ユーザ・カスタマイズ・モニタ機能の使用方法について

(第 4 版)

目 次

本書は、メモリバス以外に接続されたシリアル・フラッシュメモリなどのフラッシュメモリに対して、CSIDE から書き込み/読み出しを行うユーザ・カスタマイズ・モニタ機能について説明します。

1. 概要

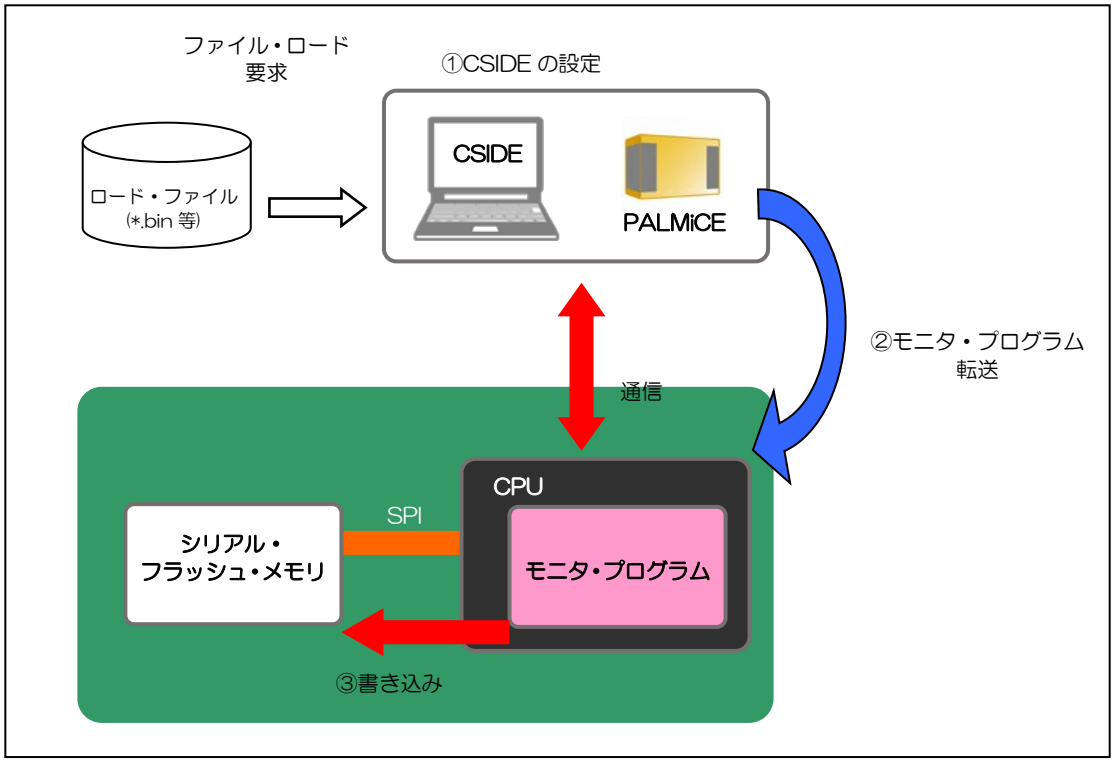
CSIDE からの通常のメモリアクセスは、CSIDE が CPU のリード命令やライト命令を使用して、メモリ空間(メモリバス)に接続されたデバイスの読み書きを行っています。このため、メモリバス以外に接続されたシリアル・フラッシュメモリなど、CPU のリード/ライト命令では直接読み書きが行えないメモリデバイスについては、CSIDE から通常のメモリアクセスでの読み書きができません。

このようなフラッシュメモリへの書き込み/読み出しを行う手段として、CSIDE ではユーザ・カスタマイズ・モニタ機能を提供しています。本機能を利用することで、別途ツールをご用意頂くことなく、CSIDE からのフラッシュメモリへの書き込み/読み出しを行う事を可能にします。

1.1 ユーザ・カスタマイズ・モニタの基本構造

ユーザ・カスタマイズ・モニタ機能は、ターゲット上の RAM 空間に CSIDE 上からモニタ・プログラムを転送し、そのモニタ・プログラムを動作させ通信を行う事で、フラッシュメモリの書き込み/読み出しを実現します。

■ **例)シリアル・フラッシュ・メモリへのファイル・ロード**



※ファイル・ロード例

①モニタ・プログラムの登録や設定、書き込みを行うプログラムファイルの設定など、フラッシュメモリに書込むための設定を CSIDE 上で行います。

②モニタ・プログラムをターゲットボードの RAM 空間にダウンロードします。

③CSIDE 上でファイル・ロードが実行されると、CSIDE はモニタ・プログラムと通信を行い、モニタ・プログラムを介して対象となるフラッシュメモリの書き込みを行います。

読み出しの場合も同様の仕組みで、メモリ内容の読み出しを行います。

1.2 対応 CSIDE

ユーザ・カスタマイズ・モニタ機能の対応 CSIDE は以下となっています。

- **CSIDE for PALMiCE3 SH V6.09.00 以降**
- **CSIDE for PALMiCE3 SHA V6.09.00 以降**
- **CSIDE for PALMiCE3 ARM V6.22.00 以降**

1.3 本書をお読み頂く上での注意事項

- 本書の内容は、CSIDE のセットアップ、及び、ターゲットボードの起動処理(ターゲットシステムの更新)を行える状態からの設定手順について解説を行います。予め CSIDE 付属のユーザーズ・マニュアルを参照して、CSIDE のセットアップや PALMiCE3 のドライバをインストールなど、ご使用頂くための準備を行っておいてください。
- 本資料は、CSIDE for PALMiCE3 ARM を使用した画面で説明を行います。別機種でご利用の場合、画面表示が異なる箇所がありますが、ユーザ・カスタマイズ・モニタ機能の設定については、共通の設定画面になっています。
- CSIDE の操作について不明な点がありましたら、CSIDE ヘルプやチュートリアルをご参照ください。

1.4 ユーザ・カスタマイズ・モニタ機能使用時の注意事項

ターゲット・システムの状態

ユーザ・カスタマイズ・モニタ機能は、モニタ・プログラムをターゲット・システム上の RAM 空間に転送できなければ使用することができません。このため、本機能を使用する前に、モニタ・プログラムを配置する RAM 領域をマクロファイル等で初期化を行い、CSIDE から RAM 領域の書き込みが行える状態にしておいてください。

モニタ動作時のレジスタについて

モニタ・プログラムの動作時には、プログラムカウンタはモニタ・プログラム関数の先頭、スタックポインタは、「2.3 モニタ・プログラムの変数」の_ucm_stack[]配列の最後のアドレスに設定されて動作します。それ以外のレジスタについては、動作時点のレジスタ値を保持した状態で動作するため、レジスタの状態によっては正しくモニタ・プログラムが動作しない可能性があります。

このため、ユーザ・カスタマイズ・モニタ機能を使用する際には事前にハードウェアの初期化を行い、CPU がリセット状態に近い状況でご使用ください。

ファイル・ロード/ライト時のアドレスについて

CSIDE では、ファイル・ロードやファイル・ライトで使用するロード(ライト)アドレスは 0 番地がフラッシュメモリアドレスの先頭番地として、モニタとやり取りを行います。ELF/DWARF2 など CPU アドレス情報がある場合、例えば CPU アドレスで 0x1800_0000 からフラッシュの先頭で配置されている場合、モニタ・ファイルのアドレス変換マスク定義で 0 番地台に変換する必要があります。

書き込み後のデータチェックについて

ユーザ・カスタマイズ・モニタ機能では、セクタ・ブロック単位で書き込みを行った後に、チェック・サム関数を動作させて書き込み内容の確認を行っています。

しかし、モニタ・プログラム・ソース内に定義するセクタ・ブロック情報やアドレス変換マスク等の使用するデバイス情報に誤りがある場合、フラッシュメモリの先頭に上書きされる事があります。この場合、チェック・サムエラーとはなりませんのでご注意ください。なお、フラッシュメモリのセクタやマスク情報の設定は、「2.3 モニタ・プログラムの変数」の変数で行います。

割り込み動作等について

ファイル・ロード等でモニタ・プログラムが動作する際には、通常のプログラム実行と同じように CPU が動作をします。
このため、ファイル・ロードやファイル・ライトを行う前に、ユーザ・プログラムを動作させるなどして割り込みが有効な状態でブレークさせている場合、モニタ・プログラムが動作すると割込み処理が入ってしまい、正しくユーザ・カスタマイズ・モニタ機能が動作しない可能性があります。
この場合、ハードウェアの初期化を行い、ターゲット・システムを初期化してから、再度、操作を行ってください。

2. モニタ・プログラム

2.1 モニタ・プログラムについて

モニタ・プログラムは、CSIDE からターゲット・システム上の RAM 空間に配置され、CSIDE 上からのファイル・ロード/ライト操作に対して、実際の書き込み/読み出しを行うプログラムです。
ただ、使用される環境によって CPU やフラッシュメモリが異なるため、CPU のインターフェース設定やフラッシュメモリの仕様も様々であり、共通のモニタ・プログラムでは読み書きを行う事はできません。
このため、ユーザ・カスタマイズ・モニタ機能を使用いただくには、使用されるターゲット・システム環境に合わせてカスタマイズしたモニタ・プログラムをお客様にて作成して頂く必要があります。

モニタ・プログラムのファイル一式は、CSIDE のインストールフォルダ以下の下記パスに収録されています。

CSIDE インストールフォルダ¥Etc¥[機種名]¥UCM
(例:PALMiCE3 ARM の標準インストールフォルダは「C:¥Program Files (x86)¥CSIDE¥Etc¥PALMiCE3 ARM¥UCM」です)

