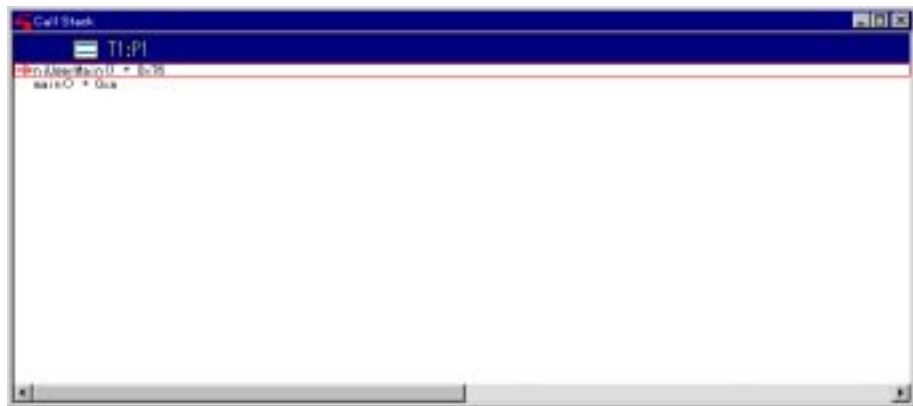
この先頭アドレス (0x8C013FA0) が、[Watch] リージョンで見た変数 count のアドレスと同じであることを確認してください。

変数 count の型は int つまり 4 バイト長ですから、このメモリダンプで先頭からの 4 バイト、ここでは“ 0A 00 00 00 ”がその値であることが分かります。

2.2.5 プログラムの流れを調べる

デバッグ上、プログラムがどのような流れで実行されてきているのか調べたい時があります。このような時には [Call Stack] リージョンが役に立ちます。

[Call Stack] リージョンを開きます。



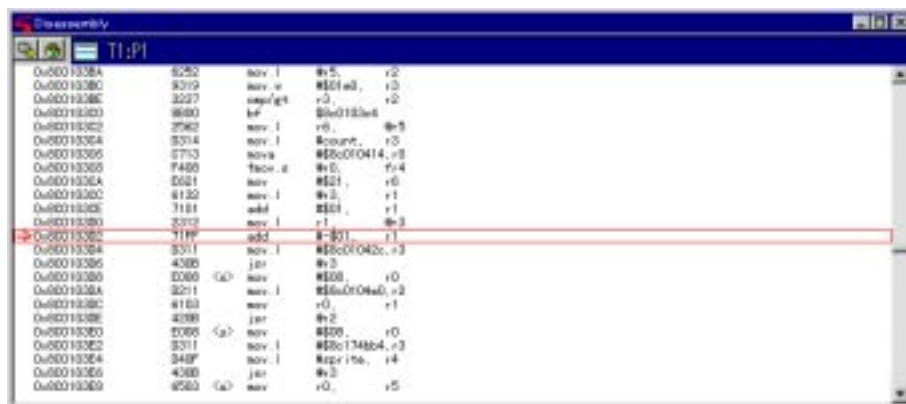
最上段に現在のプログラムカウンタ (PC)、そしてその下に現在のプログラムスタックの状態がリストされています (リストは下に行くほど古いものになります)。

ここにはアドレスが、関数名、またその関数名からのオフセット値で示されています (今、先ほどの Breakpoint が有効なままですので、その行で停止しています^{* 12})。

