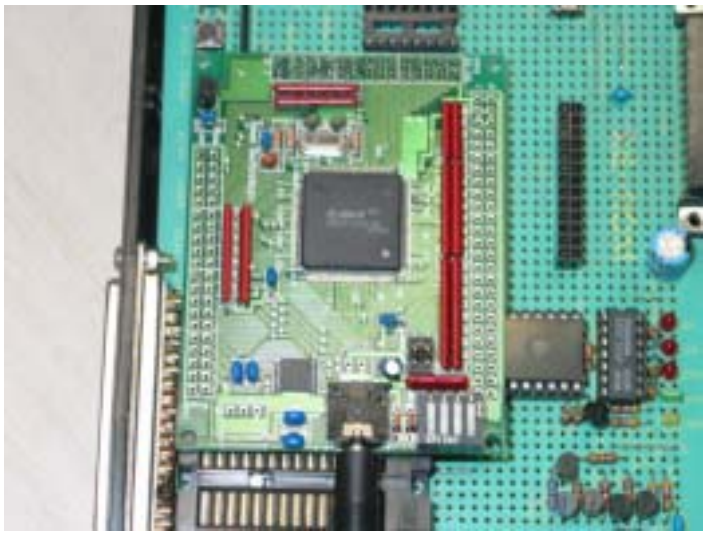
はじめに

このドキュメントは、MATLAB6.1/SIMULINK に Real-time Workshop をベストテクノロジ社の SH2 マイコンに適用する方法について記述したものである。これにより SIMULINK 上からＧＵＩプログラミングが可能になる。一応、例として 12 個のＰWM制御ができるようにはなっているが、処理速度の面ではまだ改良の余地がある。

Real-Time Workshop とは？

MATLAB 上で動作する SIMULINK ブロックダイアグラムを C 言語に変換するツールボックスである。Real-TimeWorkshop は、標準的な C 言語のソースコードを吐き出すようにつくられており、標準的な C コンパイラであるならば、コンパイル実行可能である。
そのため、VisualC++,BorlandC++,WatcomC++,gcc などどのようなプロセッサでも実行できる。ここでは、GCC が使用可能なベストテクノロジー社の SH2 マイコンを対象に Real-Time Workshop を使うためにどのような設定をするのかを解説していく。

SH2 マイコンへの適用例

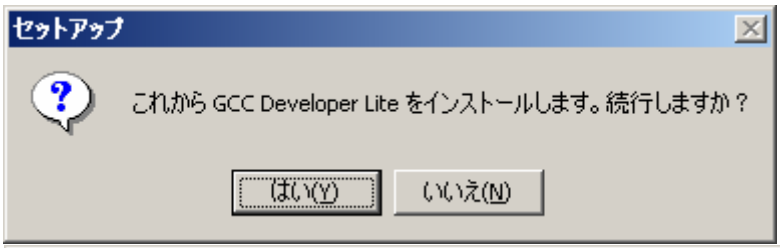


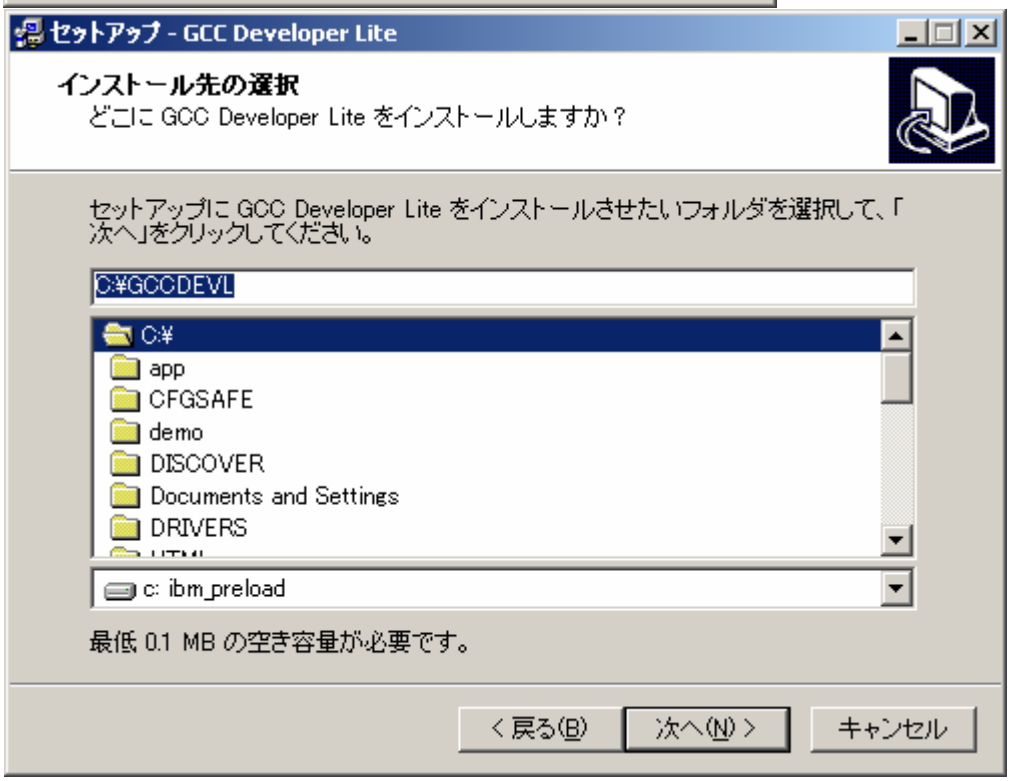
ここでは、MATLAB＆SIMULINK を用い SH2 マイコン用プログラムを自動生成する。ここで使用する SH2 マイコンは、ベストテクノロジー社製を対象に、PWM、AD コンバータ、シリアル通信ポート、PIO などの機能を SIMULINK 上で使えるようにするのが目的となる。