このモニタ・プログラムを使用する環境に合わせてカスタマイズを行います。

なお、この UCM フォルダ以下には、**[CPU 名]¥[CPU ボード名]** で、当社で動作確認を行ったターゲットボードのモニタ・プログラム一式を収録しています。同じターゲットボードをご利用の場合、このモニタ・プログラムをそのまま使用して書き込みを行う事ができます。このフォルダ内に収録されていないターゲットボードにてご利用になる場合には、使用される環境に近いモニタ・プログラム一式を元にしてカスタマイズを行ってください。

モニタ・プログラムのカスタマイズについては、本書の以降の内容で解説を行います。

2.2 モニタ・プログラムのカスタマイズ

2.2.1 モニタ・プログラム作成時の注意点

モニタ・プログラムをカスタマイズしてビルドを行う際には、以下の点をご注意ください。

・モニタ・プログラムは、ターゲットボード上の RAM 空間上で動作させる必要があります。
コンパイルの際には、モニタ・プログラムがご使用のターゲットボードの RAM 空間に配置するよう設定を行ってください。

・CSIDE は、モニタ・プログラムのデバッグ情報を元に、モニタ・プログラム上の変数情報を読み取り、モニタ・プログラムの制御を行います。このため、モニタ・プログラムは、デバッグ情報を出力した ELF/DWARF 形式のファイルで出力するようコンパイラの設定を行ってください。

2.2.2 ファイル構成

モニタ・プログラムは、C 言語、または、アセンブリ言語で記述されおり、以下のソース・ファイルで構成されます。

下の表の”CPU 依存”,”フラッシュメモリ依存”の項に、”有”と記されているソース・ファイルには、それぞれ使用する CPU/フラッシュメモリに関する記述があります。カスタマイズを行う際には、これらのソース・ファイルの編集を行います。

[フォルダ] / ファイル	ファイル内容	CPU 依存	フラッシュメモリ依存
UcmCommon.h	共通ソース・ファイルです(変更の必要はありません)。		
UcmMonitor.h、UcmMonitor.c	CSIDE の呼び出しに対応した関数や初期値付変数等が定義されているソース・ファイルです。このファイルに定義されている関数を操作して CSIDE はファイル・ロード/ファイル・ライト機能を提供します。		有
UcmCpuControl.h、UcmCpuControl.c	CPU のペリフェラルに対応したソース・ファイルです。ペリフェラルの設定や CtexFlash.c へのインターフェースを提供しています。	有	
UcmFlash.h、UcmFlash.c	フラッシュメモリに対応したソース・ファイルです。		有
[project]	ビルド環境関連一式です。	有	

当社で使用したコンパイラのプロジェクトを、**[project]**フォルダに収録しています。

ビルド環境の設定方法については、各開発環境のマニュアルを参照してください。

2.2.3 CPU に合わせる

CPU に関する記述がソース・ファイルの UcmCpuControl.h/ UcmCpuControl.c にあり、また、ビルド環境のプロジェクト内にも CPU に依存する設定があります。これを、ご使用になる CPU に合わせて編集してください。

UcmCpuControl.h/c は、主に以下の関数が定義されています。

関数名	(説明)
CntError Init(void);	CPU の割り込みの停止やフラッシュメモリに接続されているポートの設定コードを記述します。
CntError Exit(void);	CS を無効にしています。
CntError SetCS(int en);	フラッシュメモリに接続されている CS 信号についての制御コードを記述します。
CntError sendCommand(DBit8 cmd);	コマンドを送信するための制御コードを記述します。
CntError sendAddress(DBit32 address,int size);	アドレスを送信するための制御コードを記述します。
CntError sendData(DBit8 *d8, int size);	データを送信するための制御コードを記述します。
CntError recvData(DBit8 *d8, int size);	データを受信するための制御コードを記述します。

2.2.4 フラッシュメモリに合わせる

フラッシュメモリに関する記述が、ソース・ファイルの UcmFlash.h/ UcmFlash.c、UcmMonitor.h / UcmMonitor.c にあります。
ご使用になるフラッシュメモリに合わせて編集してください。
UcmFlash.h/ UcmFlash.c は、UcmCpuControl.h で定義されている関数を使用して記述します。

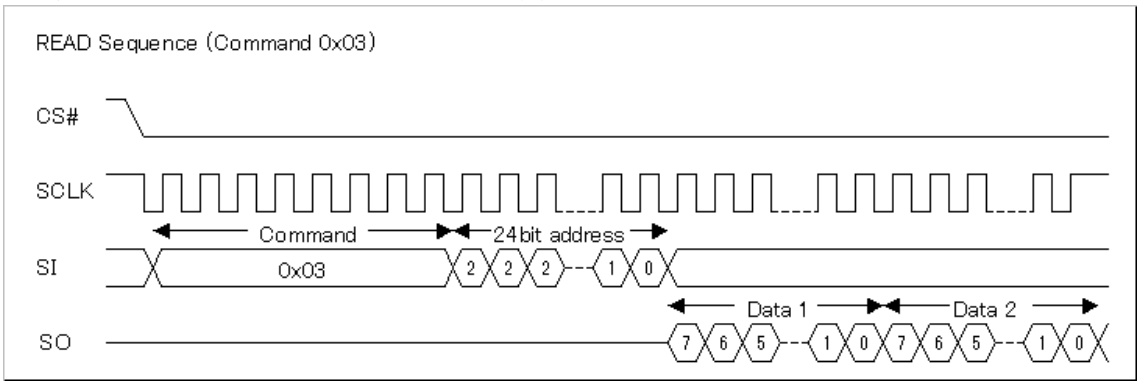
関数名	(説明)
CntError execRDID(DBit8 *buf, int size);	RDID コマンドを発行します。
CntError execWREN(void);	WREN コマンドを発行します。
CntError execWRDI(void);	WRDI コマンドを発行します。
CntError execREAD(DBit32 addr, DBit8 *buf, int size);	READ コマンドを発行し、size バイト分のデータを buf に格納します。
CntError execWRITE(DBit32 addr, DBit8 *buf, int size);	WRITE コマンドを発行し、size バイト分の buf データを送信します。
CntError execRDSR(DBit8 *buf, int size);	RDSR コマンドを発行し、size バイト分のデータを buf に格納します。
CntError execWRSR(DBit8 *buf, int size);	WRSR コマンドを発行し、size バイト分の buf データを送信します。
CntError execRES(DBit8 *buf, int size);	RES コマンドを発行し、size バイト分のデータを buf に格納します。

Info.

コマンド名やその動作内容については、使用するフラッシュメモリによって違いがあります。
カスタマイズの元にしたモニタ・プログラムで使用するフラッシュメモリや、実際に使用されるフラッシュメモリのマニュアルを参照して作成を行ってください。

例

以下は、フラッシュメモリの READ コマンドの対応例です。




```
CntError execREAD(DBit32 addr, DBit8 *buf, int size)
{
        SetCS(1);
        if(sendCommand(0x03) != C_OK) return retExecErr();
        if(sendAddress(addr,3) != C_OK) return retExecErr();
        if(recvData(buf, size) != C_OK) return retExecErr();
        SetCS(0);
        return C_OK;
}
```

2.3 モニタ・プログラムの変数

ユーザ・カスタマイズ・モニタ機能では、モニタ・プログラム内の変数を、CSIDE から直接参照したり変更したりすることで、パラメータの受け渡しを行って制御を行います。

変数には、フラッシュメモリのセクタ・ブロック・サイズや CSIDE とのデータ送受信テーブルのバッファサイズなど使用環境の定義を行うものが含まれているので、ご使用の環境に合わせて値の設定を行ってください。

なお、以下の変数は、CSIDE がモニタ・プログラムの制御のために参照しているので、変数名を変更するとユーザ・カスタマイズ・モニタ機能が正常に動作しなくなります。変数名の変更は、絶対に行わないでください。

■ グローバル変数（CSIDE やモニタ内でのデータのやり取りを行うための変数です）

変数名	説明
volatile unsigned long _ucm_para[PARAMMAX]	任意のパラメータが格納されます。 モニタ・プログラム側でこれらの値を使用することができます。
volatile unsigned long _ucm_stack[STACKMAX+1]	関数が使用するスタック領域です。 CSIDE からの呼び出し時、スタックチェックを行うため+1 されています。
volatile unsigned long _ucm_size	CSIDE で読み出されたファイルのバイト数が格納されます。 この値が 0 の場合は EOF に到達したことを示します。
volatile unsigned long _ucm_ret[PARAMMAX]	CSIDE コマンドで指定されたファイルのサイズが格納されます。
volatile unsigned long _ucm_err	モニタ・プログラムの関数を実行した後のステータスを返します。0 の場合は, 正常終了したことを示し、それ以外の場合は何らかのエラーが発生したことを示します。
unsigned char _ucm_buf[BUFMAX];	CSIDE で読み出されたファイルの内容が格納されます。 BUFMAX は任意のサイズを指定します。

■ 初期値付変数（フラッシュメモリやモニタの固有情報を定義する変数です）

以下の変数は、変更の必要はありません。

変数名	説明
const unsigned long _UCM_VER	モニタ・バージョン反映されます。 #define VER で設定される値でユーザが変更不可の内容です。
const unsigned long _UCM_PARAMMAX	CSIDE とモニタがやり取りする際の変数テーブル(_ucm_para/_ucm_ret)数が反映されます。#define PARAMMAX で設定される値でユーザが変更不可の内容です。
const unsigned long _UCM_BLOKTBL_MAX	セクタ・ブロック・テーブル数が反映されます。 #define BLOKTBL_MAX で設定される値でユーザが変更不可の内容です。