* 12

SET4 (or SET5) の実行中、[Call Stack] リージョンには RUNNING... とだけ表示されます。

確認のため、この状態から [Disassemble] リージョンを開いて見てみましょう * 13。

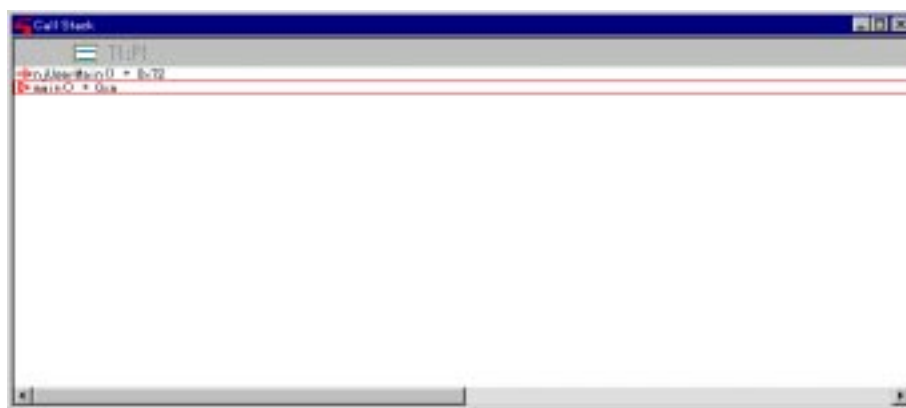


開いた直後は、カーソル位置が現在の PC です。

確かに現在その位置 (njUserMain() + 0x72) にいることが分かります。

その 1 つ下のスタックも確認してみましょう。

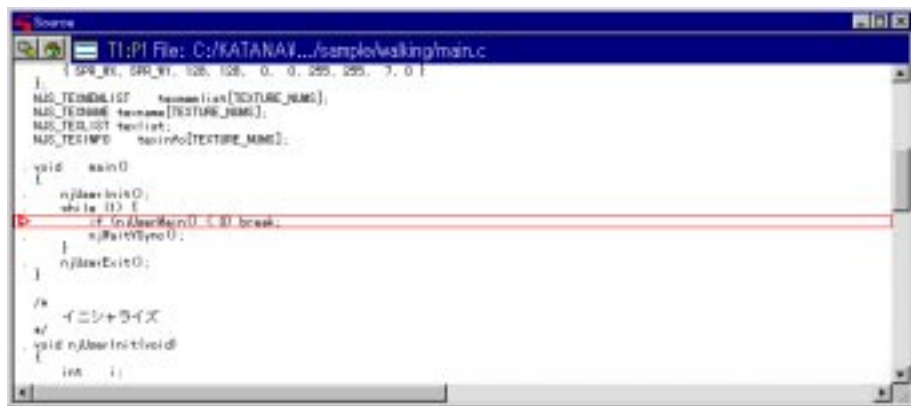
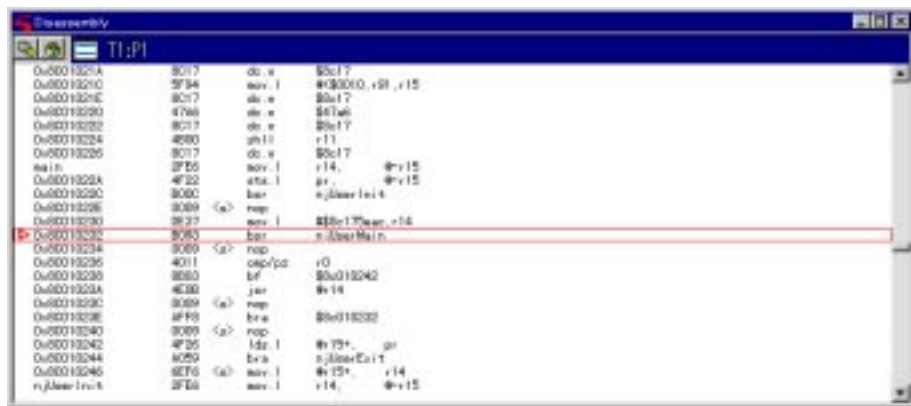
Call Stack リージョンの、 “ main() + 0xa ” の行をダブルクリックします。



このように表示が変わり、またこれに連動して [Disassemble] リージョンと [Source] リージョンのカーソル位置が、この njUserMain(); 関数の呼び出しを行った個所に移ります。

* 13

また [Disassemble] リージョンの表示から、SH4 CPU は 2 バイトを 1 命令の単位としていることも分かります。



ところで今現在、このように [Source] リージョンと [Disassembly] リージョンが同時に開いているはずですが、この2つは同期を取って、つまり片方の動きに合わせてもう片方を動かすことが可能です (具体的な方法については『CodeScape マニュアル』を参照してください)。ソースレベルで取りきれないバグについては、この方法で調査することになります* 14。

* 14

SH4 CPU についての知識と、コンパイラの動作の理解が必要になります。

2.2.6 トレース

CocScope は幾つかのトレース実行機能を持っています。

これらの機能はデバッグツールバーから呼び出すことができます。



カーソル位置まで実行 (Run to cursor)

現在のカーソル位置まで実行し、停止します。

ここでは、先ほど Breakpoint を設定した行の位置まで、この機能で実行することにしましょう。[Source] リージョンを開きます。

以前の Breakpoint が残っている場合には、その行で [F5] キーを押してこれを解除します。続いて、カーソルを実行したい行の位置に持って行きます* 15。



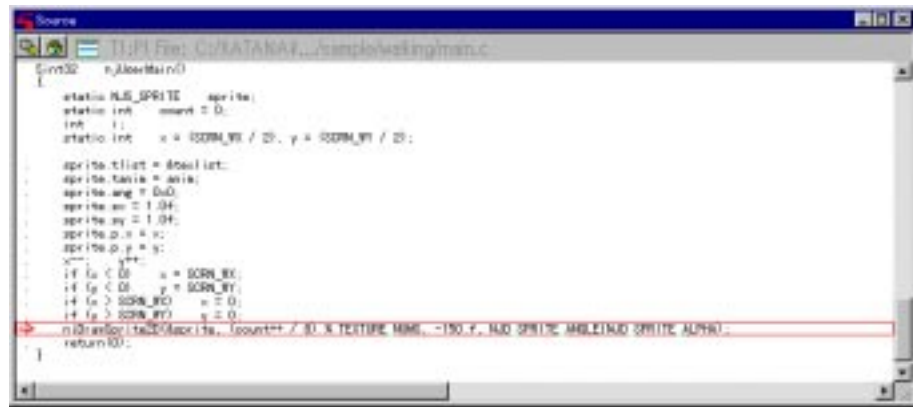
この状態で、次のいずれかの方法で現在のカーソル位置まで実行します。

- [Debug] メニューの [Execution] をクリックし、[Run to Cursor] をクリック
- [Debug] ツールバーの [Run to cursor] ボタンをクリック
- [Alt] + [F9] キー
- [Source] リージョンを右クリックし、[Execution] をクリックして [Run to Cursor] をクリック