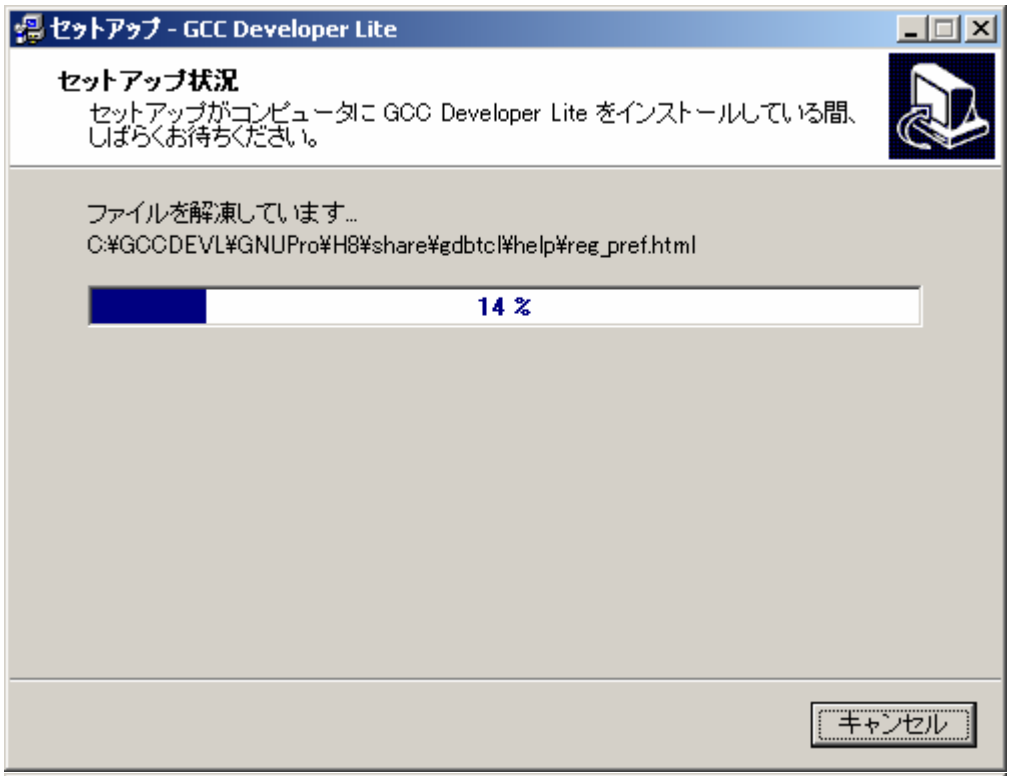
Step.1　GDL のインストール

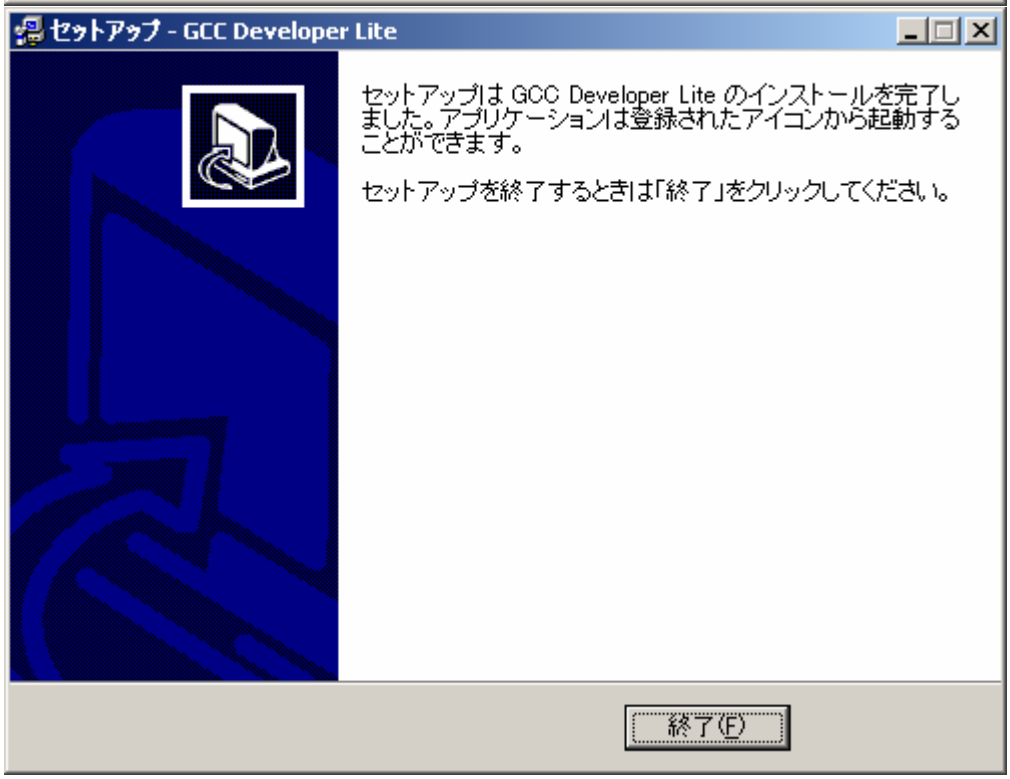
まず、SH2 用のＣコンパイラをインストールする必要がある。SH2 用Ｃコンパイラとして、SH2 マイコンに付属している GDL を用いる。このコンパイラは、C++の機能はないが、RealTimeWorkShop のための支障はない。
まず、そのインストール手順を示す。


C:¥app¥sh2¥GDL
ファイル(F)　編集(E)　表示
戻る　検索
アドレス(D)　C:¥app¥sh　移動
GDLFull1.2.4.0
.exe
1 個のオブジェ
67.1 MB
マイ コ


セットアップ
これから GCC Developer Lite をインストールします。続行しますか？
はい(Y)
いいえ(N)


セットアップ - GCC Developer Lite
インストール先の選択
どこに GCC Developer Lite をインストールしますか？
セットアップに GCC Developer Lite をインストールさせたいフォルダを選択して、「次へ」をクリックしてください。
C:¥GCCDEVL
C:¥
app
CFGSAFE
demo
DISCOVER
Documents and Settings
DRIVERS
c: ibm_preload
最低 0.1 MB の空き容量が必要です。
< 戻る(B)
次へ(N) >
キャンセル





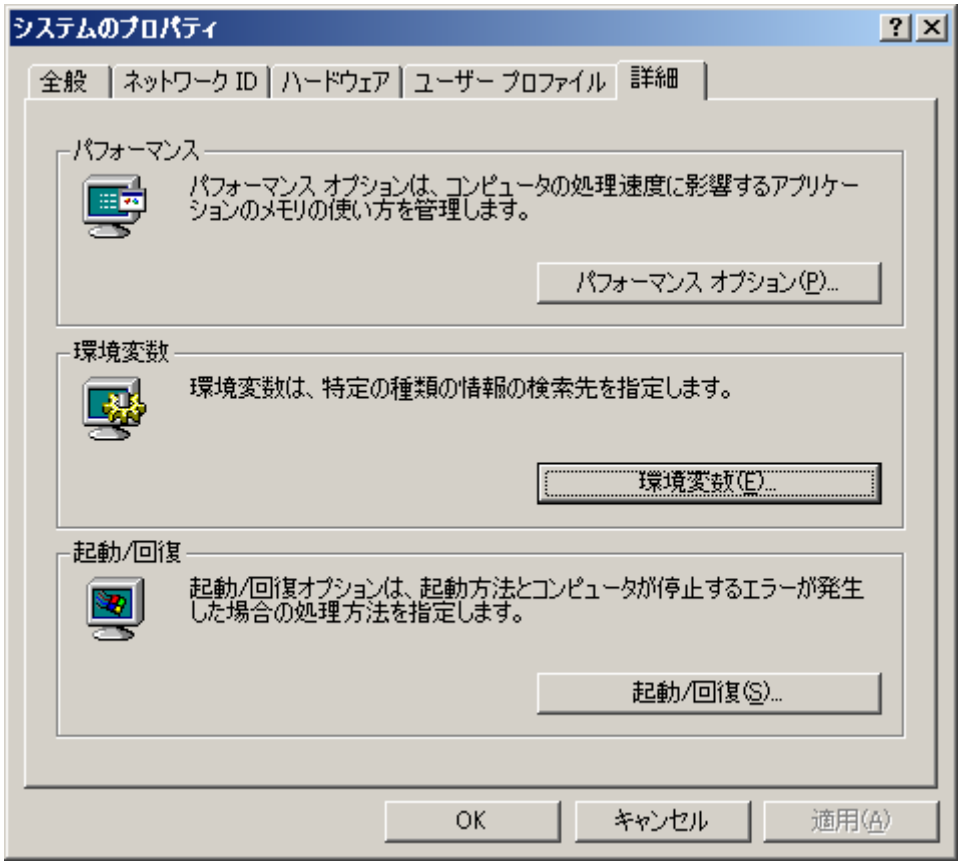
Windows 上で GCC を使用する場合の注意点

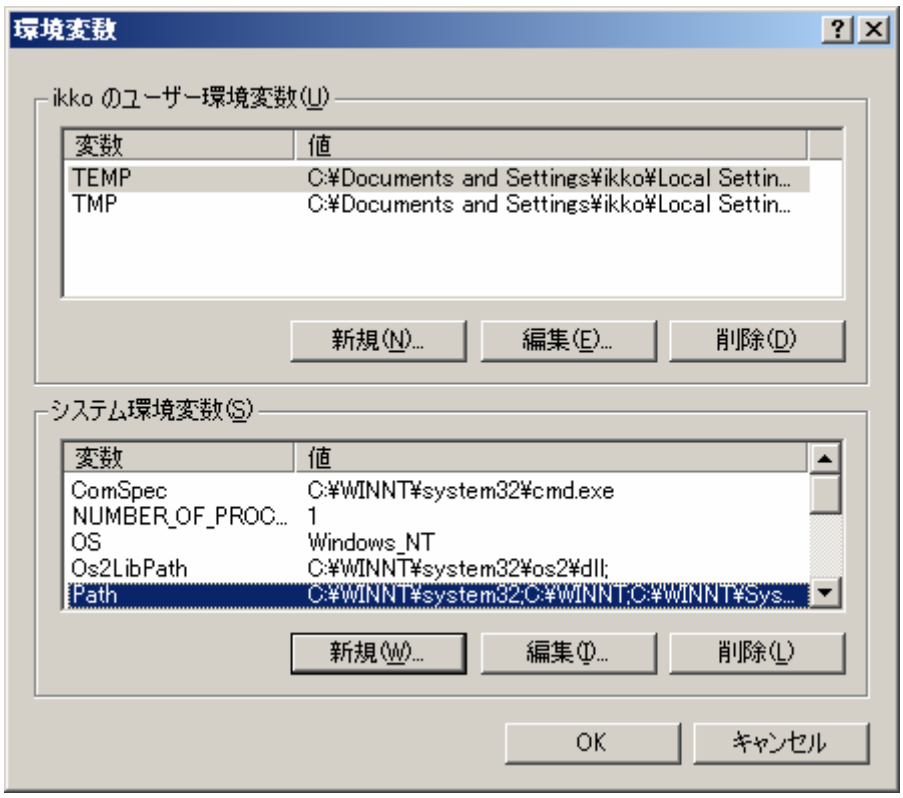
Windows 上の C では、コマンドラインでコンパイルする場合、ディレクトリの指定は、バックスラッシュ¥、スラッシュ/どちらでも動作するのに対し、GCC では、スラッシュ/を使

