

μ T-Kernel3.0
EK-RX72N（RXv3コア）向け
実装仕様書

Version. 01. 00. 03

2025. 2. 6

目次

1. 概要	6
1.1 目的	6
1.2 対象ハードウェア	6
1.3 ターゲット名	6
1.4 関連ドキュメント	6
1.5 ソースコード構成	7
2. 基本実装仕様	8
2.1 対象マイコン	8
2.2 プロセッサモードと保護レベル	8
2.3 CPU レジスタ	8
2.4 低消費電力モードと省電力機能	9
2.5 コプロセッサ対応	9
3. メモリ	10
3.1 メモリモデル	10
3.2 マイコンのアドレス・マップ	10
3.3 OS のメモリマップ	10
3.4 スタック	11
3.5 OS 内の動的メモリ管理	11
4. 割込みおよび例外	12
4.1 マイコンの割込みおよび例外	12
4.2 ベクタテーブル	12
4.3 割込み優先度とクリティカルセクション	12
4.3.1 割込み優先度	12
4.3.2 多重割込み対応	12
4.3.3 クリティカルセクション	12
4.3.4 システムタイマの割込み優先度	13
4.3.5 各割込み優先度の関係	13
4.4 OS 内部で使用する割込み	13
4.5 μ T-Kernel/OS の割込み管理機能	13
4.5.1 割込み番号	13
4.5.2 割込みハンドラの優先度	13
4.5.3 割込みハンドラ属性	14
4.5.4 デフォルトハンドラ	14
4.6 μ T-Kernel/SM の割込み管理機能	14
4.6.1 CPU 割込み制御	14
4.6.2 割込みコントローラ制御	15

4.7	OS 管理外割込み	15
4.8	OS 管理外割込みの記述例	15
4.8.1	ベクタテーブルの変更	16
4.8.2	OS 管理外割込み関数の記述	16
5.	起動および終了処理	17
5.1	リセット処理	17
5.2	デバイスの初期化および終了処理	17
6.	その他の実装仕様	19
6.1	タスク	19
6.1.1	タスク属性	19
6.1.2	タスクの処理ルーチン	19
6.2	時間管理機能	19
6.2.1	システムタイマ	19
6.2.2	タイムイベントハンドラ	19
6.3	T-Monitor 互換ライブラリ	20
6.3.1	ライブラリ初期化	20
6.3.2	コンソール入出力	20
7.	Light センサ	21
7.1	Light センサの接続回路図	21
7.2	Light センサの制御方法	21
7.2.1	対象デバイス	21
7.2.2	デバイス名	21
7.2.3	固有機能	21
7.2.4	属性データ	21
7.2.5	固有データ	22
7.2.6	事象通知	22
7.2.7	エラーコード	22
7.3	Light センサ (ISL29034) のレジスタ	22
7.3.1	デバイスアドレス	22
7.3.2	レジスタマップ	22
7.3.3	シリアルインタフェース	23
7.4	サンプルプログラム	24
8.	タッチパネル	26
8.1	タッチパネルの接続回路図	26
8.2	タッチパネルの制御方法	26
8.2.1	対象デバイス	26
8.2.2	デバイス名	26
8.2.3	固有機能	27

8.2.4	属性データ	27
8.2.5	固有データ	27
8.2.6	事象通知	27
8.2.7	エラーコード	27
8.3	タッチパネル (FT5206) のレジスタ	27
8.3.1	デバイスアドレス	27
8.3.2	レジスタマップ	27
8.3.3	WAKE 信号	29
8.3.4	INT 信号	29
8.3.5	シリアルインタフェース	29
8.4	サンプルプログラム	30
9.	TFT ディスプレイ	34
9.1	TFT ディスプレイの接続回路図	34
9.2	TFT ディスプレイの制御方法	34
9.2	画面の定義	34
9.3	出力信号	37
9.4	フレームバッファ	38
9.5	グラフィック 1、2 画面	39
9.6	矩形アルファブレンド	40
9.7	表示する画像の問題点	40
9.8	サンプルプログラム	44
9.9	キャラクタ・コンソールドライバ	48
9.9.1	対象デバイス	48
9.9.2	デバイス名	48
9.9.3	固有機能	48
9.9.4	属性データ	48
9.9.5	固有データ	48
9.9.6	事象通知	48
9.9.7	エラーコード	48
9.9.8	ヘッダファイルとサービス関数	48
9.9.9	キャラクタ・コンソールドライバが使用する資源	49
9.9.10	キャラクタ・コンソールドライバのコンフィグレーション	49
9.9.11	スタックサイズ	49
9.9.12	サンプルプログラム (usermain_tft.c)	50
10.	シリアルフラッシュ	52
10.1	シリアルフラッシュの接続回路図	52
10.2	シリアルフラッシュ (MX25L3233F)	52
10.2.1	メモリ構成	52