すると表示が次のように変わり、カーソル位置まで実行され、停止したことがわかります。

* 15

左クリックでも、カーソルキーでも構いません。



またこの時 [Target] ウィンドウは



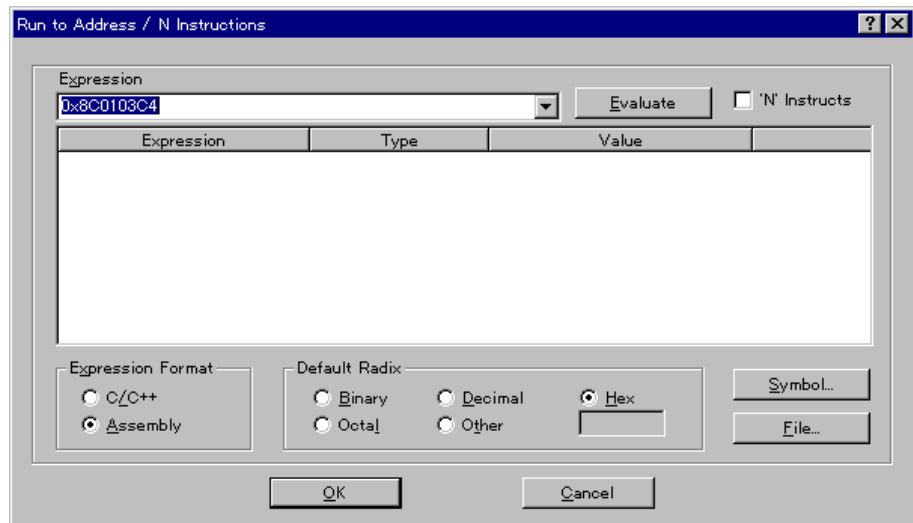
という表示になります。

指定アドレスまで実行 (Run to Address)

指定したアドレスまで実行し、停止します。

- ・ [Debug] メニューの [Execution] をクリックし、[Run to Address] をクリック
- ・ [Debug] ツールバーの [Run to address] ボタンをクリック
- ・ [Shift] + [F9] キー
- ・ [Source] リージョンを右クリックし、[Execution] をクリックして [Run to Address] をクリック

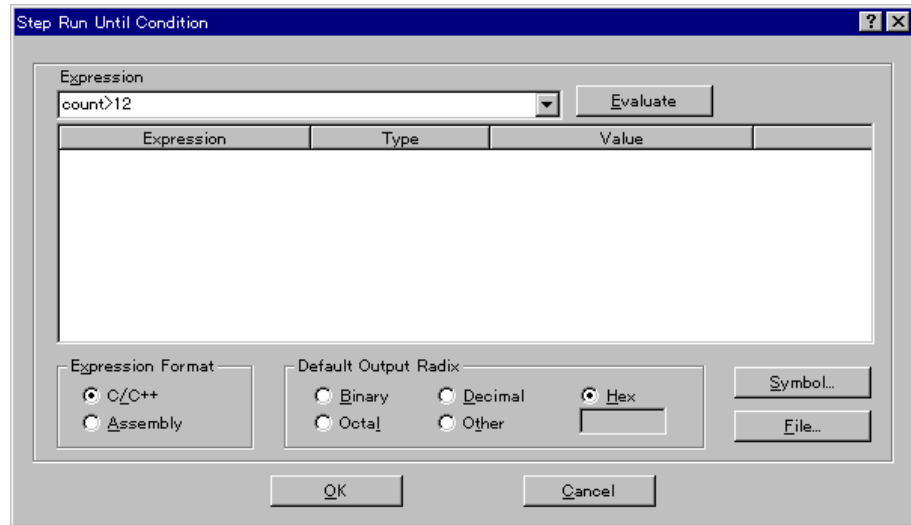
次のようなダイアログボックスが表示されます。



条件が満たされるまで実行 (Run until) ...

与えた式が満たされるまで実行します。

- ・ [Debug] メニューの [Execution] をクリックし、[Step Run Until] をクリック
- ・ [Debug] ツールバーの [Step run until] ボタンをクリック
- ・ [Alt] + [F8] キー
- ・ [Edit] リージョン以外のリージョンを右クリックし、[Execution] をクリックして [Step Run Until] をクリック



[Expression] に条件式を入れます。

この式が満たされるまで実行されます。

しかしながらこの機能はかなり実行時間がかかります* 17。つまり実時間動作が行えません。このことが原因で期待したトレースが行えない場合があることも頭に入れて使用してください。また Watch access breakpoint と同様、停止した時には既にアセンブラレベルで数命令先に進んでいることにも気を付けてください。

* 17

しかし [Debug] - [Execution] - [Enable Animated Step Run] のチェックをオンにしておくことで、実行状況を刻々各リージョンに表示することができます。

ステップ実行

プログラムを 1 行単位で実行します。

ステップ実行には、以下の 3 種類の方法があります。

- ・ 関数の中もトレース (Step into)
- ・ 関数の中はトレースしない (Step over)
- ・ この関数から抜け出る (Step Out)

これらの機能を 1 つずつ見て行ってみましょう。

関数の中もトレース (Step into)

呼び出した関数や、サブルーチンの中もトレースします。

Step into は以下のいずれかの操作で行い、これらを行うごとに 1 行ごとに進みます。

- ・ [Debug] メニューの [Execution] をクリックし、[Single Step] をクリック
- ・ [Debug] ツールバーの [Single step (into)] ボタンをクリック
- ・ [F7] キー
- ・ [Edit] リージョン以外のリージョンを右クリックし、[Execution] をクリックして [Single Step] をクリック