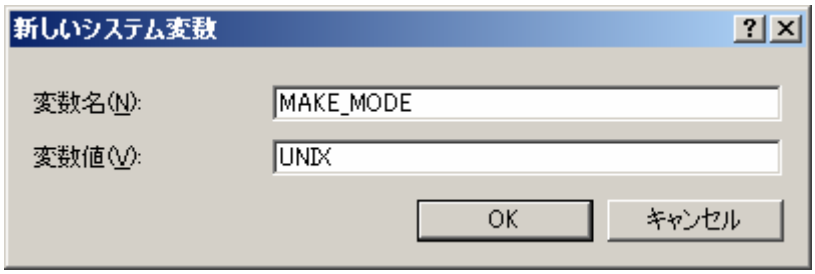
用する。
Real-Time Workshop のプログラムを動作させるには、バックスラッシュをスラッシュに置き換える必要がある。

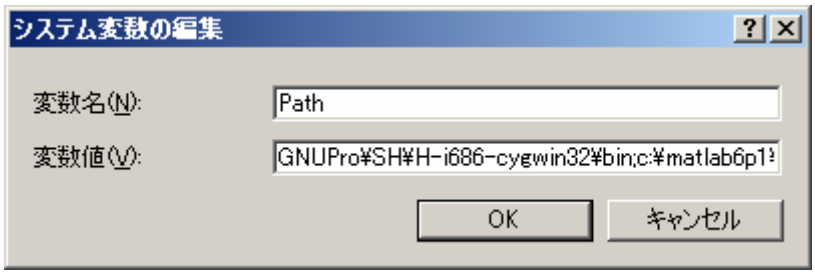
GCC コンパイラのパスの設定

ここでは、GCC コンパイラとして、GNUPro でインストールされた GCC をそのまま使用する。SH2 用として、GNUPro の SH 用 GCC コンパイラを使用する。また、MAKE ユーティリティを使ってコンパイルする必要があるため、環境変数として MAKE_MODE を UNIX として指定する必要がある。また MAKE モードを UNIX とした場合、ファイル名の展開などのために sh.exe が必要になる。sh.exe は、GNUPro には無いので、インストールされた Cygwin のツールの中などから sh.exe のみをコピーし、パスの効いたディレクトリにコピーする必要がある。例えば、c:¥GCCDEVL¥GNUPro¥SH¥H-i686-cygwin32¥bin の中に入れておくとよい。
次の設定は、WINDOWS95/98 であるならば、c:¥autoexec.bat
に追加記述
SET MAKE_MODE=UNIX
SET PATH=c:¥GCCDEVL¥GNUPro¥SH¥H-i686-cygwin32¥bin;%PATH%
WINDOWS　ME ならば、ファイル名の実行で msconfig を実行し
PATH を追加
MAKE_MODE も追加する。
WINDWOS2000 ならば、環境変数を追加する必要がある。


システムのプロパティ
全般
ネットワーク ID
ハードウェア
ユーザー プロファイル
詳細
パフォーマンス
パフォーマンス オプションは、コンピュータの処理速度に影響するアプリケーションのメモリの使い方を管理します。
パフォーマンス オプション(P)...
環境変数
環境変数は、特定の種類の情報の検索先を指定します。
環境変数(E)...
起動/回復
起動/回復オプションは、起動方法とコンピュータが停止するエラーが発生した場合の処理方法を指定します。
起動/回復(S)...
OK
キャンセル
適用(A)

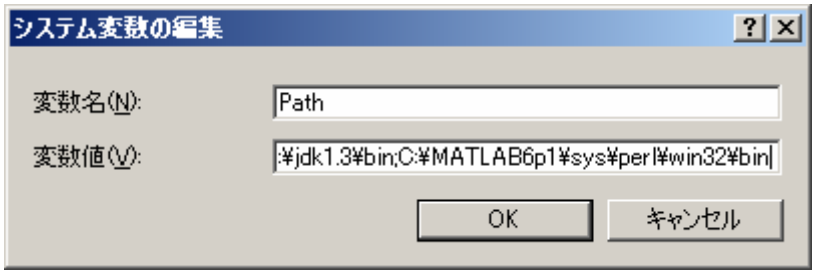

環境変数
ikko のユーザー環境変数(U)
変数
値
TEMP
C:¥Documents and Settings¥ikko¥Local Settin...
TMP
C:¥Documents and Settings¥ikko¥Local Settin...
新規(N)...
編集(E)...
削除(D)
システム環境変数(S)
変数
値
ComSpec
C:¥WINNT¥system32¥cmd.exe
NUMBER_OF_PROC...
1
OS
Windows_NT
Os2LibPath
C:¥WINNT¥system32¥os2¥dll;
Path
C:¥WINNT¥system32;C:¥WINNT;C:¥WINNT¥Sys...
新規(W)...
編集(I)...
削除(L)
OK
キャンセル





また、SIMULINK 用ライブラリを自動コンパイルするためには、perl スクリプトを使うため Perl へのパスを追加する必要がある。幸いにも MATLAB をインストールすると付属として Perl がインストールされるため、そこへのパスを追加するだけでよい。以下に Perl へのパスの追加例を示す。

C:¥MATLAB6p1¥sys¥perl¥win32¥bin に perl があるのでパスを追加する。



SIMULINK ブロックランタイムライブラリ

SIMULINK ランタイムライブラリ構築のためのソースコードは、

$(MATLABROOT)/rtw/c/libsrc/

にある。これらを全て SH2 用にコンパイルしライブラリ化し、rtwlib.a として保存しておく。コンパイルの際の GCC コンパイルオプションは、以下の通りである。

-m2　SH2 用コードの生成

-DRT　RealTimeWorkshop 用オプション

-DUNIX

ヘッダのインクルード

-Id:/matlabr12/simulink/include

-Id:/matlabr12/extern/include

-Id:/matlabr12/rtw/c/src

-Id:/matlabr12/rtw/c/libsrc
を行う。この際、MATLAB のディレクトリは、環境により異なるので変更する必要がある。
また、GCC のコンパイルの際に指定するディレクトリは、パス設定の時のバックスラッシュではなく、スラッシュである点に注意すること。
コンパイルするには、GNUPro は使用せずコマンドラインでコンパイルを行なう。
例えば、rt_atan2.c をコンパイルするには、

sh-elf-gcc -m2 -DRT -Id:/matlabr12/simulink/include -Id:/matlabr12/extern/include -Id:/matlabr12/rtw/c/src -Id:/matlabr12/rtw/c/libsrc -c d:/matlabr12/rtw/c/libsrc/rt_atan2.c

とする。このフォルダ$(MATLABROOT)/rtw/c/libsrc/にある全ての C ファイルのコンパイルを実行した後、次のコマンドを実行し、ランタイムライブラリ化する。

sh-elf-ar crv rtwlib.a *.o

sh-elf-ranlib rtwlib.a

これでランタイムライブラリの完成である。

SIMULINK 用ランタイムライブラリの自動生成

実は、上記のことをやらなくても libsrc の中には、SIMULINK 用ランタイムライブラリを自動生成するための Perl スクリプトがある。では、このスクリプトを SH2 用に変更するときの変更手順を以下に示す。
ここでは、mkliblcc.pl をベースに mklibsh2.pl を作成する際の変更点のみを示しておく。

GCC コンパイラを UNIX モードで使用するため

```
BEGIN {
    my ($scriptDir) = ($0 =~ /(.+)[¥¥¥/]/);
    push(@INC, "$scriptDir/../tools");
}
use win_subs;
SetupWinSysPath();
```

を削除する。

GCC コンパイラへのオプション設定として、-c オプションを追加する。

```
    local($cmd)    = "$rtw_cc $rtw_cflags $file";
```

を

```
    local($cmd)    = "$rtw_cc $rtw_cflags -c $file";
```

に変更

UNIX モードで動作させるため、パスの設定中で¥¥となっている箇所を全て/に置換する。
たとえば、

```
    push(@cfiles,"${dir}¥¥rt_enab.c");
```

から

```
    push(@cfiles,"${dir}/rt_enab.c¥");
```

変更する。

ただし、以下の 3 箇所の設定は、

```
$ofile =~ s/.*¥¥//g;
$ofile =~ s/¥.[cC]/.obj/g;
```

を

```
$ofile =~ s|/.*/||g;
$ofile =~ s/¥.[cC]/.o/g;
```

とする。

ライブラリリンクコマンドのオプションを GCC 用に変更する。

```
$cmd = "$rtw_libcmd /out:$lib $ofileList ";
```

から

```
$cmd = "$rtw_libcmd crv rtwlib.a $ofileList";
```

SH2 のコード自動生成スクリプトの設定

まず、自動生成スクリプトに必要なファイルは、tmf ファイル、tlc ファイル、c ファイルの３つである。それらのサンプルファイルは、環境別に

c:¥matlab6p1¥rtw¥c

の中にある。

tmf ファイルは、MAKE で実行する MAKEFILE スクリプトを作り出すファイルである。

tlc ファイルは、Target Language Compiler で記述されたファイルで、同様の名前の C 言語で記述された S-function 関数に対応し、SIMULINK 上で設定したパラメータなどを受け渡し、ターゲット用に最適な C で記述された関数を自動生成するためのファイルである。

c ファイルは、メインプログラムであり、SIMULINK の関数から生成された C 関数ファイルを統括する。ここで記述する内容は、割り込みなどによる時間管理関数の部分を追加する必要がある。またデバックのために、何か出力できるようにしておくと便利である。