以下の変数は、通常は変更の必要はありません。

変数名	説明
const unsigned long _UCM_MAP_ADDR_START	セクタ・ブロック領域の有効開始アドレスが反映されます。 部分的に書き込みを行いたい場合に設定されるものです。 #define MAP_ADDR_START で設定する値で通常は 0 から変更する必要はありません。
const unsigned long _UCM_MAP_ADDR_END	セクタ・ブロック領域の有効終了アドレスが反映されます。 部分的に書き込みを行いたい場合に設定されるものです。 #define MAP_ADDR_END で設定する値で通常はセクタ総サイズ-1 から変更する必要はありません。
const unsigned long _UCM_ADDR_MASK	CSIDE アドレス変換マスクが反映されます。CPU アドレス情報をデバイス(ROM)のアドレスに変換する際に設定されるものです。#define ADDR_MASK で設定する値で使用しない場合、0xFFFFFFFF から変更する必要はありません。

以下の変数は、使用する環境に合わせて値を変更してください。

変数名	説明
const unsigned long _UCM_STACKMAX	各関数で使用するスタック領域(配列)カウントが反映されます。変数テーブル(_ucm_stack)で long 分のカウントです。#define STACKMAX で設定する値でソース追加などで必要に応じて変更してください。
const unsigned long _UCM_PAGE_SIZE	デバイス書き込み時の 1 ページの書き込みサイズが反映されます。 #define PAGE_SIZE で設定する値で使用するデバイスにあわせて変更してください。
const unsigned long _UCM_BUFMAX	CSIDE とモニタ・プログラムが送受信バッファのサイズが反映されます。 #define BUFMAX で設定する値でページサイズやセクタ・ブロック・サイズとの関係で変更してください。
const unsigned long _UCM_RDID_DATA1～3	ID チェックで使用される比較用 ID データ値が反映されます。 #define RDID_DATA1～3 で設定する値で使用するデバイスにあわせて変更してください。
const unsigned long _UCM_BLOCK_SIZE0～7	セクタ・ブロック・サイズが反映されます。#define BLOCK_SIZE0～7 で設定する値で使用するデバイスにあわせて変更してください。
const unsigned long _UCM_BLOCK_COUNT0～7	同じセクタ・ブロック・サイズが連続する数が反映されます。 #define BLOCK_COUNT0～7 で設定する値で使用するデバイスにあわせて変更してください。

2.4 モニタ・プログラム関数

2.4.1 関数の仕様

CSIDE は、モニタ･プログラムのデバッグ情報からシンボル参照を行い、モニタ･プログラム関数の実行を行います。

以下に基本仕様を満たした関数例を示します。

```
void UF_UcmFuncBreak(void)
{
        return;  /* ブレーク関数(この関数の先頭にブレークポイントが設定される) */
}

void UF_UcmFuncSample(void)
{
        /*
ユーザ処理
*/
        ._ucm_err = 0;     /* エラー・ステータス(0=エラーなし) */
        UF_UcmFuncBreak ();          /* ブレーク関数を呼び出す */
}
```

使用例	UF_UcmCallFunction(UF_UcmFuncSample, UF_UcmFuncBreak, 0, 0, 0)

2.4.2 基本関数一覧

UF_UcmFuncInit	初期化用関数
UF_UcmFuncExit	終了処理用関数
UF_UcmFuncRead	読み出し用関数
UF_UcmFuncWrite	書き込み用関数
UF_UcmFuncBlockErase	(ブロック)消去用関数
UF_UcmFuncCheckSum	チェック･サム用関数
UF_UcmFuncCheckID	ID チェック用関数
UF_UcmFuncBreak	ブレーク用関数
UF_UcmFuncChipErase	全消去関数

※基本関数は、CSIDE から制御を行う際に使用しますので、関数名の変更はしないでください

2.4.3 基本関数の詳細

UF_UcmFuncInit

説明	初期化用関数。変数の初期化や I/F,ピンファンクションの設定を行います。CSIDE からファイル・ロード/ファイル・ライトで初めに呼び出される関数です。		
パラメータ	para1	0	I/F,ピンファンクションの設定を行いません。
		1	I/F,ピンファンクションの設定を行う。
使用例	UF_UcmCallFunction(UF_UcmFuncInit, UF_UcmFuncBreak, 0x1) UF_UcmFuncInit 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータには、para1=0x1 が指定されており、 I/F,ピンファンクションの設定を含みます。 なお、CSIDE のファイル・ロードやファイル・ライトからは para1=0x1 で呼び出されます。		

Info.

UF_UcmFuncInit は変数の初期化や I/F、ピンファンクションの設定を行う関数のため、他の UF_UcmFuncRead や UF_UcmFuncWrite などの関数を使用する前には、一度実行しておいてください。

UF_UcmFuncExit

説明	終了処理用関数。モニタ・プログラム実行の終了処理を行います。CSIDE からファイル・ロード/ファイル・ライトの最後に呼び出される関数です。		
パラメータ	para1	0	CS 信号を操作しません。
		1	CS 信号を無効にします。
使用例	UF_CallFunction(UF_UcmFuncExit, UF_UcmFuncBreak, 0x1) UF_UcmFuncExit 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータには、para1=0x1 が指定されており、CS 信号を無効にする操作を含みます。 なお、CSIDE のファイル・ロードやファイル・ライトからは para1=0x1 で呼び出されます。		

UF_UcmFuncRead

説明	リードコマンドを送信します。受信結果はバッファ(_uf_buf)に保存されます。受信サイズはバッファサイズ分です。		
パラメータ	para1	任意	リード・アドレスのオフセットを指定します。
使用例	UF_UcmCallFunction(UF_UcmFuncRead, UF_UcmFuncBreak, 0x1000) UF_UcmFuncRead 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータには、para1=0x1000 が指定されており、0x1000 番地からバッファサイズ分のメモリを読み出します。		

UF_UcmFuncWrite

説明	ライトコマンドを送信します。送信結果はバッファ(_uf_buf)に保存されます。送信サイズは_size にセットされます。		
パラメータ	para1	任意	ライト・アドレスのオフセットを指定します。
使用例	UF_UcmCallFunction(UF_UcmFuncWrite, UF_UcmFuncBreak, 0x1000) UF_UcmFuncWrite 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータには、para1=0x1000 が指定されており、0x1000 番地からバッファサイズ分のメモリを書き込みます。		

UF_UcmFuncBlockErase

説明	ブロック消去コマンドを送信します。【注意】ブロック消去が完了するまで関数は返ってきません。		
パラメータ	para1	任意	ブロック消去アドレスのオフセットを指定します。必ずブロック境界のアドレスを指定してください。
使用例	UF_UcmCallFunction(UF_UcmFuncBlockErase, UF_UcmFuncBreak, 0x10000) UF_UcmFuncBlockErase 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータには、para1=0x10000 が指定されており、0x10000 番地からブロックサイズ分が消去されます。 ブロックサイズは、UcmFlash.h で定義されるコマンドにより変化します		

UF_UcmFuncCheckSum

説明	32 ビットチェック・サムを算出します。算出した値はビッグ・エンディアンでバッファ(_uf_buf)に保存されます。	
パラメータ	para1	指定された範囲のデータのサム値を設定します。
	para2	チェックを開始するアドレスを指定します。
	para3	チェックを終了するアドレスを指定します。
使用例	UF_CallFunction(UF_UcmFuncCheckSum, UF_UcmFuncBreak, 0x1234, 0x0, 0xffff) UF_UcmFuncCheckSum 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータには、para1=0x0x1234, para2=0x0, para2=0xffff が指定されており、0x0 番地から 0xFFFF 番地までのサム値を 0x1234 と比較します。	

UF_UcmFuncCheckID

説明	ID チェック・コマンドを送信します。受信結果はバッファ(_uf_buf)に保存されます。		
パラメータ	para1	0	ID チェックを行いません。(受信結果を得るのみ)
		1	ID チェックを行う。
	para2	任意	Manuf. ID の値を指定します。(para1=1 のとき有効)(para2～para4 は一括して省略可能)
	para3	任意	Device ID byte 1 の値を指定します。(para1=1 のとき有効)(para2～para4 は一括して省略可能)
	para4	任意	Device ID byte 2 の値を指定します。(para1=1 のとき有効)(para2～para4 は一括して省略可能)
使用例	UF_UcmCallFunction(UF_UcmFuncCheckID, UF_UcmFuncBreak, 0x0, 0, 0, 0) UF_UcmCallFunction(UF_UcmFuncCheckID, UF_UcmFuncBreak, 0x0,) UF_UcmFuncCheckID 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータには、para1=0x0 が指定されており、ID チェックは行いません。後者の指定方法では para2,para3,para4 は省略されています。 なお、CSIDE のファイル・ロードやファイル・ライトからは para1=0x1,para2～4 はモニタの設定値で呼び出されます。		

UF_UcmFuncBreak

説明	ブレーク用関数。常に、[モニタ・プログラムの停止アドレス]に指定します。
パラメータ	ありません。(ブレーク用のため指定できません。)
使用例	UF_UcmCallFunction(UF_UcmFuncXYZ, UF_UcmFuncBreak, 0x1) UF_UcmFuncXYZ 関数から実行し、UF_UcmFuncBreak 関数でブレークします。なお、パラメータは UF_UcmFuncXYZ 関数に対する指定になります。