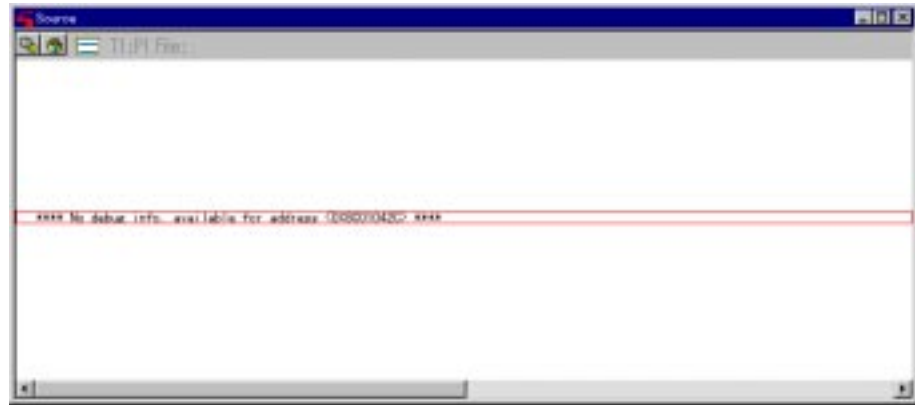
たとえば上の状態で停止している場合、[F7] キーを 1 回押します。



1 行分進みました (宣言文や空行は実行されません)

この時点で現在のカーソル位置までは未だ実行されていないことに気を付けてください。

何度か [F7] キーを押し、進めてみます。



すると、あるところからソースが見えなくなっていました。

これはソースのない、ライブラリ内関数 `njDrawSprite2D()`; 内部のトレースに入ったためです* 18。この場合、後に述べる Step Over を使用して、先ほどのソースまで進めることができます。またトレースの必要ない関数の場合、これも後に述べる Step Out を使用してスキップする (実行するが、トレース状況を表示しない) ことができます。

この Step into を行った場合に限り、Unstep で 1 行ずつプログラムを逆に戻ることができます。

Unstep は以下のいずれかで行います。

- ・ [Debug] メニューの [Execution] をクリックし、[Unstep] をクリック
- ・ [Ctrl] + [F7] キー
- ・ [Debug] ツールバーの [Unstep] ボタンをクリック
- ・ [Edit] リージョン以外で右クリックし、[Execution] をクリックして [Unstep] をクリック

* 18

ソースのない関数のトレースが必要な場合、[Disassemble] リージョンと [Register] リージョンなどを使用します。

関数の中はトレースしない (Step over)

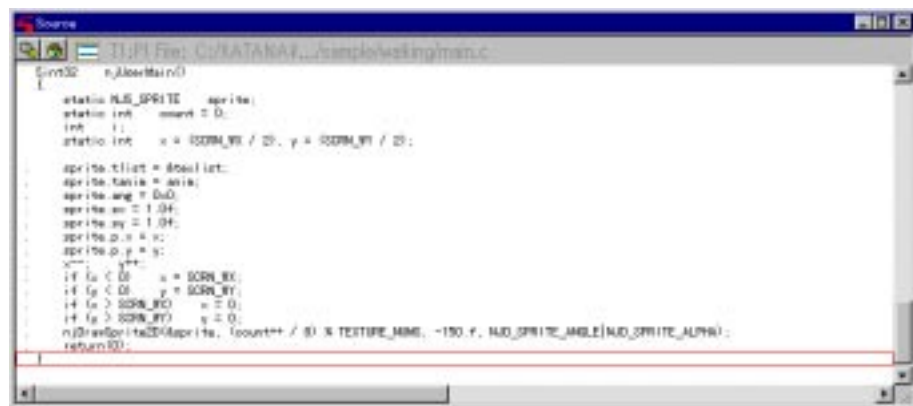
Step into では関数コールや JSR、BSR があった場合には呼び出した関数またはサブルーチンの中までトレースしましたが、Step over ではそれを行いません。

- ・ [Debug] メニューの [Execution] をクリックし、[Step Over] をクリック
- ・ [Debug] ツールバーの [Step over] ボタンをクリック
- ・ [F8] キー
- ・ [Edit] リージョン以外のリージョンを右クリックし、[Execution] をクリックして [Step Over] をクリック

まず既に説明した Breakpoint の設定機能を使い、njDrawSprite2D(); が呼ばれるところまで実行しましょう。



ここで [F8] キーを 1 回押します。



このように、Step into と異なり関数 njDrawSprite2D(); の内部までのトレースは行いません。

ちなみに [Watch] リージョンで変数 count の値を見ている場合、ここで変数 count の値が変わることが分かります。

この関数から抜け出る (Step Out)

現在いる関数またはサブルーチンの終わりまで実行し、関数を抜けます。

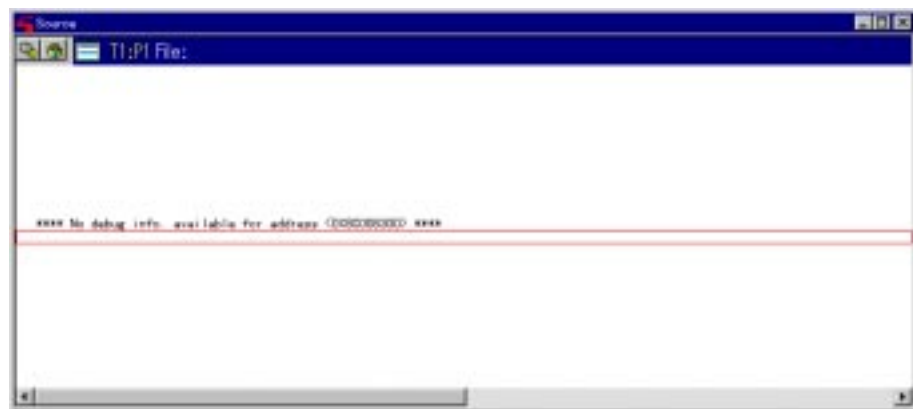
- ・ [Debug] メニューの [Execution] をクリックし、[Step Run Out] をクリック
- ・ [Debug] ツールバーの [Step Out] ボタンをクリック
- ・ [Shift] + [F8] キー
- ・ [Edit] リージョン以外のリージョンを右クリックし、[Execution] をクリックして [Step Out] をクリック