自動生成スクリプトファイル（tmf）の設定

自動生成スクリプトファイルのサンプルプログラムは、c:¥MATLABR12¥rtw¥c

の中に grt フォルダ、dos フォルダの中に実行できるサンプルファイルがある。

SH2 は GCC で動作するため、最も一般的な grt フォルダの中にある。

特に、tmf ファイルは、

VisualC++用 grt_msvc.tmf, grt_vc.tmf

BorlandC 用 grt_bc.tmf

LCC 用　　grt_lcc.tmf

UNIX 用　　grt_unix.tmf

WatComC 用　grt_watc.tmf

など種類が多く、これらがそれぞれ機種やコンパイラの違いを吸収している個所である。

SH2 で使用するコンパイラは GNUPro の GCC である。GCC は、UNIX 用の C に近いた

め、ここでは、grt_unix.tmf をベースに変更していく。
grt_unix.tmf
grt.tlc
grt_main.c
の 3 つのファイルを参考に変更開発を行う。
まず、これらファイルを d:¥sh2 の中にコピーをしてファイル名を以下の様に変更する。
grt_unix.tmf　→grt_sh2.tmf
grt.tlc　　→grt_sh2.tlc
grt_main.c　　→sh2rt_main.c
この名前の変更に特別の理由はないが、名前の変更は、それぞれ 3 つのスクリプト内でも記述する場合があるので注意すること。
STEP１　TMF ファイルの設定
C プログラムをコンパイルできるようにするためのファイルを作成する。必要なリンクオプション、コンパイルオプションの指定はここで行う。コンパイラ、環境に合わせた MAKEFILE を作ることに相当する。
grt_sh2.tlc の SH2 用に変更するための注意点
SYS_TARGET_FILE = grt_sh2.tlc に変更する。
標準のままでは、|>MATLAB_ROOT<| が c:¥matlab6p1 と展開されてしまう。
そこで
MATLAB_ROOT　　= c:/matlab6p1
に変更する。
コンパイラとして
CC　　= sh-elf-gcc
LD　　= $(CC)
を設定する。
MATLAB__INCLUDES を以下のように追加変更する。通常のスクリプトでは、EXPAND_INCLUDES で必要なディレクトリを自動展開するが、この時のディレクトリ表示がバックスラッシュなってしまう。そのためここではあらかじめマニュアルで展開しておく。

```
MATLAB_INCLUDES = ¥
 -I$(MATLAB_ROOT)/simulink/include ¥
 -I$(MATLAB_ROOT)/extern/include ¥
 -I$(MATLAB_ROOT)/rtw/c/src ¥
 -I$(MATLAB_ROOT)/rtw/c/libsrc ¥
 -I$(MATLAB_ROOT)/toolbox/commblks/commmex ¥
 -I$(MATLAB_ROOT)/toolbox/dspblks/dspmex ¥
```

-I$(MATLAB_ROOT)/toolbox/dspblks/src/rt ¥
-I$(MATLAB_ROOT)/toolbox/dspblks/src/sim ¥
-I$(MATLAB_ROOT)/toolbox/fixpoint ¥
-I$(MATLAB_ROOT)/toolbox/fuzzy/fuzzy/src

コンパイルオプションの設定には、以下の設定を追加する。

OPTS = -m2 -nostartfiles -O0

SH2 では、printf 文が使えないので、-DHAVESTDIO の設定を解除する。

CPP_REQ_DEFINES = -DMODEL=$(MODEL) -DRT -DNUMST=$(NUMST) ¥
-DTID01EQ=$(TID01EQ) -DNCSTATES=$(NCSTATES) -DUNIX ¥
-DMT=$(MULTITASKING)
-DMT=$(MULTITASKING) -DHAVESTDIO

LDFLAGS には以下のオプションを追加する。

LDFLAGS = -nostartfiles

Perl による自動ランタイム生成するためには、

PERL = perl

とし、

MKLIB_PL = ../mklibsh2.pl

とする。この場合、mklibsh2.pl を SIMULINK の MDL ファイルのある同じフォルダにある必要がある。

LIBSRC = $(MATLAB_ROOT)/rtw/c/libsrc

を環境変数として追加する。また、RTWLIB は、カレントフォルダにあればよいので

RTWLIB = rtwlib.a

と指定する。

grt_unix.tmf の例であるが、WINDOWS 上では、sed などインストールする必要がある。また、これらで使用している START_EXPAND_LIBRARIES などのマクロによるディレクトリの展開では、フォルダの区切りが¥になってしまうなどの理由により、

ここでは、Perl 版の grt_lcc.tmf を参考に以下のように作り直す。

```
rtwlib.a : $(MAKEFILE) rtw_proj.tmw
 @echo "Creating $@ from $(MATLAB_ROOT)/rtw/c/libsrc/*.c"
 @¥rm -f $@
 @for file in $(MATLAB_ROOT)/rtw/c/libsrc/*.c; do ¥
    $(GCC_TEST_CMD) $$file $(GCC_TEST_OUT); ¥
    echo "$(CC) -c $(CFLAGS) $$file"; ¥
    $(CC) -c $(CFLAGS) $(GCC_WALL_FLAG_MAX) $$file; ¥
    ofile=`echo $$file | sed -e 's/¥.c/¥.o/g' -e 's|/.*/||g'`; ¥
    echo "ar r $@ $$ofile"; ¥
    ar r $@ $$ofile; ¥
    ¥rm -f $$ofile; ¥
 done
```

```
|>START_EXPAND_LIBRARIES<||>EXPAND_LIBRARY_NAME<|.a                    :
|>START_EXPAND_MODULES<||>EXPAND_MODULE_NAME<|.o
|>END_EXPAND_MODULES<|
@echo "### Creating $@ "
ar    r    $@    |>START_EXPAND_MODULES<||>EXPAND_MODULE_NAME<|.o
|>END_EXPAND_MODULES<|

|>END_EXPAND_LIBRARIES<|
```

```
#--------------------Platform dependent rules ---------------------------------
RTW_CC = sh-elf-gcc
RTW_CFLAGS = '$(CFLAGS)'
RTW_LIBCMD = sh-elf-ar

rtwlib.a : $(MAKEFILE) rtw_proj.tmw
 echo "###"
 echo "### Building $@"
 echo "###"
 echo "Creating $@ from $(MATLAB_ROOT)/rtw/c/libsrc/*.c"
 $(PERL)    $(MKLIB_PL)    $(LIBSRC)    $@    $(RTW_CC)    $(RTW_CFLAGS)
$(RTW_LIBCMD)
```

LIBS には、-T../shram.x を追加する。

LIBS = -lm $(EXT_LIB) $(S_FUNCTIONS_LIB) $(INSTRUMENT_LIBS) -T../shram.x

ソースファイルには、

#---------------------------- Source Files ----------------------------------

ASM_SRCS = shram.s

ASM_OBJS = shram.o

を追加する。

REQ_SRCS の grt_main.c を sh2rt_main.c に変更する。

REQ_SRCS　　=　$(MODEL).c　$(MODULES)　sh2rt_main.c　rt_sim.c　rtwlog.c　rt_nonfinite.c ¥

　　$(EXT_SRC)

ユーザーリンクプログラムとして、link.c,mess.c を加える。それぞれ link.c は GNUPro のバグ対策、mess.c は、シリアル通信用のプログラムである。

参考までに link.c の内容を示す。

```
int _link (char *old, char *new)
{
  return -1;
}
```

mess.c は、C:¥GCCDEVL¥7045F にあるベストテクノロジー社のシリアル通信用ライブラリプログラムを RealTimeWorkshop 用に変更したものである。基本的な変更箇所としては、

コンパイルエラーが出ないようにヘッダの順序を

```
#include   <stdarg.h>
#include   "vsscanf.c"
#include   "7045.h"
```

から

```
#include   "7045.h"
#include   <stdarg.h>
#include   "vsscanf.c"
```

だけである。また、カレントフォルダに link.c,mess.c をコピーする以外に、vsscanf.c もコピーしておく必要がある。

USER_SRCS = link.c mess.c

ASM_SRC,ASM_OBJS を追加したので反映するように OBJS と LINK_OBJES に書き加える。

```
OBJS        = $(SRCS:.c=.o) $(USER_OBJS) $(ASM_OBJS)
LINK_OBJS = $(SRCS:.c=.o) $(LOCAL_USER_OBJS) $(ASM_OBJS)
```