UF_UcmFuncChipErase

説明	全消去コマンドを送信します。【注意】全消去が完了するまで関数は返ってきません。
パラメータ	ありません。【注意】最低でもひとつのカンマは必要です。
使用例	UF_UcmCallFunction(UF_UcmFuncChipErase, UF_UcmFuncBreak,) UF_UcmFuncChipErase 関数から実行し、UF_UcmFuncBreak 関数でブレークします。 パラメータは省略されています。

2.5 モニタ・プログラムのデバッグ

2.5.1 モニタプログラムデバッグ用コマンド

カスタマイズしたモニタ・プログラムのデバッグは、CSIDE のコマンドを使用して行います。

デバッグ用コマンドとして、ユーザ関数実行コマンド(UF_UcmCallFunction)と、ユーザ関数 PC/SP 設定コマンド(UF_UcmStart)を用意しています。このデバッグ・コマンドは、デバッグが完了しモニタ・プログラムが正常に動作すれば、使用する必要はありません。

■ ユーザ関数実行コマンド

書式	UF_UcmCallFunction(start, end, [para1], [para2], [para3], [para4])	
機能	ターゲット上の関数を実行します。	
引数	start	モニタ・プログラム関数のスタートアドレスを指定します。
	end	モニタ・プログラム関数の停止アドレスを指定します。停止はブレークポイントで行います。通常は、UF_UcmFuncBreak を設定します。
	[para1], [para2], [para3], [para4]	32 ビットの任意の数値をモニタ・プログラム関数に渡すときに使用します。(実行するモニタ・プログラム関数によりパラメータの意味は異なります。)
詳細	1. start で示されるアドレスに PC をセットします。 2. end で示されるアドレスにブレークポイントをセットします。 3. パラメータ([para1]～[para4])をターゲット配列._ucm_para[]に順番にセットします。 4. ターゲットをプログラム実行し、ブレークするまで待ちます。(ブレークしない場合には強制ブレークが可能) 5. ターゲット・エラー変数._ucm_err にセットされている場合、メッセージが出力されます。(処理終了。メッセージ表示モードの場合)。変数内容に応じて、評価してください。	
使用例	UF_UcmCallFunction(UF_UcmFuncInit, UF_UcmFuncBreak, 1) UF_UcmFuncInit という関数の先頭アドレスから、UF_UcmFuncBreak で示されるアドレスまで関数を実行する	

※ モニタ・プログラム関数に対するパラメータ([para1]～[para4])の個数は最大 4 個までとなっています。後ろに続くパラメータがない場合は省略可能ですが、すべて省略する場合はカンマが一つ必要です。

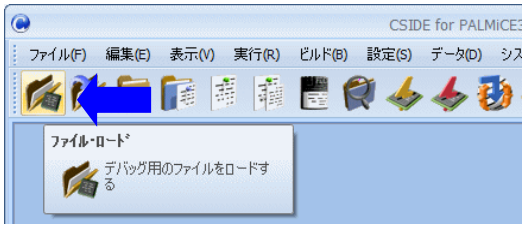
■ ユーザ関数 PC/SP 設定コマンド

書式	UF_UcmStart(start)	
機能	ユーザ・カスタマイズ・モニタ関数用の PC/SP を設定します。	
引数	start	モニタ・プログラム関数のスタートアドレスを指定します。
詳細	1. start で示されるアドレスに PC をセットします。 2. _ucm_stack[]の配列の最後のアドレスを SP に設定します。	
使用例	UF_UcmStart(UF_UcmFuncInit) UF_UcmFuncInit という関数の先頭アドレスを設定します。SP は_ucm_stack[]の配列の最後のアドレスを設定します。	

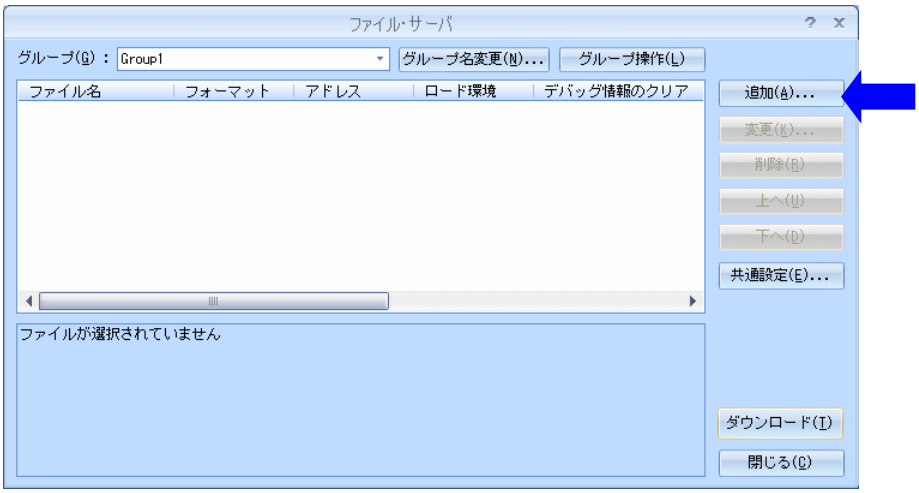
2.5.2 デバッグ方法

カスタマイズしたモニタ・プログラムのデバッグやコマンドを使用してフラッシュメモリのデータの読み書きは、CSIDE 上で以下の手順で行います。

ツールバーから **[ファイル・ロード]**ボタンを押します。



表示されるファイル・サーバに、デバッグを行うモニタ・プログラムを登録するために**[追加]**ボタンを押します。



表示されるダイアログから、デバッグするモニタ・プログラム・ファイルを選択し、ダウンロードの設定は、

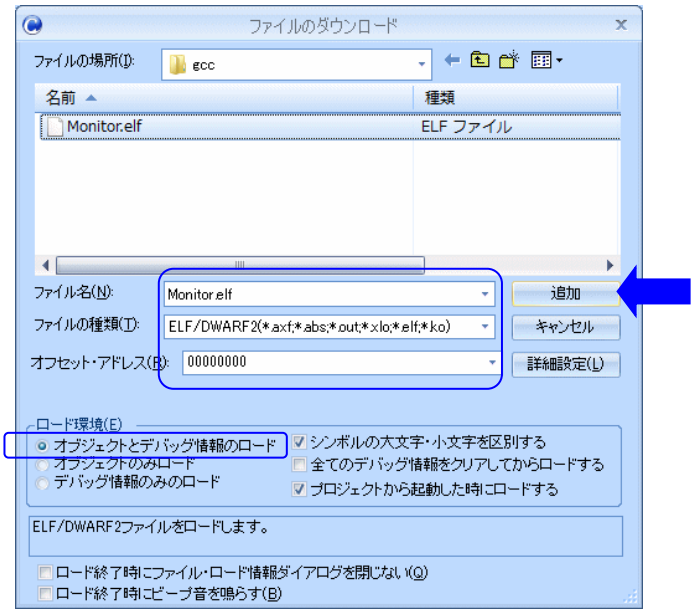
ファイルの種類　：　**[ELF/DWARF2(*.axf,*.abs,*.out,*.xlo,*.elf,*.ko)]**

オフセットアドレス　：　**0**

ロード環境　：　**[オブジェクトとデバッグ情報のロード]**

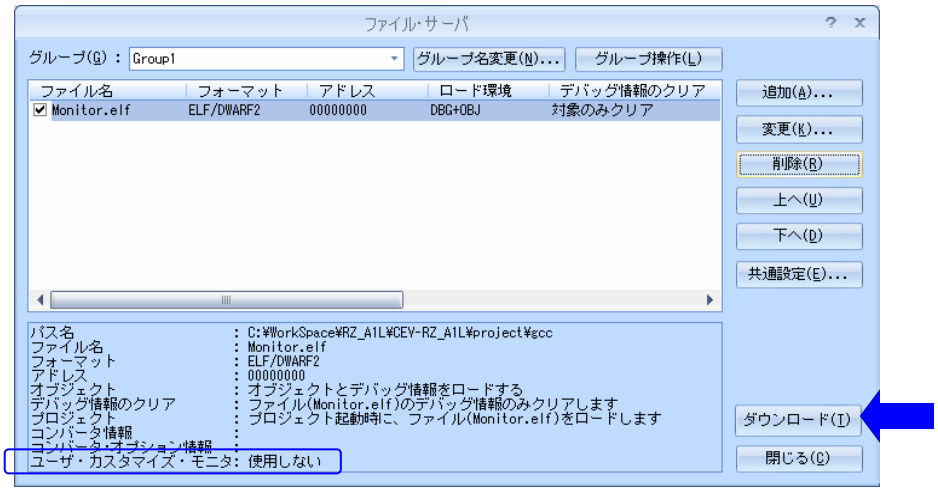
を選択します。

設定出来ましたら、**[追加]**ボタンを押します。



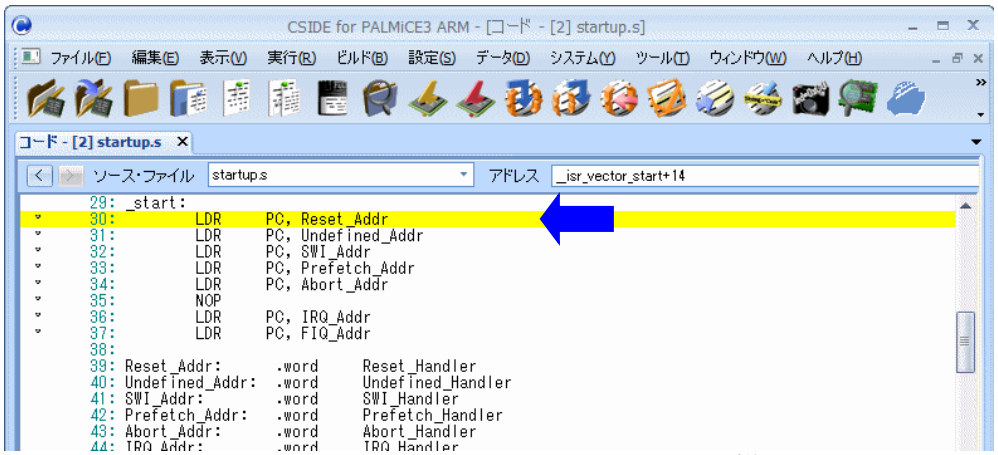
ファイル・サーバに、選択したモニタ・プログラム・ファイルが登録されています。

ダイアログ下部に表示されているステータスに、「**ユーザ・カスタマイズ・モニタ:使用しない**」となっていることを確認し、**[ダウンロード]**ボタンを押してダウンロードを行います。