2.2.7 ソースリスト中の、特定の範囲をチェックする

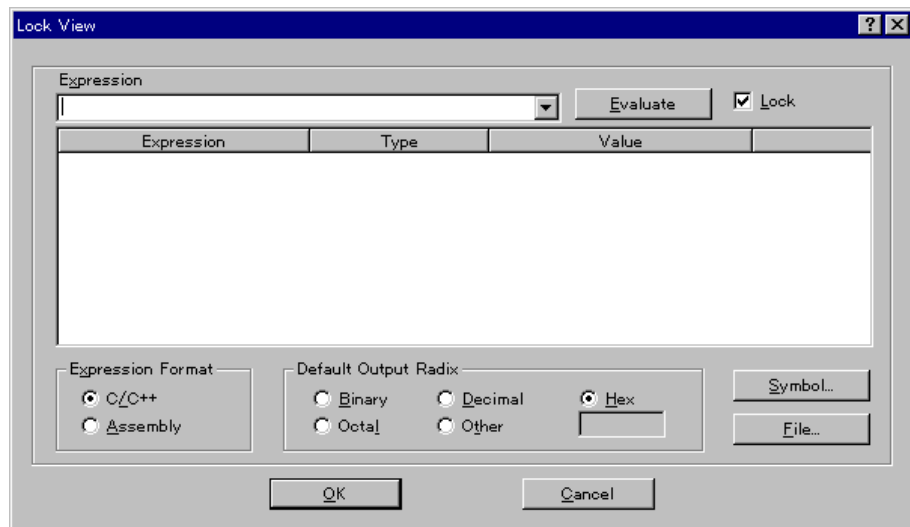
意図しない動作がある場合トレースなどを行うこととなりますが、トレースを行う時、特定の怪しそうなところを重点的に見ておきたいことがあります。

そのような時には [Source] リージョンに Lock View を行います。Lock View を行うことで、PC 値によらず特定のソースファイルの特定の個所を常に表示させておくことができます。

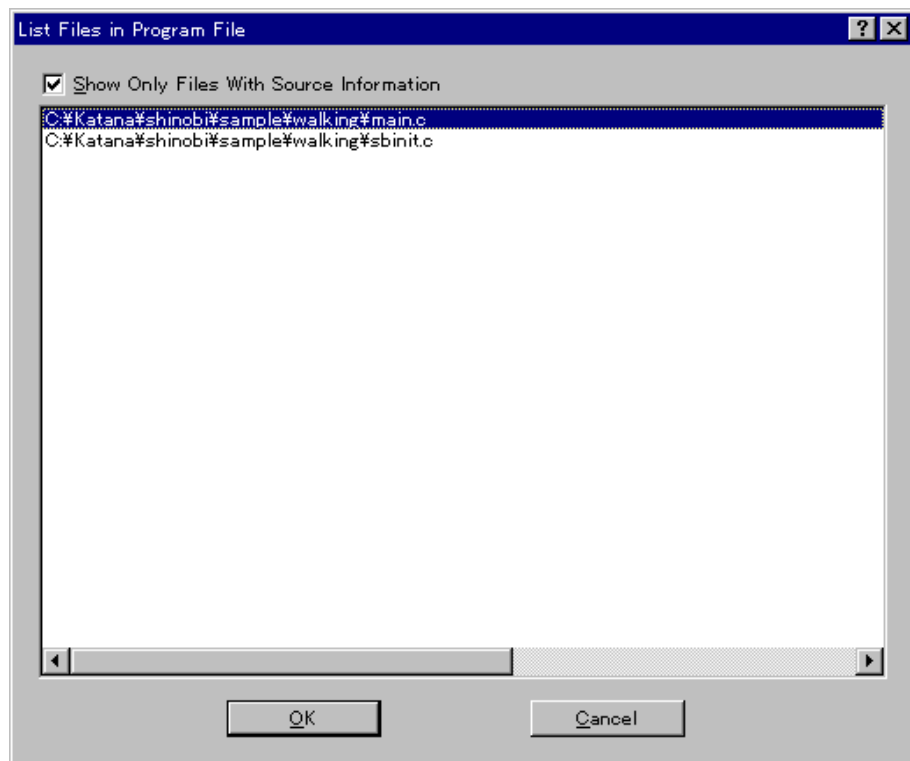
まず [Source] リージョンを開きます。



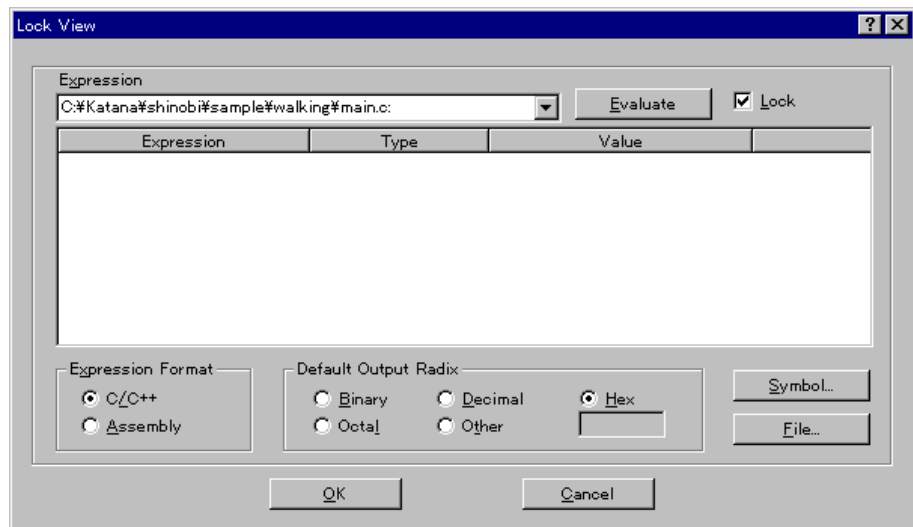
 をクリックします。



[File] をクリックすると、ソースファイルの一覧が表示されます。




常に表示させておきたい (ロックしたい) ファイルを指定します。
ここでは main.c をクリックします。



[Expression] にファイル名が入りました。ファイル名の後の:に続けてソースファイルの行指定を行うことで、特定の個所に Lock をかける (特定の行からの表示を行う) ことができます。