プログラム構築ルールに sh-elf-objcopy を追加し、MAKE を実行すると転送可能なバイナリー形式の出力を生成できるようにする。

```
$(PROGRAM) : $(OBJS) $(RTWLIB)
 $(LD) $(LDFLAGS) -o $@ $(LINK_OBJS) $(RTWLIB) $(LIBS)
 sh-elf-objcopy -O binary ../$(MODEL).exe ./$(MODEL).bin
 echo "### Created executable: $(MODEL)"
```

shram.s のコンパイル用に追加

```
%.o : ../%.s
 $(CC) -c $<
```

ここでも同様に自動展開スクリプトがあるが、これもバックスラッシュで生成されてしまうためマニュアルで記述する。

```
# Toolbox & Blocksets:

%.o :$(MATLAB_ROOT)/toolbox/commblks/commmex/%.c
 $(CC) -c $(CFLAGS) $<

%.o :$(MATLAB_ROOT)/toolbox/dspblks/dspmex/%.c
 $(CC) -c $(CFLAGS) $<

%.o :$(MATLAB_ROOT)/toolbox/dspblks/src/rt/%.c
```

```
 $(CC) -c $(CFLAGS) $<

%.o :$(MATLAB_ROOT)/toolbox/dspblks/src/sim/%.c
 $(CC) -c $(CFLAGS) $<

%.o :$(MATLAB_ROOT)/toolbox/fixpoint/%.c
 $(CC) -c $(CFLAGS) $<

%.o :$(MATLAB_ROOT)/toolbox/fuzzy/fuzzy/src/%.c
 $(CC) -c $(CFLAGS) $<

%.o : $(MATLAB_ROOT)/simulink/src/%.c
 $(CC) -c $(CFLAGS) $<

%.o : ../%.c
 $(CC) -c $(CFLAGS) $<
```

以上である。

Real-TimeWorkShop で一般的な、matlabR12¥rtw¥c¥grt 内にあるファイルを例に SH２用に設定の仕方を解説していく。
Step.1　まず、MATLAB¥RTW¥C¥GRT の中にあるファイル
grt_main.c,grt.tlc,grt_unix.tmf
の３つを SH2 用に変更すればよい。ここでは、ファイル名をそれぞれ
grt_main.c　　→grt_sh2main.c
grt.tlc　　→grt_sh2.tlc
grt_unix.tmf　→grt_sh2.tmf
と名前を変更したとし解説していく
grt_sh2.tmf の変更点
HOST は、異なるホストであるので ANY を指定する。BUILD は、Make 後にコンパイルするので no
HOST＝ANY
BUILD＝yes

SYS_TARGET_FILE=grt_sh2.tlc
CC　　= sh-elf-gcc
LD　　= $(CC)
# General User Options
OPTS = -m2 -nostartfiles -O0
LDFLAGS = -nostartfiles
コメントアウトと rtwlib.a の指定変更
#ifeq ($(OPT_OPTS),$(DEFAULT_OPT_OPTS))
#RTWLIB = $(MATLAB_ROOT)/rtw/c/lib/$(ARCH)/rtwlib.a
#else
RTWLIB　= ../rtwlib.a
#endif
LIBS に-T ../shram.x の追加
LIBS　　= -lm　$(EXT_LIB)　$(S_FUNCTIONS_LIB)　$(INSTRUMENT_LIBS) -T../shram.x
追加
ASM_SRCS = shram.s
ASM_OBJS = shram.o
USER_SRCS = link.c
に link.c を追加
OBJS　　= $(SRCS:.c=.o) $(USER_OBJS) $(ASM_OBJS)
LINK_OBJS = $(SRCS:.c=.o) $(LOCAL_USER_OBJS) $(ASM_OBJS)

STEP2　sh2rt_main.c を編集する。特に、サンプリング時間のため、割り込みなどの設定を行う必要がある。
インクルードに次のヘッダを追加する。
#ifndef _MESS_H_
#include "mess.h"
#endif
#ifndef _7045_H_
#include "7045.h"
#endif
ちなみに、ベストテクノロジ社で標準で添付される 7045.h、mess.c とは異なるので注意すること。
/*==================================*

```
 * Global data local to this module *
 *==================================*/
```

以下に以下のプログラムを追加する。これは、割り込みタイマプログラムである。なお割り込みタイマには、使用可能タイマを節約するため WDT を使用している。そのため、SH2 についている標準のタイマ２チャンネル分は他の用途で使用できる。
また、SIMULINK のブロックの大きさによるが、ここでは、サンプリングタイム以内に命令を処理し終了する必要がある。そのため、サンプリングタイムは、もっとも早くても 0.1 秒程度である。

```
/*----------------------------------------------------------*/
/*  ステータスレジスタ割り込みマスク処理                    */
/*----------------------------------------------------------*/
void SetSRReg (volatile _WORD m)
{
   volatile _LONG u;

   m <<= 4;
   m &= 0x00f0;
   asm volatile ("    stc     sr,%0 ":"=r"(u):);
   u &= 0xffffff0f;
   u |= m;
   asm volatile ("    ldc     %0,sr "::"r"(u));
}
//---------------------------------------
// 割り込みベクターを内蔵 RAM エリアに確保
//---------------------------------------
typedef void (*SHfp)(void);
struct {
   SHfp No[256];
} *SHVectorTable = (void *)0x00400000;
// shram.s の L_vector_table_top: .long 0x00400000 のこと
void *usrData;
void (*usrIntrHandler)(void *);
volatile unsigned short SHtcnt=0;
volatile unsigned short SHtscr=0xa539;
volatile double SHsampling = 0.0;
volatile int SHloop=0;
```

```
void int_iti(void) __attribute__((interrupt_handler));
void int_iti(void) {
//SHtcnt 0x5a00+ (slow) 0(*default) 1 2 ・・・254　255(fast)
// SHtcnt=0 の時の割り込み周期
//TCSR 0xa53f 8192     74.9ms
//TCSR 0xa53e 4096     37.5ms
//TCSR 0xa53d 1024     9.36ms
//TCSR 0xa53c   512     4.68ms
//TCSR 0xa53b   256     2.34ms
//TCSR 0xa53a   128     1.17ms
//TCSR 0xa539    64     0.59ms
//(1/CLK)*(256-tcnt)*(64～8192) 0.5851msec・・・　74.8983msec
 if (SHloop > 0) {
  WDT.WRITE.TCSR = (unsigned short)0x5a00;
  WDT.READ.TCSR.BIT.OVF;
  WDT.WRITE.TCSR = SHtscr;
  SHloop--;
  return;
 }

 WDT.WRITE.TCSR = (unsigned short)0x5a00+(SHtcnt % 0x00ff);
 WDT.READ.TCSR.BIT.OVF;
 WDT.WRITE.TCSR = SHtscr;
 if (SHtcnt > 0xff) SHloop = (unsigned short)(SHtcnt/256.);
//   SendMessage1 ("Tick!!¥r¥n");
 (*usrIntrHandler)(usrData);
 return;
}
int sh2rt_InitTimer(void (*handler)(), void *data, double stepSize)
{
// SHTCNT=0 の時の割り込み周期
//SHTCSR 0xa53f 8192     74.9ms
//SHTCSR 0xa53e 4096     37.5ms
//SHTCSR 0xa53d 1024     9.36ms
//SHTCSR 0xa53c   512     4.68ms
//SHTCSR 0xa53b   256     2.34ms
```

```
//SHTCSR 0xa53a    128     1.17ms
//SHTCSR 0xa539     64     0.59ms
//(1/CLK)*(256-SHTCNT)*(64～8192) 0.5851msec・・・　74.8983msec
// stepSize が 74.9mS より大きいとき 8192 と 74.9 と 255 で調整
// 以下の時は場合分けと 255 で調整
 SHtscr=0xa539;
 SHtcnt=(int)(stepSize*28000000./64.+.5);
 SHsampling = SHtcnt*64/28000000.;
 SHloop = 0;
 if(stepSize*28000000. > (64.*256.)) {
  SHtscr=0xa53a;
  SHtcnt=(unsigned short)(stepSize*28000000./128.+.5);
  if (SHtcnt > 255) SHloop =(unsigned short)(SHtcnt/256.);
  else SHloop = 0;
  SHsampling = ((SHtcnt % 0xff)+1+SHloop*256.)*128./28000000.;
 }
 if(stepSize*28000000. > (128.*256.)) {
  SHtscr=0xa53b;
  SHtcnt=(unsigned short)(stepSize*28000000./256.+.5);
  if (SHtcnt > 255) SHloop =(unsigned short)(SHtcnt/256.);
  else SHloop = 0;
  SHsampling = ((SHtcnt % 0xff)+1+SHloop*256.)*256./28000000.;
 }
 if(stepSize*28000000. > (256.*256.)) {
  SHtscr=0xa53c;
  SHtcnt=(unsigned short)(stepSize*28000000./512.+.5);
  if (SHtcnt > 255) SHloop =(unsigned short)(SHtcnt/256.);
  else SHloop = 0;
  SHsampling = ((SHtcnt % 0xff)+1+SHloop*256.)*512./28000000.;
 }
 if(stepSize*28000000. > (512.*256.)) {
  SHtscr=0xa53d;
  SHtcnt=(unsigned short)(stepSize*28000000./1024.+.5);
  if (SHtcnt > 255) SHloop =(unsigned short)(SHtcnt/256.);
  else SHloop = 0;
  SHsampling = ((SHtcnt % 0xff)+1+SHloop*256.)*1024./28000000.;
```

```
 }
 if(stepSize*28000000. > (1024.*256.)) {
  SHtscr=0xa53e;
  SHtcnt=(unsigned short)(stepSize+28000000./4096.+.5);
  if (SHtcnt > 255) SHloop =(unsigned short)(SHtcnt/256.);
  else SHloop = 0;
  SHsampling = ((SHtcnt % 0xff)+1+SHloop*256.)*4096./28000000.;
 }
 if(stepSize*28000000. > (4096.*256.)) {
  SHtscr=0xa53f;
  SHtcnt=(unsigned short)(stepSize*28000000./8192.+.5);
  if (SHtcnt > 255) SHloop =(unsigned short)(SHtcnt/256.);
  else SHloop = 0;
  SHsampling = ((SHtcnt % 0xff)+1+SHloop*256.)*8192./28000000.;
 }
 SHVectorTable->No[152] = (SHfp)int_iti;// WDT
 WDT.WRITE.TCSR = SHtscr;
 WDT.WRITE.TCSR = (unsigned short)0x5a00+(SHtcnt % 0x00ff);
 usrData = data;
 usrIntrHandler = handler;
 INTC.IPRH.WORD=0xf000;//優先順位=14
 SendMessage1 ("loop[%d] TSCR[%x] cnt[%d]¥r¥n",SHloop,SHtscr,(int)SHtcnt);
 return(1);
}
```