ダウンロードが完了したら、コード・ウィンドウを開きます。(メニューバー　**[表示][コード]**)

ファイル・ロードを行った事により、PC はエントリーポイント又は PC の初期値に設定されています。



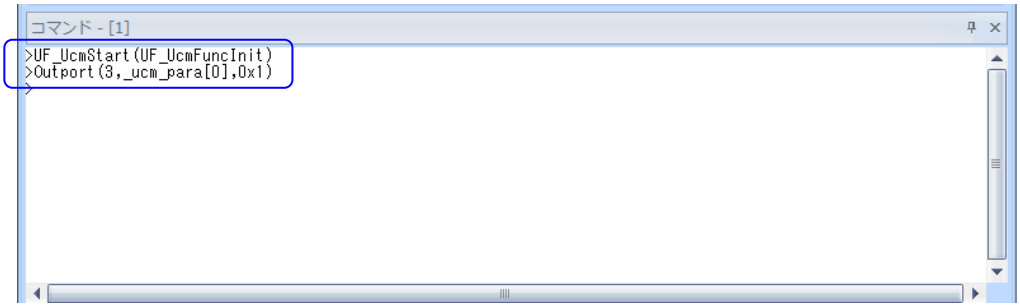
※ご使用のコンパイラによりコードは異なります。

さらに、コマンド・ウィンドウを開きます。(メニューバー**[表示][コマンド]**)

表示されたコマンド・ウィンドウに、**UF_UcmStart(デバッグする関数)**を入力後、Enter キーを押して実行します。

例) UF_UcmFuncInit 関数をデバッグする場合 ：**UF_UcmStart(UF_UcmFuncInit)**

UF_UcmStart コマンドを実行すると、対象の関数がデバック出来るよう PC、SP が設定されます。

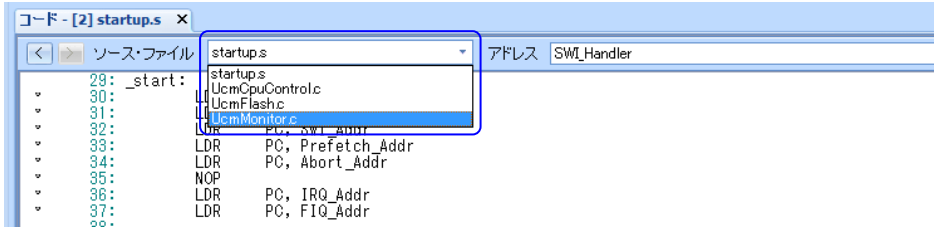


なお、デバッグ対象の関数にパラメータがある場合、_ucm_para[]配列がパラメータとして使用されます。デバッグを行う前に、以下のどちらかの方法でパラメータを設定してください。

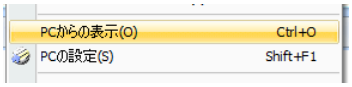
・変数ウィンドウのグローバルタブにて、_ucm_para[]の値を入力する

・Outport コマンドを使用してパラメータを設定する。 例) **Outport(3,_ucm_para[0],0x1)**

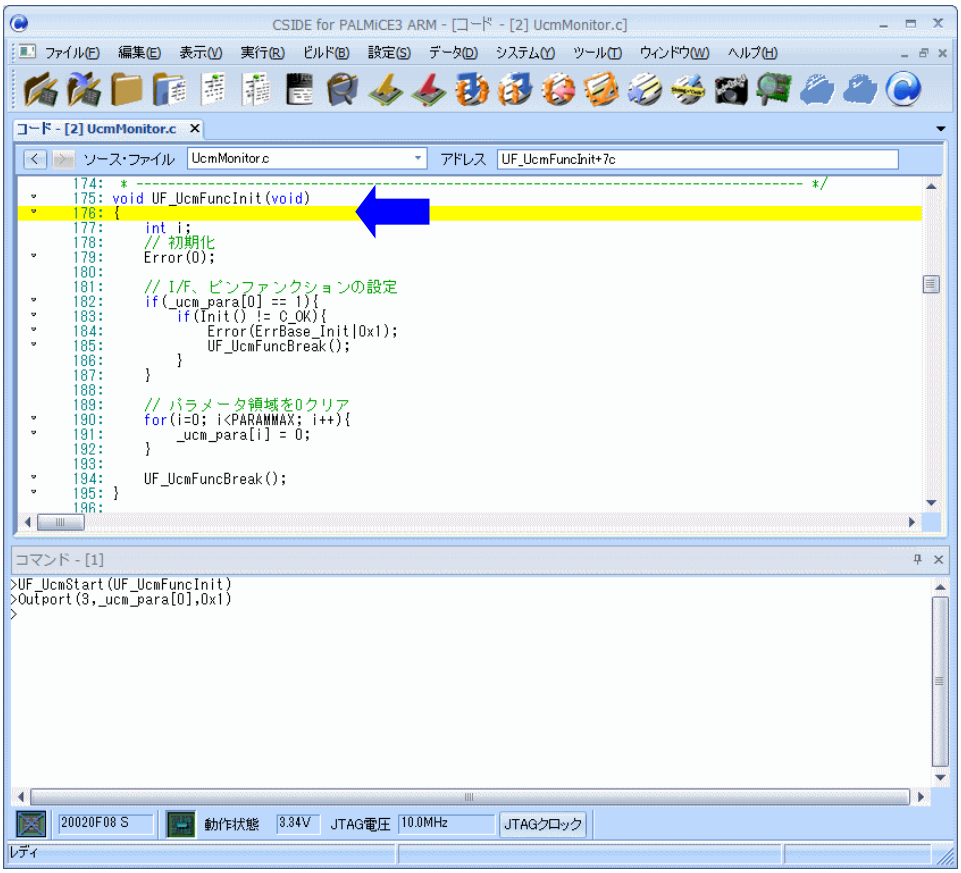
ソース・ファイルから"**UcmMonitor.c**"を選択します。



ソース・ファイルが表示されたら、コード・ウィンドウのポップアップメニューから”**PC からの表示**”を選択します。



表示が切り替わり、"**UF_UcmStart**"コマンドで指定した関数の先頭アドレスが表示されます。



以上の手順で、関数デバッグの開始準備が整いました。

その後、トレース実行やステップ実行などを用いて、デバッグ対象の関数が最後まで正常な動作を行えるか、デバッグを行ってください。

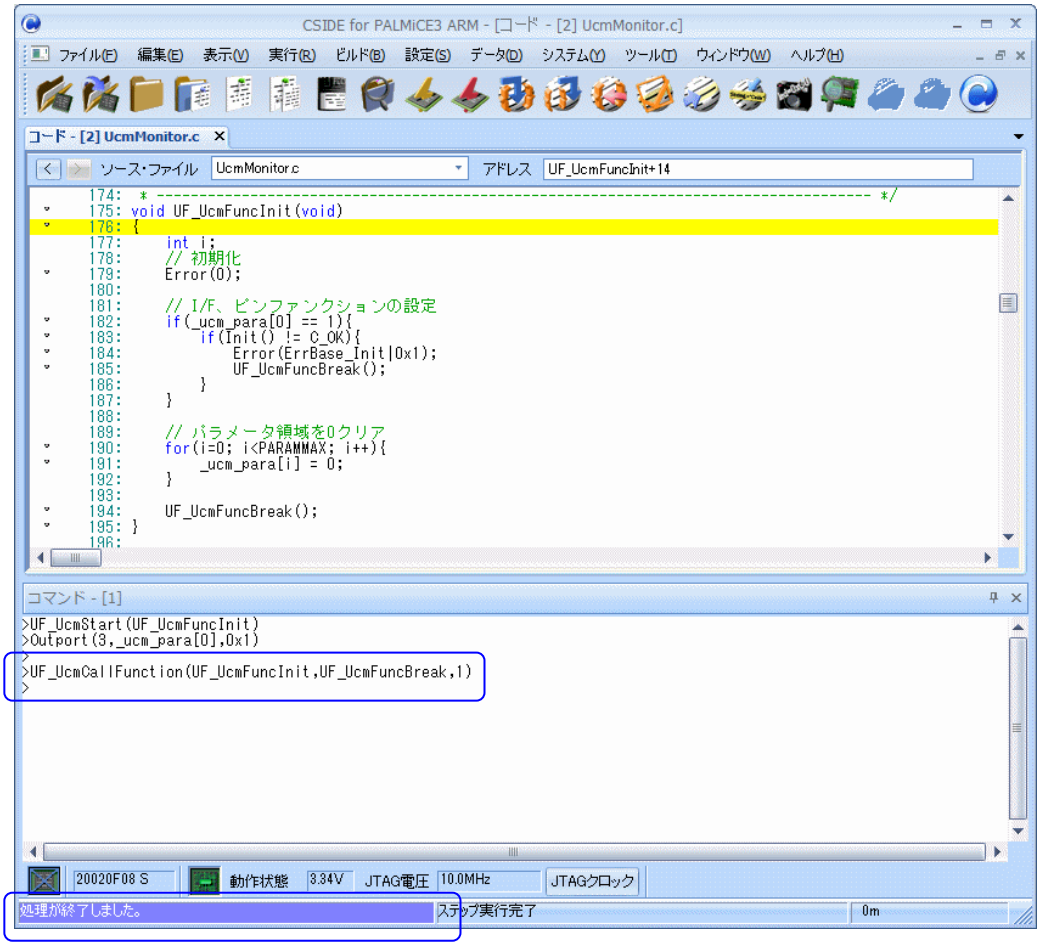
なお、関数の先頭からトレース実行などが動作しない場合には、CSIDE の設定、ハードウェアの動作、モニタ・プログラムの配置アドレスの誤り、モニタ・プログラムのワーク領域の設定、デバッグ実行時点でのレジスタの値などフラッシュメモリへのアクセスとは関係ない事で問題が発生している可能性があります。