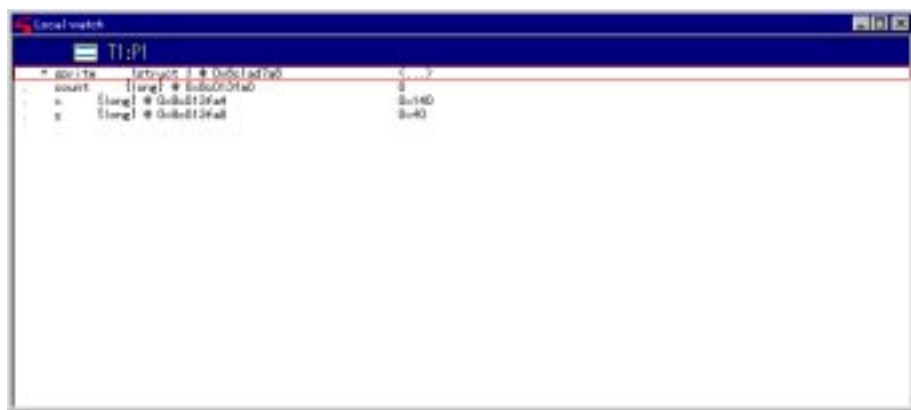
Lock されたリージョンは、スクロールバーの操作なども無効になります。

解除するには再度  をクリックし、[Lock View] ダイアログで [OK] ボタンをクリックします。

変数の内容を見る

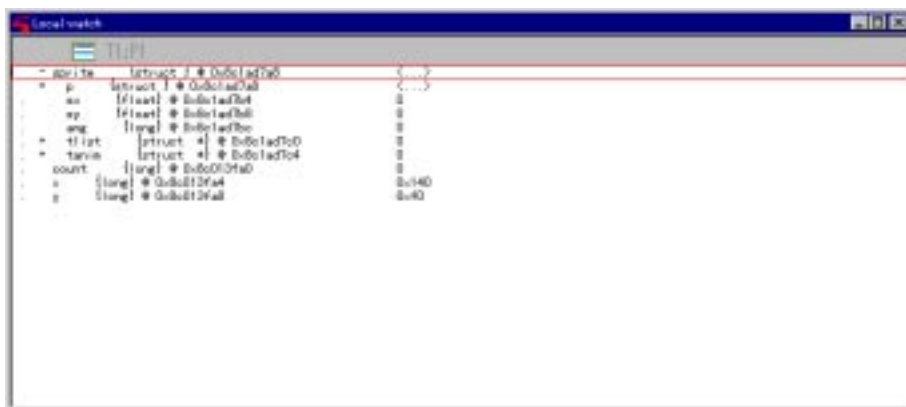
続いて変数の内容を調べる場合、調べたい変数名を 1 つ 1 つ打ち込んでも良いのですが、[Local watch] リージョンを開くことで、その関数のローカル変数一覧を見ることができます。

停止している時に、[Local watch] リージョンを開きます。



停止している個所の関数で使用されているローカル変数の一覧とその内容が表示されます。
ここでは njUserMain(); のローカル変数が表示されています。

ローカル変数が構造体の場合、その構造体名の上で右クリックし [Open/Close] を選ぶことで、構造体のメンバー一覧とその内容を見ることができます* 19。



* 19

この動作はトグルになっていますので、今展開した構造体を再度 Open/Close すると閉じられます (構造体名のみ表示に戻ります)。

第 3 章

トラブルシューティング

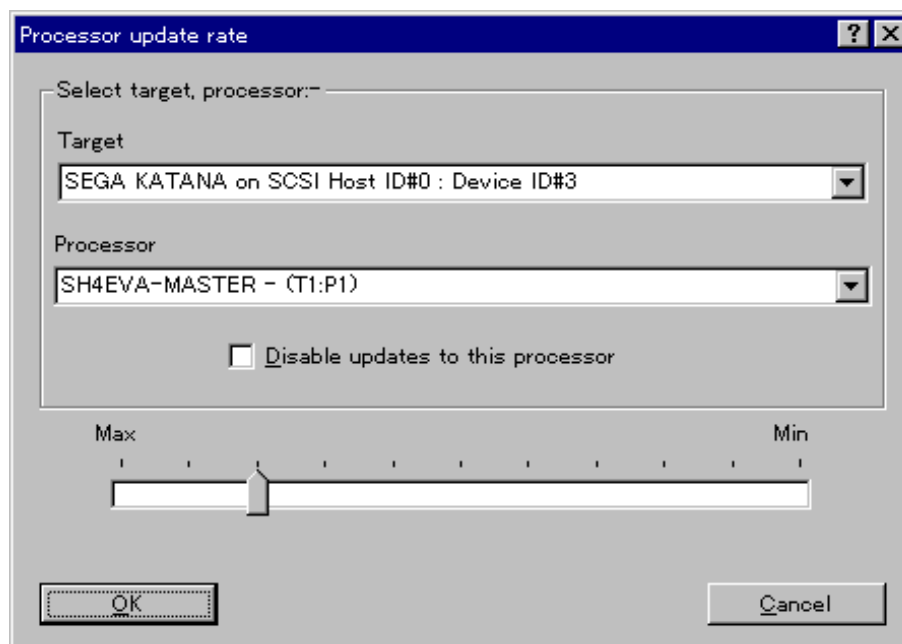
ここでは、比較的良好にあるトラブルとその回避方法を示します。

3.1 SET5

3.1.1 プログラムをロードしても動作しない(ケース 1)