メイン文の中の S = MODEL();が設定されエラーチェックが終わった後に、

```
    InitSCI1 (br57600); //  シリアル(SCI1)初期化  57600[bps]
    if (!sh2rt_InitTimer(rt_OneStep, S, ssGetStepSize(S))) {
  SendMessage1("Failed to initialize timer¥r¥n");
  return(EXIT_FAILURE);
    }
 SendMessage1 ("Welcome to RealTime Workshop(for SH2) by Gerox(c)2001¥r¥n");
 SendMessage1("Actual Sampling interval %f[sec]¥r¥n",SHsampling);
 SetSRReg (0);
```

のコードを追加する。

割り込みベースで rtOneStep が動くためメインループの中に rtOneStep は必要ない。コメントアウトしておく。

```
    while (!GBLbuf.stopExecutionFlag &&
           (ssGetTFinal(S) == RUN_FOREVER ||
            ssGetTFinal(S)-ssGetT(S) > ssGetT(S)*DBL_EPSILON)) {

                if (ssGetStopRequested(S)) break;
    }
```

の中に rtOneStep があったら削除する。

STEP3　RealTimeWorkshop で SH2 オプションが選択できるようにする。

grt_sh2.tlc の設定

スクリプトの最初の行と２行目を以下のように変更する。

%% SYSTLC: IKKO Generic Real-Time Target for SH2 ¥

%%　　TMF: grt_sh2.tmf MAKE: make_rtw EXTMODE: ext_comm

次に rtwgensettings.BuildDirSuffix の設定を_grt_sh2_rtw と変更する。

```
  rtwgensettings.BuildDirSuffix = '_grt_sh2_rtw';
```

この様にすることで、自動生成されたソースファイルは、このサフィックスのフォルダに展開される様になる。

取り合えずシステム設定はこれで終了である。

以上、3 つのファイルの入ったフォルダにカレントディレクトリを移動し、SIMULINK を起動、RealTimeWorkshop のオプションを選択すると、SH2 用が選択できるようになる。

これで SIMULINK ファイルを一応、C コードに自動生成できるようになる。

自動生成・コンパイルの実行法

自動生成・コンパイル実行のためには、いくつか手順が必要になる。概略を説明すると、まず、MATLAB を起動し SIMULINK を起動、SIMULINK ダイアグラムから C コードを自動生成する。DOS プロンプトを起動し、自動生成された C コードと同じフォルダに移動する。この際の移動には、cd コマンドなどを使う必要がある。

次に、make -f モデル名.mk

としてコンパイルし、モデル名.bin と言うファイルが生成される。

次に、GNUPro を起動その中から HyperTerm を起動。

SH2 を起動しロード受付画面が出てきたら、x-modem でモデル名.bin を転送する。転送し終わると自動的に実行される。

一応は、実行されるが、SIMULINK 上でハードウェアを実際に操作できなければ無意味である。そこでハードウェアにアクセスするための記述をする必要がある。ハードウェアをアクセスするには、フルインライン関数で記述する必要がある。

本格的な S 関数の作り方は、マニュアル等に詳しく書いてあるので詳しくは述べない。ここでは、特に RealTimeWorkshop に特化した S 関数の作り方を述べる。
RealTimeWorkshop でハードウェアをアクセスためには、C 言語だけの記述ではだめで、C 言語を自動生成するときに使用される TLC 言語で記述する必要がある。
つまり、RealTimeWorkShop では、
S 関数は、TLC とのパラメータの受け渡し用として
また、TLC で実際のハードウェアとのアクセスを記述する。
ハードウェアの種類としては、
CASE a
　AD コンバータ、ディジタル入力など、外部からデータを得るタイプ
CASE b
　シリアル出力、PWM 出力、ディジタル出力など、外部にデータを出力するタイプ
などがある。
AD コンバータ用プログラムの設計例
S-関数の記述で注意する点、
SIMULINK 上で設定したいパラメータとしては、
何チャンネル分使用するのか？
サンプリングタイムの設定は？
などがある。
S 関数ファイルと SIMULINK ブロックとのパラメータのやり取りは、
mdlCheckParameters(SimStruct *S)関数でパラメータのチェック
mdlInitializeSizes(SimStruct *S)では、パラメータチェックに基づき
出力ポートの幅（ssSetOutputPortWidth(S, 0, NUM_CHANNELS);）
を決定する。なお、Input が無いのに ssSetInputPortWidth を指定（例え、0 と書いても）クラッシュするので注意すること。
mdlOutputs(SimStruct *S, int_T tid)関数は、SIMULINK でシミュレーションする場合に呼び出される関数である。ここを設定しておかないと、RealTimeWorkshop では動作するが、シミュレーションを実行するとクラッシュしてしまう。クラッシュを防ぐため、出力ポートの幅に合わせ何らかの値を出力するようにする必要がある。ここでは、0 を出力する例を示す。

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T portWidth=ssGetOutputPortWidth(S,0);
    real_T *y = ssGetOutputPortRealSignal(S,0);
    uint_T   i;
    for (i = 0; i < portWidth; i++) {
```

```
            y[i] = 0.0;
    }
}
```

mdlRTW(SimStruct *S)関数内で受け渡しパラメータの設定

```
    (void)ssWriteRTWParamSettings(S, NUM_PARAMS,
    SSWRITE_VALUE_NUM, "Channel",(double)NUM_CHANNELS,
    SSWRITE_VALUE_NUM, "sampleTime",(double)SAMPLE_TIME);
```

とする。この場合、2 つしかないがこの時の"Channel", "sampleTime"が TLC 言語でのパラメータアクセスのキーワードとなる。

TLC ファイルの設定

TLC ファイルでは、実際のハードウェアアクセスに関する記述を行う。例えば、SH2 では、最大で 8 チャンネルの AD コンバータがある。しかし、0～3 チャンネルまでのアクセスは容易であるが、4～7 チャンネルのアクセスは、若干異なるのでその点の配慮が必要となる。また、TLC では、明示的に C コードをインライン展開するため配慮して各必要がある。

%function BlockInstanceSetup(block, system) void

ここでは、AD コンバータのブロックが１つのモデル中１つしか存在しないようにするため以下のチェックを行う。

```
  %if EXISTS("Drt_sh2ad")
    %assign errTxt = "Only 1 sh2ad block is allowed in " ...
      "model: %<CompiledModel.Name>."
    %exit RTW Fatal: %<errTxt>
  %endif
  %%
```

インクルードファイルの設定

```
  %if !EXISTS("Drt_sh2ad")
    %assign ::Drt_sh2ad = 1
    %openfile buffer
    #include "sh2ad.h"
    %closefile buffer
    %<LibCacheIncludes(buffer)>
  %endif
  %%
```

スタート関数では、初期化する命令を記述する。ハードに関する個所は、C 言語での記述とまったく同じである。

%function Start(block, system) Output