また、main 関数が用意されているモニタ・プログラムでは、モニタをダウンロード後、トレース実行を行い main 関数まで動作させると初期化やチェック ID の処理を確認することも可能です。

◎関数単位の実行確認

先の手順では、デバッグ対象の関数の先頭でブレークさせていますが、"**UF_UcmCallFunction**"コマンドを使用することで、関数単位で実行させて動作の確認を行うことも可能です。モニタ・プログラムのダウンロード後に、コマンド・ウィンドウから **UF_UcmCallFunction(デバッグする関数, UF_UcmFuncBreak,引数)**を入力し実行します。

実行後、指定した関数が実行され、正常に実行が行えた場合、画面左下に"**処理が終了しました**"と表示されます。



Info.

ファイル・ロード/ファイル・ライトで使用しているモニタ関数と呼び出し順は下記となります。
CSIDE のメニュー**[ファイル][ユーザ・カスタマイズ・モニタ・マップ]**の設定から各関数の呼び出しを行わないようにすることが可能です。

●ファイル・ロード(呼び出し関数順)	●ファイル・ライト(呼び出し関数順)
① UF_UcmFuncInit	① UF_UcmFuncInit
② UF_UcmFuncCheckID	② UF_UcmFuncCheckID
③ UF_UcmFuncRead ※	③ UF_UcmFuncRead ※
④ UF_UcmFuncBlockErase ※	④ UF_UcmFuncExit
⑤ UF_UcmFuncWrite ※	※③は、読み込みサイズにより繰り返します
⑥ UF_UcmFuncCheckSum ※	
⑦ UF_UcmFuncExit	
※ ③～⑥は、書き込むセクタ・ブロックにより繰り返します。③はセクタの途中からの書き込み時のみ呼び出されます。	

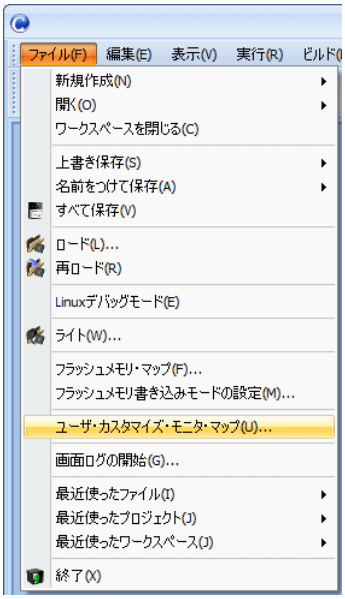
3. CSIDE の設定

モニタ・プログラムのデバッグを行って正常な動作が確認できましたら、続いて CSIDE の設定を行います。

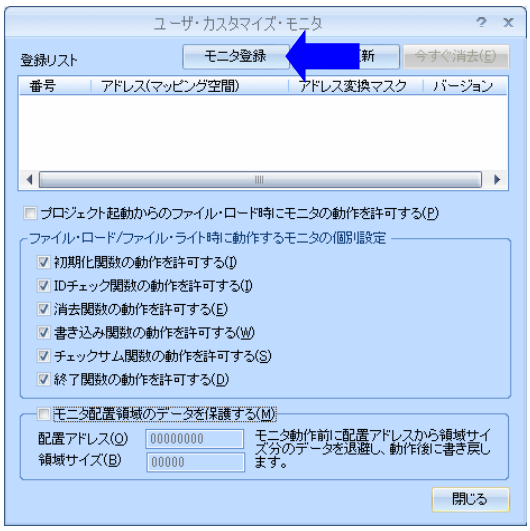
3.1 モニタ・プログラムの設定

ユーザ・カスタマイズ・モニタを使用してファイル・ロード/ファイル・ライトを行うに、作成したモニタ・プログラムを CSDIE に登録が必要です。以下の手順で、登録を行います。

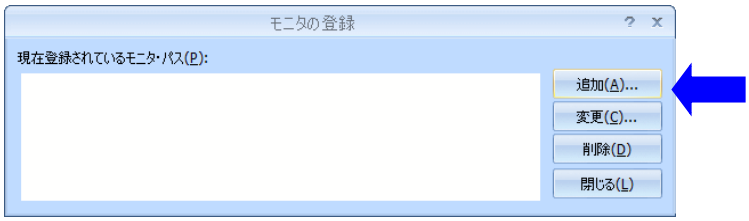
メイン・メニュー**[ファイル] [ユーザ・カスタマイズ・モニタ・マップ]**を選択します。



選択するとユーザ・カスタマイズ・モニタ・ダイアログ・ボックスが表示されます。
モニタ・ファイルを登録するため**[モニタ登録]**ボタンを押します。

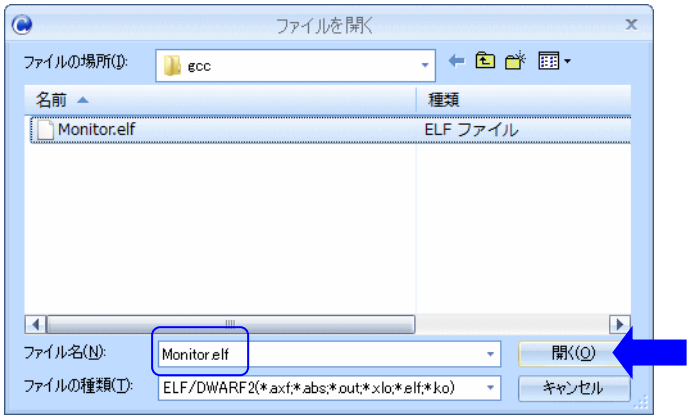


モニタの登録ダイアログ・ボックスが開きます。 **[追加]**ボタンを押します。



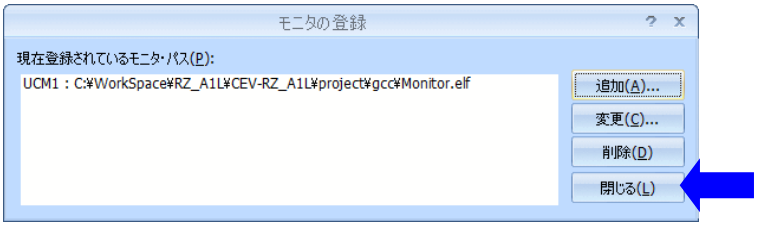
[参照]ボタンを押し、モニタ・ファイルを指定します。

モニタ・ファイルを指定して、**[開く]**ボタンを押します。



モニタ・ファイルが指定されたので、**[OK]**ボタンを押します。

ユーザ・カスタマイズ・モニタ 1(UCM1)にモニタ・ファイルが登録されます。**[閉じる]**ボタンを押します。



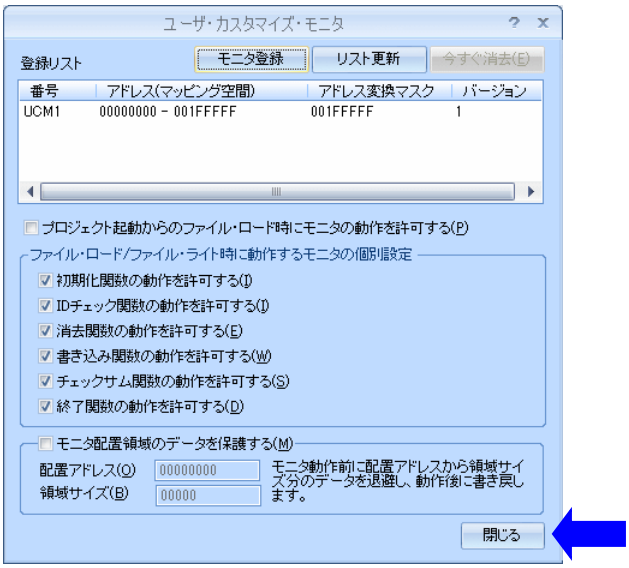
モニタ・ファイルをターゲット・システムへダウンロードする確認メッセージが表示されます。**[はい]**ボタンを押します。
なお、この時点でダウンロードするエリアの RAM は初期化されている(読み書き可能な状態である)必要があります。

Info.