前提としている CPU パワー以下のホストマシンで実行した場合、SET5 との通信がうまくいかず、その結果として動作しない、または動作が不安定になる場合があります。

この場合、CodeScape の [Processor Combo] ツールバーで [Configure Processor] ボタンをクリックし、[Processor update rate] ダイアログボックスで SET5 との通信速度を落とすことによって動作可能になる場合があります* 1。



このデータスライダーを [Min] 方向に下げる、または [Disable updates to this processor] チェックボックスをオンにします。

* 1

通信速度を下げることによって Pentium 133MHz 機以下でも動作した事例があります。ただし、前提としている環境と異なるもので動作保証は行っておりません。

3.1.2 プログラムをロードしても動作しない(ケース 2)

ケース 1 に該当しない場合、ハードウェアの故障を除き、たとえば以下の原因が考えられます。

- ・ SET5 が正しくセットアップされていない* 2
- ・ SCSI ボードが推奨のものではない
- ・ SCSI ケーブルやターミネーターの不良、またはそれらの接続不良
- ・ ASPI ドライバが古い
- ・ ディスプレイタイプが異なる

SDK 添付のサンプルプログラムは VGA ディスプレイを想定して作成されているものがあります。NTSC モニタでは、正しく同期が取れない、画面が縮む、などの現象が起こります。

3.1.3 ホストマシンとの接続について

SCSI ID

デフォルトでは 3, 4 を使用します。必要に応じて変更することも可能です。

SCSI インターフェイスについての情報

現在 Adaptec AHA-2940U (or AHA-2940AU) 以外での動作保証は行っておりません。

当社指定のものをお使いください。安定動作のためには EZ-SCSI Pro を追加購入されて最新の ASPI ドライバを入手されるか、それが添付されたパッケージ製品をお勧めします。

同一 SCSI バス上には SET5 以外のものを接続しないでください。

なお SCSI カードの複数挿しなどの環境に対しての動作確認は現在行っておりませんので、他に、たとえば MO などの SCSI デバイスが必要である場合、LAN 経由でお使いいただく方が安全です。また HDD も IDE のものをお使いください。

添付の SCSI ケーブルが短い

筐体を開け、適当な長さのケーブルに付け替えることが可能です。

ただし必要以上に長くすることはお勧めしませんので、可能な範囲で短いものをお使いください。

* 2

Katana/SET4 Setup Guide もあわせてご参照ください。

3.2 CodeScape

3.2.1 バージョンを上げたら起動しなくなった

SDK を新しいものに差し替えたなどで CodeScape のバージョンを上げた場合、CodeScape の起動時に No valid device と表示され、起動できなくなる場合があります。

バージョンによっては、ショートカットから起動した場合に CodeScape と同一ディレクトリにある SET5 の新しいファームウェアを認識できず、アップデートが行えないことがその理由です。このような状況に陥った場合、一度 Explorer から CodeScape.exe を起動し、ファームウェアをアップデートさせてください。

ファームウェアは自動的にアップデートされます。終了したら、一度 SET5 の電源を入れ直してください。

ファームウェアのアップデートは DACHECK ユーティリティからも行えます。

DACHECK は、SDK をインストールしたディレクトリの ¥katana¥utility¥dacheck にあります。

3.2.2 Autoexec.bat に set_kt.bat が記述できない

Windows 95 では、“ autoexec.bat ”の中に“ set_kt.bat ”を含めると、Windows 95 の起動時に“ Not enough space for environment. ”と表示されることがあります。

この場合、“ config.sys ”に以下の記述を追加して環境変数エリアを広げてください。

```
shell = c:¥windows¥command.com c:¥ /E:2048 /P
```

/E:が環境変数領域の指定です。単位はバイトです。

または、“ set_kt.bat ”を“ autoexec.bat ”から外し、DOS プロンプトから“ set_kt.bat ”を実行して SET コマンドで得られる環境変数一覧をファイルにリダイレクトし、そのファイルを [Project Setup] ダイアログで指定することで環境変数を設定する手段があります。詳細は本チュートリアル該当ページをご覧ください。

なお Windows NT 4.0 では、このトラブルは発生していません。

3.3 プログラム作成に関わる注意点

SEGA Saturn または他プラットフォームから移行されてくる場合の注意点を示します。

3.3.1 グラフィックデータの利用について

Shinobi ライブラリでは、テクスチャデータ形式として PVR 形式 (PVR ファイルフォーマット) を使用しています。思想は従来の DGT 形式などと似ていますが、ライブラリ自身がサポートするところが大きく異なります。

Shinobi ライブラリではテクスチャパターンに固有のインデックス番号を割り付けることでその管理を行っています。ライブラリ関数ではこのインデックス番号を利用してパターンへのアクセスを行います。