```
/* %<Type> Block: %<Name> (%<ParamSettings.FunctionName>) */
```

```
{
  // A/D モジュール 0(AN0～3)設定
  AD.CSR0.BIT.ADST = 0;       // A/D 変換停止
  AD.CSR0.BIT.ADIE = 0;       // 割り込み禁止
  AD.CSR0.BIT.SCAN = 1;       // スキャンモード
  AD.CSR0.BIT.CKS = 0;        // 変換時間 266 ステート
  AD.CSR0.BIT.CH = 3;         // AN0～AN3 チャネル
  AD.CSR0.BIT.ADST=1;         // AD 変換開始
  // 以後 AN0～AN3 はデータレジスタのみ読み出せば OK
}
%endfunction %% Start
```

出力関数は、チャンネル数により生成するソースコードを配慮しながら記述する必要がある。LibDataOutputPortWidth(0)関数により出力チャンネル数を取り出す。

%foreach 命令は繰り返しを行う命令である。C 言語の FOR 文とは異なり、C 言語に変換された場合、条件設定にもよるが、ループせずインライン展開するので注意が必要である。

%if も C 言語と同じである。

出力は、LibBlockOutputSignal(0, "", "", idx)で idx の値によりどのチャンネルかが変化する。%if 文により、チャンネル数が 0 から 3 と 4 から 7 の時の場合分けを行っている。

```
%function Outputs(block, system) Output
%%
/* %<Type> Block: %<Name> (%<ParamSettings.FunctionName>) */
{
 %assign numChannels = LibDataOutputPortWidth(0)
 %foreach idx=numChannels
  %assign y = LibBlockOutputSignal(0, "", "", idx)
  %if idx == 0
   %<y>=(AD.DRA0.WORD >> 6);
  %endif
  %if idx == 1
   %<y>=(AD.DRB0.WORD >> 6);
  %endif
  %if idx == 2
   %<y>=(AD.DRC0.WORD >> 6);
  %endif
  %if idx == 3
   %<y>=(AD.DRD0.WORD >> 6);
```

```
  %endif
  %if (idx == 4)||(idx == 5)||(idx == 6)||(idx == 7)
   {
    int module;
     module = (%<idx+1> > 3) ? 1 : 0;   // モジュール選択 (0 or 1)
     while (AD2.CSR[module].BIT.ADST) ;   // 変換完了待ち
     AD2.CSR[module].BYTE = %<idx+1> % 4;          // 低速・単一モード・チャネル指定
(0～3)
     AD2.CSR[module].BIT.ADST = 1;            // A/D 変換開始
      while (!AD2.CSR[module].BIT.ADF) ;   // 変換完了待ち
     %<y>= AD2.AN[%<idx+1>].WORD10.DATA;
    }
   %endif
 %endforeach
}
%endfunction %% Outputs
```

以下に出力が 1 チャンネルの時の C 言語出力結果と 8 チャンネルの時の C 言語生成結果を示す。

SIMULNK 上でブロックを作りパラメータを割り当てる。

ちなみに、MATLAB6.0 で SIMULINK をする場合、設定がうまくできない。MATLAB コマンドプロンプトで、set(0,'language','english');として英語モードにするとうまく行く。

PWM 出力用プログラムの設計例

S-関数の記述で注意する点、

SIMULINK 上で設定したいパラメータとしては、

最大カウント数

周波数

サンプリングタイム

などがある。最大カウント数は、16 ビットで 0 から 65535 までで、周波数は、PWM でベースとなる周波数である。なおプログラム簡便のため、チャンネル数は、12 チャンネル固定とし周波数も全て共通周波数となるようにする。

S 関数ファイルとのパラメータのやり取りは、

mdlCheckParameters(SimStruct *S)関数でパラメータのチェック

mdlInitializeSizes(SimStruct *S)では、パラメータチェックに基づき

入力ポートの幅（ssSetInputPortWidth(S, 0, 12);）

を決定する。なお、先ほどと同様に output が無いのに ssSetOutputPortWidth を指定（例

え、0 と書いても）クラッシュするので注意すること。
mdlOutputs(SimStruct *S, int_T tid)関数は、SIMULINK でシミュレーションする場合に呼び出される関数である。この場合、出力は無いので何も記述しなくてもよい。

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
}
```

mdlRTW(SimStruct *S)関数内で受け渡しパラメータの設定

```
    (void)ssWriteRTWParamSettings(S, NUM_PARAMS,
    SSWRITE_VALUE_NUM, "Maxval",(double)NUM_MAXVAL,
    SSWRITE_VALUE_NUM, "Freqval",(double)NUM_FREQVAL,
    SSWRITE_VALUE_NUM,  "sampleTime",(double)SAMPLE_TIME);
```

とする。この場合、3 つあるが、この時の"Maxval", "Freqval", "sampleTime"が TLC 言語でのパラメータアクセスのキーワードとなる。
TLC ファイルの設定
TLC ファイルでは、主に、初期化のためのルーチン、実行のためのルーチンの 2 つを記述する。PWM での初期化ルーチンには、ポートの設定、ポート入出力、デフォルトの周波数、PWM のデューティ比の設定を行う。
%function BlockInstanceSetup(block, system) void
ここでは、PWM のブロックが１つのモデル中１つしか存在しないようにするため以下のチェックを行う。

```
  %if EXISTS("Drt_sh2pwm12")
    %assign errTxt = "Only 1 sh2pwm12 block is allowed in " ...
      "model: %<CompiledModel.Name>."
    %exit RTW Fatal: %<errTxt>
  %endif
  %%
```

インクルードファイルの設定

```
  %if !EXISTS("Drt_sh2pwm12")
    %assign ::Drt_sh2pwm12 = 1
    %openfile buffer
    #include "sh2pwm12.h"
    %closefile buffer
    %<LibCacheIncludes(buffer)>
  %endif
  %%
```