- モニタ・ファイル登録時のロードは、必須ではありません。ロードを行わない場合、ユーザ・カスタマイズ・モニタ・ダイアログ・ボックスの**[モニタ動作前にモニタ・ファイルのロードを許可する]**のチェックを有効にしてください。
- モニタ・ファイルにシンボル情報(変数テーブルや初期値付変数)が正しく出力されていないと正常のロードすることはできません。

正常にダウンロードが行えたら、下記メッセージが表示されます。**[OK]**ボタンを押します。

モニタが正常にロードできると、モニタに登録された情報(一部)がダイアログ・ボックスに反映されます。



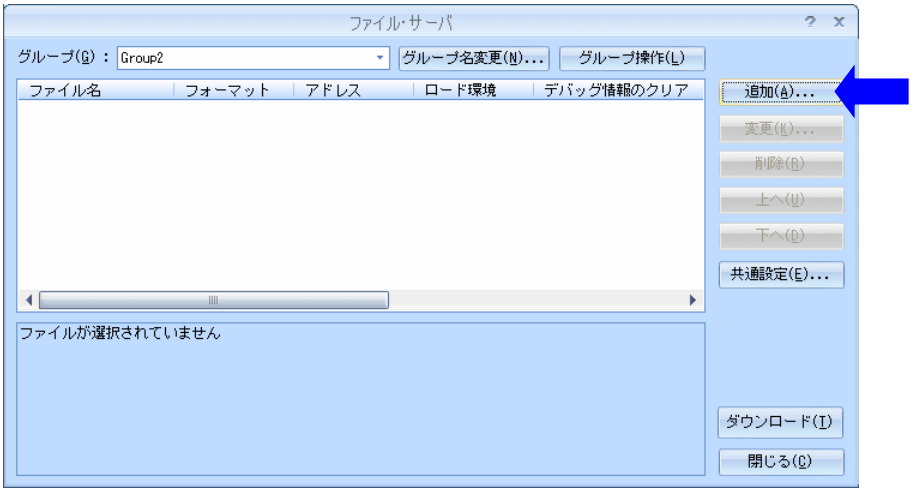
[閉じる]ボタンを押します。以上で準備は完了です。

3.2 フラッシュメモリへの書き込み

モニタ登録後、ファイル・ロード機能を使用してフラッシュメモリに書き込みを行います。

ツールバーから**[ファイル・ロード]**を選択します。

ファイル・サーバにロードするファイルを登録するために**[追加]**を選択します。



フラッシュメモリに書き込むファイルを選択します。ユーザ・カスタマイズ・モニタを使用したファイル・ロードでは、フラッシュメモリの先頭アドレスを 0 番地として書き込みを行います。

ここでは、バイナリ・ファイルをフラッシュメモリの先頭から書き込むため、

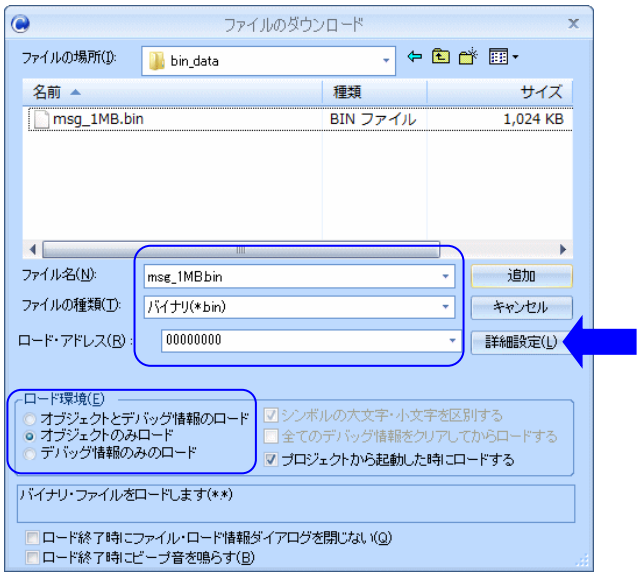
ファイルの種類　：**[バイナリ(*.bin)]**

ロードアドレス　：**0**

ロード環境　　：**[オブジェクトのみロード]**

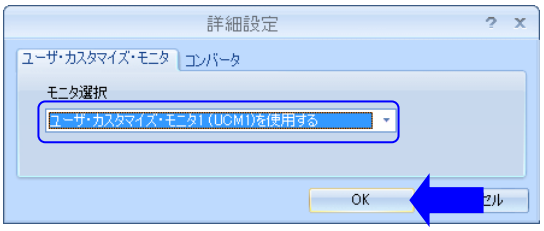
を選択します。

続いて、**[詳細設定]**ボタンを押します。

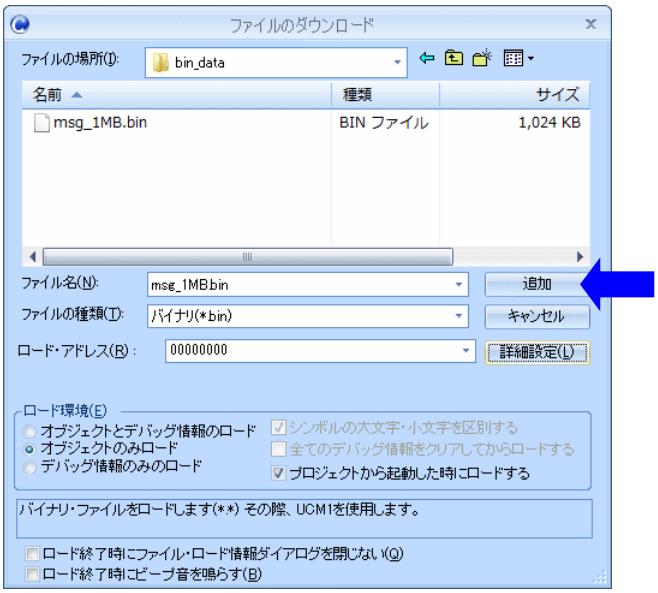


詳細設定ダイアログ・ボックスで、登録されている**"ユーザ・カスタマイズ・モニタ 1(UCM1)を使用する"**を選択します。

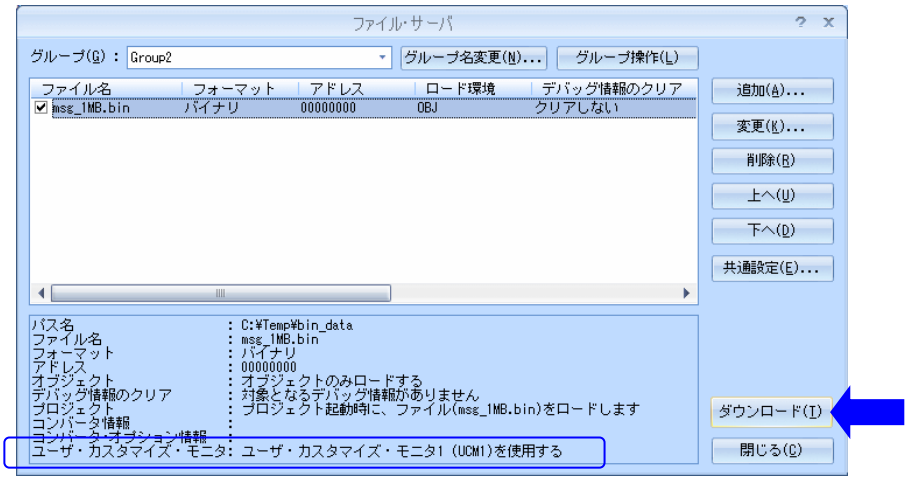
[OK]ボタンを押します。



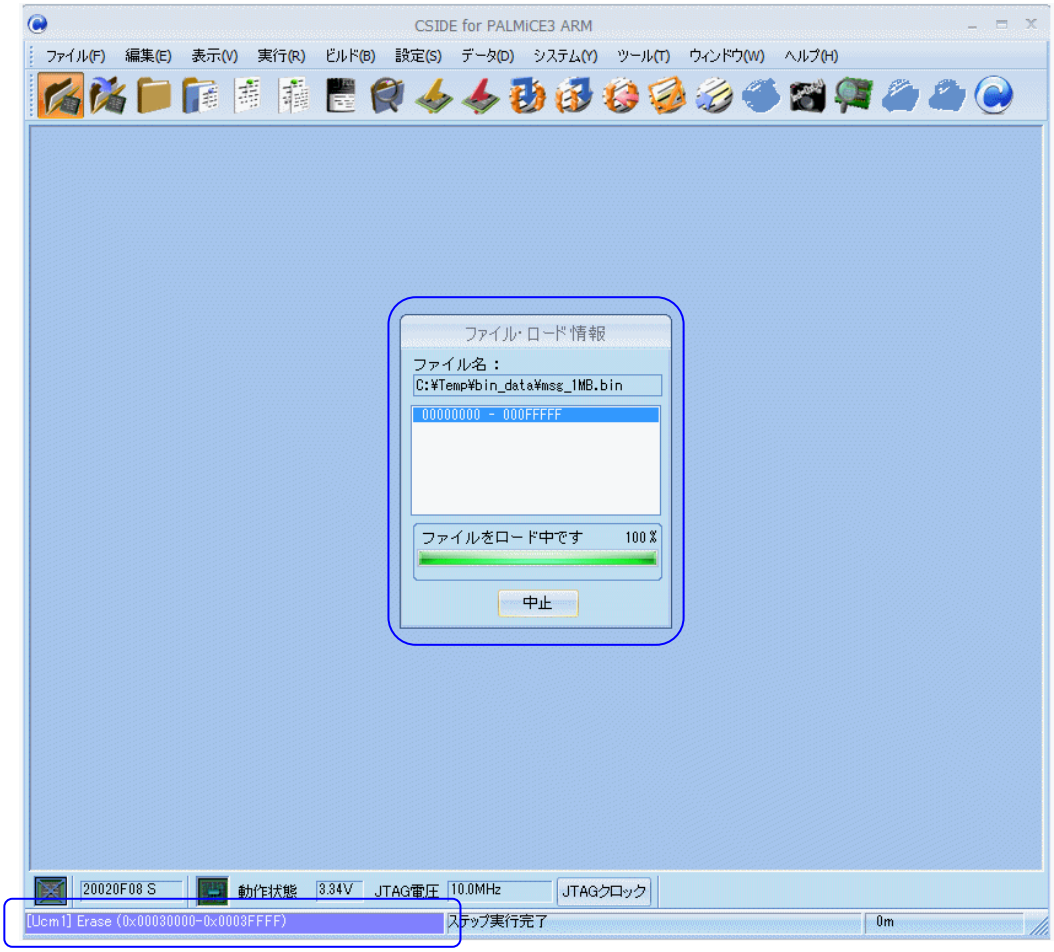
ユーザ・カスタマイズ・モニタ番号が選択されると、ダイアログ下部のメッセージ表示が変更されます。
表示内容を確認して、**[追加]**ボタンを押します。