このことによりユーザープログラムサイドでの管理の手間が大幅に縮小されますが、この従来と異なる手法には幾つか注意しなければならないことがあります。

テクスチャリスト

いくつかのテクスチャパターンを集めたものです。複数設定できます。

テクスチャ関連関数の多くが、このテクスチャリストを単位としてテクスチャ操作を行います。

また複数のテクスチャリストを使用する場合、現在どのテクスチャリストに対して操作を行うのかを指定する、「カレントテクスチャリストの設定 (njSetTexture();)」、「カレントテクスチャの設定 (njSetTextureNumG(); njSetTextureNum();)」が必要になります。それぞれの関数の詳細については、『Ninja ライブラリ 関数リファレンス』をご参照ください。

グローバルインデックス

テクスチャパターンに割り振る、固有の番号です。

この値が同じテクスチャパターンは、同一のものとみなされます。

設定は njSetTextureName(); でその都度割り付ける方法と、PVR ファイル自身にあらかじめ持たせておく方法の 2 つがあります。

複数のテクスチャリストで同一のテクスチャパターンを使用したい場合、それぞれに同じグローバルインデックス値を指定することで、複数のテクスチャリスト間で同じテクスチャパターンを使いまわすことができます。これによりテクスチャメモリの浪費を防ぐことができます。

インデックスには、すべてのテクスチャに対して有効である、上記「グローバルインデックス」と、そのテクスチャリスト内で有効な「テクスチャ番号」があります。後者は njSetTextureName(); の引数である NJS_TEXNAME 構造体の配列のはじめから、0, 1, 2, 3... のように付きます。

この 2 つの値は、それぞれ引数として使用される関数が異なります。ただし多くのケースでは、njSetTextureNumG(); と njSetTextureNum(); のように、それぞれに意味の同じ関数が用意されています。

抜きパターンを別に用意する

従来はカラーコードの 0 がいわゆる「抜き色」になっていましたが、KATANA では透明度合いを示す「アルファ値」という要素が増えました。

この値を使用してテクスチャ(スプライト)パターンの「抜き」を行います。要するに、「抜き」にしたい個所の透明度合いを設定する / 上げるということです。

テクスチャの元データに Windows ビットマップファイル(.bmp)を使用した場合、同一ファイル中にこのアルファ値を含めることはできませんので、全くサイズの同じビットマップファイルを用意し、そちらで抜きにしたい個所を赤(Red)の値で指定して示します^{* 3}。たとえば以下のようにします。

: RGB = 00-00-00

: RGB = FF-00-00

これで、 の個所が抜きに指定されます。

この場合はアルファが 2 値(00, FF)なので、pvrconv を使用するとデフォルトでは ARGB1555 にコンバートされますが、00, 3F, 7F, FF のように 2 値以上を指定すると ARGB4444 でコンバートされます。この時は、上のデータは単なる「抜き」ではなく、抜きの度合い、つまり半透明の度合いを示す値(アルファ値)として使用されます。

詳しくは pvrconv のマニュアルを参照してください。

* 3

正確には赤成分の値を見ますので、抜きにや半透明にたくない個所は必ず赤成分を 0 (たとえば黒(00-00-00)) にしてください。勿論抜きにする必要のないパターンでは、これらの処置は必要ありません。

3.3.2 その他

画面のフリップ

サンプルプログラムでは明示的に画面のフリップを行う箇所がありませんでしたが、これは `njWaitVSync()` の中で自動的に行われています。またこのフリップのタイミングは `sbInit...System()` の引数で設定します。

3.3.3 CodeScape 開発環境に併用すると便利なツール類

以下に CodeScape と併用すると効果がある、または必要なツール類を示します。

ただし当社ではこれらのものが安全に使用できることの保証は行いませんので、開発者のご判断の基にご利用ください。

VisualStudio

Microsoft の開発環境で、Windows プログラミング環境としては標準的なものです。本書で使用している `nmake.exe` もこれに添付されています。

EZ-SCSI Pro

アダプテックから発売されている、ASPI ドライバと SCSI ユーティリティが入ったパッケージ製品です。最新ドライバへの差分は、アダプテックの Web ページ(<http://www.adaptec.co.jp/>) から入手が可能です* 4。

バイナリエディタ

foxbin などのフリーウェアがあります。

foxbin は、<http://www.maple.or.jp/~sagawa/foxbin.html> から入手することができます。

Susie

フリーのグラフィックビューワです。

いくつかのプラグインが公開されており、各種フォーマットに対応可能であることに加え、プラグインを自作することで、特殊なフォーマットに対応することが可能です。

Susie は、<http://www.digitalpad.co.jp/~takechin/> から入手することができます。

* 4

差分は、現在 <http://www.adaptec.co.jp/download/download.html> に置かれています。

- CodeScape ユーザーズガイド
株式会社セガ・エンタープライゼス
- Katana/SET4 Setup Guide
株式会社セガ・エンタープライゼス
- SH7091 プログラミングマニュアル
株式会社日立製作所
- SH アセンブルマニュアル
株式会社日立製作所
- リンカなど H シリーズマニュアル
株式会社日立製作所
- SHC コンパイラマニュアル
株式会社日立製作所
- SuperH RISC engine ファミリ C コンパイラ編
株式会社日立製作所

付録 B

参考 URL

株式会社セガ・エンタープライゼス	http://www.sega.co.jp/
Cross Products Limited.	http://www.crossprod.co.uk/
株式会社日立製作所	http://www.hitachi.co.jp/
マイクロソフト株式会社	http://www.microsoft.com/japan/
アダプテックジャパン株式会社	http://www.adaptec.co.jp/
アダプテック (U.S)	http://www.adaptec.com/