スタート関数では、初期化する命令を記述する。ハードに関する個所は、C 言語での記述と

まったく同じである。
ここでは、LibDataInputPortWidth(0)でチャンネル数、
CAST("Number",SFcnParamSettings.Freqval)で周波数、
CAST("Number", SFcnParamSettings.Maxval)で最大カウント数を SIMULINK のダイアグラムから取り出している。これらは、TLC により C コードが自動的に定数や変数に置換わり生成される。この時の Freqval,Maxval が S 関数内で指定した ssWriteRTWParamSettings のキーワードである。

```
%function Start(block, system) Output
  /* %<Type> Block: %<Name> (%<ParamSettings.FunctionName>) */
{
 %assign channel = LibDataInputPortWidth(0)
 %assign freqval = CAST("Number",SFcnParamSettings.Freqval)
 %assign maxval = CAST("Number", SFcnParamSettings.Maxval)
 PFC.PEIOR.BIT.B14=1;      // *0:PE14, TIOC4C, DACK0, ~AH
 PFC.PEIOR.BIT.B12=1;      // *0:PE12, TIOC4A
 PFC.PEIOR.BIT.B10=1;      // *0:PE10, TIOC3C
 PFC.PEIOR.BIT.B8 =1;      // *0:PE8,   TIOC3A
 PFC.PEIOR.BIT.B7 =1;      // *0:PE7,   TIOC2B
 PFC.PEIOR.BIT.B6 =1;      // *0:PE6,   TIOC2A
 PFC.PEIOR.BIT.B5 =1;      // *0:PE5,   TIOC1B
 PFC.PEIOR.BIT.B4 =1;      // *0:PE4,   TIOC1A
 PFC.PEIOR.BIT.B3 =1;      // *0:PE3, TIOC0D, DRAK1
 PFC.PEIOR.BIT.B2 =1;      // *0:PE2, TIOC0C, ~DREQ1
 PFC.PEIOR.BIT.B1 =1;      // *0:PE1, TIOC0B, DRAK0
 PFC.PEIOR.BIT.B0 =1;      // *0:PE0, TIOC0A, ~DREQ0
 PFC.PECR1.BIT.MD8       = 1;      // *0:PE8, 1:TIOC3A
 PFC.PECR1.BIT.MD10      = 1;      // *0:PE10, 1:TIOC3C
 PFC.PECR1.BIT.MD12      = 1;      // *0:PE12, 1:TIOC4A
 PFC.PECR1.BIT.MD14      = 1;      // *0:PE14, 1:TIOC4C, 2,DACK0, 3:~AH
 PFC.PECR2.BIT.MD7       = 1;      // *0:PE7, 1:TIOC2B
 PFC.PECR2.BIT.MD6       = 1;      // *0:PE6, 1:TIOC2A
 PFC.PECR2.BIT.MD5       = 1;      // *0:PE5, 1:TIOC1B
 PFC.PECR2.BIT.MD4       = 1;      // *0:PE4, 1:TIOC1A
 PFC.PECR2.BIT.MD3       = 1;      // *0:PE3, 1:TIOC0D, 2:DRAK1
 PFC.PECR2.BIT.MD2       = 1;      // *0:PE2, 1:TIOC0C, 2:~DREQ1
```

```
 PFC.PECR2.BIT.MD1       = 1;      // *0:PE1, 1:TIOC0B, 2:DRAK0
 PFC.PECR2.BIT.MD0       = 1;      // *0:PE0, 1:TIOC0A, 2:~DREQ0

 MTU.TSYR.BYTE=0x00;
 MTU.TSTR.BYTE=0x00;
 PWM.TOER.BYTE=0xd2;
 PWM.TCR3.BYTE=0x20+%<freqval-1>;
 PWM.TIOR3.BYTE.H=0x65;
 PWM.TIOR3.BYTE.L=0x65;
 PWM.TCNT3=0;
 PWM.TGR3A=%<maxval>;
 PWM.TGR3B=0;//pwm9;
 PWM.TGR3C=%<maxval>;
 PWM.TGR3D=0;//pwm10;
 PWM.TMDR3.BYTE=0xc2;//PWM mode 1
 PWM.TCR4.BYTE=0x60+%<freqval-1>;//30
 PWM.TIOR4.BYTE.H=0x65;
 PWM.TIOR4.BYTE.L=0x65;
 PWM.TCNT4=0;
 PWM.TGR4A=%<maxval>;
 PWM.TGR4B=0;//pwm11;
 PWM.TGR4C=%<maxval>;
 PWM.TGR4D=0;//pwm12;
 PWM.TMDR4.BYTE=0xc2;//PWM mode 1
// channel 0
 MTU0.TCR.BYTE=0x60+%<freqval-1>;
 MTU0.TIOR.BYTE.H=0x22;
 MTU0.TIOR.BYTE.L=0x22;
 MTU0.TCNT=0;
 MTU0.TGRA=0;//pwm1;
 MTU0.TGRB=0;//pwm2;
 MTU0.TGRC=0;//pwm3;
 MTU0.TGRD=0;//pwm4;
 MTU0.TMDR.BYTE=0xc3; //1100 0011

// channel 1
```

```
 MTU1.TCR.BYTE=0x60+%<freqval-1>;
 MTU1.TIOR.BYTE=0x22;
 MTU1.TCNT=0;
 MTU1.TGRA=0;//pwm5;
 MTU1.TGRB=0;//pwm6;
 MTU1.TMDR.BYTE=0xc3; //1100 0011

// channel 2
 MTU2.TCR.BYTE=0x60+%<freqval-1>;
 MTU2.TIOR.BYTE=0x22;
 MTU2.TCNT=0;
 MTU2.TGRA=0;//pwm7;
 MTU2.TGRB=0;//pwm8;
 MTU2.TMDR.BYTE=0xc3; //1100 0011
 MTU.TSYR.BYTE=0xc7;
 MTU.TSTR.BYTE=0xc7;//11000111
}
%endfunction %% Start
```

出力関数は、ハードウェアに対する命令を実行する。その際、SIMULINK 上から入力される最大カウントパラメータは、簡便のため 0 から 100 の数にノーマライズするようにしている。そのため、この関数内では、出力値が 0 から 100 の範囲内にあるかチェックを行い。データが必ず 0 から 100 の範囲内になるようにしている。また、PWM の周波数の変更は、サンプル実行時に毎回更新するわけではなく、１ポートでも出力が変化した場合のみ全ての PWM 周期を再設定するように記述している。そのため、static 変数を使い、前回とのデータを比較違いがあった場合のみ実行するようになっている。本当は、変更したポートのみ PWM 周期を変更しても良いが、チャンネルごとに事情が異なるなどプログラムが煩雑になるためそのような方策は採用していない。

```
%function Outputs(block, system) Output
   %%
   /* %<Type> Block: %<Name> (%<ParamSettings.FunctionName>) */
{
 %assign portWidth = LibDataInputPortWidth(0)
 %assign freqval = CAST("Number",SFcnParamSettings.Freqval)
 %assign maxval = CAST("Number", SFcnParamSettings.Maxval)
 static double ppwm[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
```

```
 static double pwm[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
 int check = 0;
 %foreach idx=portWidth
  pwm[%<idx>]=%<LibBlockInputSignal(0,"","",idx)>;
  if(pwm[%<idx>] >100.) pwm[%<idx>] = 100.;
  else if(pwm[%<idx>] < 0.) pwm[%<idx>] = 0.;
  pwm[%<idx>] *=(%<maxval>/100.);
  if(check == 0) if(ppwm[%<idx>] != pwm[%<idx>]) check++;
 %endforeach
 if(check != 0) {
  %foreach idx=portWidth
   ppwm[%<idx>] = pwm[%<idx>];
  %endforeach
  MTU.TSYR.BYTE=0x00;
  MTU.TSTR.BYTE=0x00;
  PWM.TOER.BYTE=0xd2;
  PWM.TCR3.BYTE=0x20+%<freqval-1>;
  PWM.TIOR3.BYTE.H=0x65;
  PWM.TIOR3.BYTE.L=0x65;
  PWM.TCNT3=0;
  PWM.TGR3A=%<maxval>;
  PWM.TGR3B=(unsigned short)pwm[8];
  PWM.TGR3C=%<maxval>;
  PWM.TGR3D=(unsigned short)pwm[9];
  PWM.TMDR3.BYTE=0xc2;//PWM mode 1
  PWM.TCR4.BYTE=0x60+%<freqval-1>;//30
  PWM.TIOR4.BYTE.H=0x65;
  PWM.TIOR4.BYTE.L=0x65;
  PWM.TCNT4=0;
  PWM.TGR4A=%<maxval>;
  PWM.TGR4B=(unsigned short)pwm[10];
  PWM.TGR4C=%<maxval>;
  PWM.TGR4D=(unsigned short)pwm[11];
  PWM.TMDR4.BYTE=0xc2;//PWM mode 1
 // channel 0
  MTU0.TCR.BYTE=0x60+%<freqval-1>;
```

```
  MTU0.TIOR.BYTE.H=0x22;
  MTU0.TIOR.BYTE.L=0x22;
  MTU0.TCNT=0;
  MTU0.TGRA=(unsigned short)pwm[0];
  MTU0.TGRB=(unsigned short)pwm[1];
  MTU0.TGRC=(unsigned short)pwm[2];
  MTU0.TGRD=(unsigned short)pwm[3];
  MTU0.TMDR.BYTE=0xc3; //1100 0011
//   channel 1
  MTU1.TCR.BYTE=0x60+%<freqval-1>;
  MTU1.TIOR.BYTE=0x22;
  MTU1.TCNT=0;
  MTU1.TGRA=(unsigned short)pwm[4];
  MTU1.TGRB=(unsigned short)pwm[5];
  MTU1.TMDR.BYTE=0xc3; //1100 0011
// channel 2
  MTU2.TCR.BYTE=0x60+%<freqval-1>;
  MTU2.TIOR.BYTE=0x22;
  MTU2.TCNT=0;
  MTU2.TGRA=(unsigned short)pwm[6];
  MTU2.TGRB=(unsigned short)pwm[7];
  MTU2.TMDR.BYTE=0xc3; //1100 0011
  MTU.TSYR.BYTE=0xc7;
  MTU.TSTR.BYTE=0xc7;//11000111
 }
}
%endfunction %% Outputs
```

C:¥GCCDEVL¥7045F にあるソースコード

mess.c

shram.x

shram.s

vsscanf.c

7045.h

をカレントフォルダにコピーする。RealTimeWorkshop で使えるようにするために以下の

改造をする。

mess.c

インクルードの順序を変更する。

```
#include   <stdarg.h>
#include   "vsscanf.c"
#include "7045.h"
```

から

```
#include "7045.h"
#include   <stdarg.h>
#include   "vsscanf.c"
```

へ変更

shram.x

変更なし

shram.s

変更なし

vsscanf.c

変更なし

7045.h

asm 命令など機種依存の含む関数 SetSRReg があるため、CacheOn,CacheOff は、多重定義のため#if 0 で無効にする。また、割り込み命令に関しては、SIMULINK で DLL を作るとき構造体が必要になるため、VisualC＋＋であるかどうかで区別する。

```
#if 0
/*------------------------------------------------------------*/
/*  ステータスレジスタ割り込みマスク処理                      */
/*------------------------------------------------------------*/
void SetSRReg (volatile _WORD m)
…
void CacheOff (void)
{
   CCR.BIT.CECS1 = 0;   // キャッシュコントローラ OFF
}
#endif
#ifndef _MSC_VER
/*------------------------------------------------------------*/
/*  割り込み関数プロトタイプ宣言                              */
/*------------------------------------------------------------*/
・・・
#endif
```