設定の詳細表示にて、「**ユーザ・カスタマイズ・モニタ:ユーザ・カスタマイズ・モニタ 1(UCM1)を使用する**」と表示されている事を確認して、**[ダウンロード]**を行います。



ダウンロードが開始されると、ファイルロード・ダイアログ・ボックスとステータス・バーに書き込み状態が表示されます。

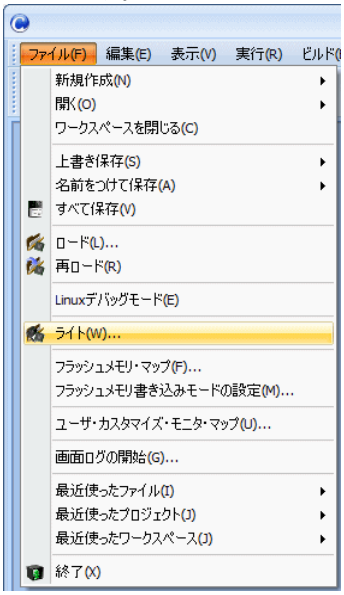


ファイル・ロードが完了(ファイル・ダイアログ・ボックスが自動で閉じられる)すると書き込み完了です。

3.3 フラッシュメモリ内容のファイルへの書き出し

フラッシュメモリの内容を読み出してファイルとして書き出します。

メイン・メニュー**[ファイル]** **[ファイル・ライト]**を選択します。



ファイル・ライト・ダイアログが開くと、ファイル・ライトする場所に移動し、出力ファイル名を入力します。

ユーザ・カスタマイズ・モニタを使用したファイル・ライトは、0 番地がフラッシュメモリの先頭アドレスとして処理されます。

ここでは、バイナリ・ファイルをフラッシュメモリの先頭から 1MB 分を書き出す設定として、

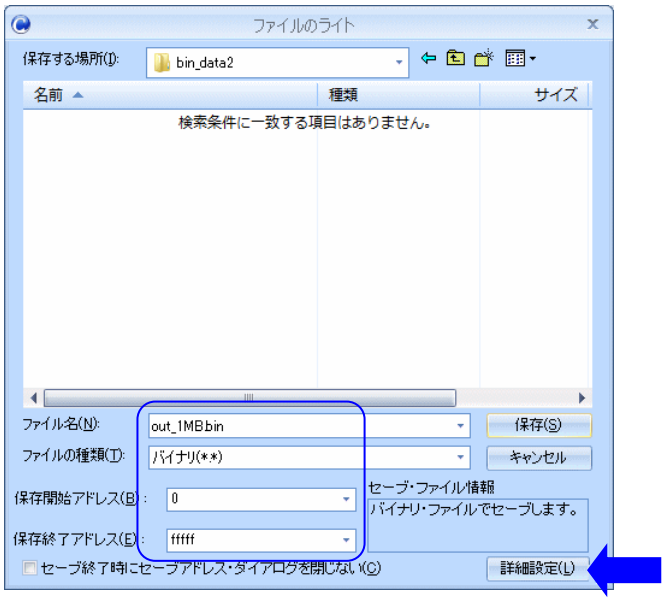
ファイルの種類　:**[バイナリ(*.bin)]**

保存開始アドレス　:**0**

保存終了アドレス　:**fffff**

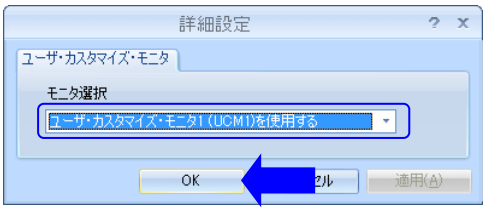
を選択します。

続けて、**[詳細設定]**ボタンを押します。



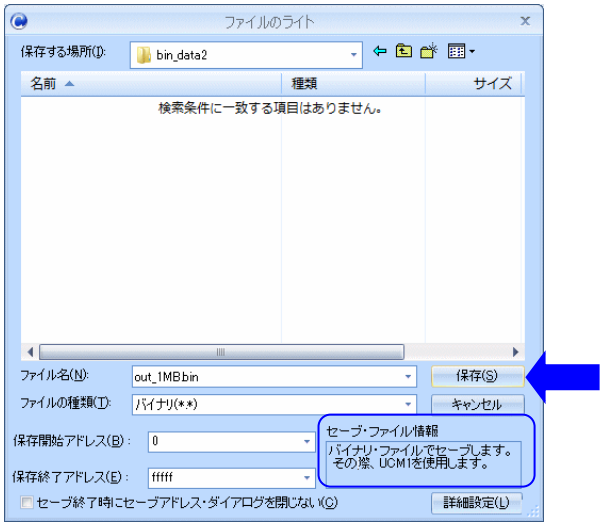
詳細設定ダイアログ・ボックスで、登録されている"**ユーザ・カスタマイズ・モニタ 1(UCM1)を使用する**"を選択します。

[OK]ボタンを押します。

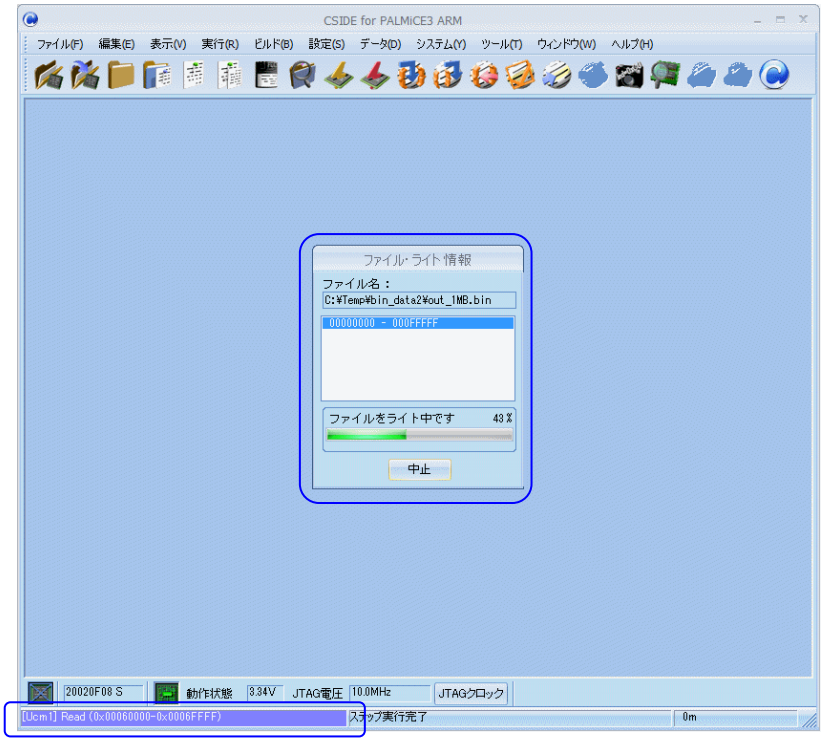


ユーザ・カスタマイズ・モニタ番号が選択されると、表示が変更されます。

表示内容を確認して、**[保存]**ボタンを押します(セーブ・ファイル情報に"**その際、UCM1 で使用します**"と表示されているか注意ください)。



ファイル・ライトが開始されると、ファイル・ライト・ダイアログ・ボックスとステータス・バーに書き込み状態が表示されます。



ファイル・ライトが完了(ファイル・ライト・ダイアログボックスが自動で閉じる)すると終了となります。

4. 使用上の注意事項

- 本機能を使用したフラッシュメモリの書き込みの結果について当社は一切の責任を負いかねますのでご了承ください。
- 本機能の内容、および仕様に関しては信頼性、設計等々の改良により将来予告なしに変更することがあります。

株式会社コンピューテックス

本 社
〒605-0846 京都市東山区五条橋東 4-432-13 對嵐坊ビル
TEL:075-551-0528(代) FAX:075-551-2585
東京営業所
TEL:03-5753-9911(代) FAX:03-5753-9917
テクニカルセンタ
TEL:075-551-0373 FAX:075-551-2585

ユーザ・カスタマイズ・モニタ機能の使用方法について
2016 年 3 月 第 4 版
CX486(D)1603