10.2.2 レジスタマップ	53
10.2.3 コマンド説明	55
10.3 シリアルフラッシュドライバ	61
10.3.1 対象デバイス	61
10.3.2 デバイス名	61
10.3.3 固有機能	61
10.3.4 属性データ	61
10.3.5 固有データ	63
10.3.6 事象通知	63
10.3.7 エラーコード	63
10.3.8 ヘッダファイルとサービス関数.....	63
10.3.9 シリアルフラッシュドライバが使用する資源.....	64
10.3.10 シリアルフラッシュドライバのコンフィグレーション.....	64
10.3.11 スタックサイズ	65
10.3.12 サンプルプログラム (usermain_qspi.c)	66
11. 問い合わせ先	69

1. 概要

1.1 目的

本書はEK-RX72N (RXv3コア) 向けの μ T-Kernel3.0の実装仕様を記載した実装仕様書である。対象は、TRONフォーラムから公開されている μ T-Kernel3.0 (V3.00.00) をルネサスエレクトロニクス社のRX用に改変したソースコードのうち、EK-RX72N (RX72N ENVISION KIT) 向けの実装部分である。

ハードウェアに依存しない共通の実装仕様は μ T-Kernel3.0共通実装仕様書を参照のこと。以降、単にOSと称する場合は μ T-Kernel3.0を示し、本実装と称する場合、前述の実装を示す。

1.2 対象ハードウェア

実装対象のハードウェアは以下の通りである。

分類	名称	備考
実機	EK-RX72N-	ルネサスエレクトロニクス製
搭載マイコン	RX72N R5F572NNHxFB	ルネサスエレクトロニクス製

1.3 ターゲット名

EK-RX72Nのターゲット名は以下とする。

分類	名称	対象
ターゲット名	_EK_RX72N_	
対象システム	EK-RX72N	
対象CPU	RX72N	R5F572NNHxFB
対象CPU アーキテクチャ	CPU_CORE_RXV3	RXv3コア

1.4 関連ドキュメント

OSの標準的な仕様は「 μ T-Kernel3.0仕様書」に記載される。

ハードウェアに依存しない共通の実装仕様は「 μ T-Kernel3.0共通実装仕様書」に記載される。

また、対象とするマイコンを含むハードウェアの仕様は、それぞれの仕様書などのドキュメントに記載される。以下に関連するドキュメントを記す。

分類	名称	発行
OS	μ T-Kernel3.0仕様書 (Ver. 3.00.00)	TRONフォーラム TEF020-S004-3.00.00
	μ T-Kernel3.0共通実装仕様書	TRONフォーラム TEF033-W002-191211
T-Monitor	T-Monitor仕様書	TRON フォーラム TEF-020-S002-01.00.01
搭載マイコン	RX72Nグループ ユーザーズマニュアル ハードウェア編	ルネサスエレクトロニクス

1.5 ソースコード構成

機種依存定義sysdepedディレクトリ下の本実装のディレクトリ構成を以下に示す。太文字で書かれた箇所が、本実装のディレクトリである。

— sysdepend	実装依存定義
└ ek_rx72n	EK-RX72N RX 依存部
└ :	
└ <ターゲットn>	ターゲットn 依存部
└ cpu	CPU 依存部
└ <r5f572n>	R5F572N 依存部
└ :	
└ <CPUn>	CPUn 依存部
└ core	コア依存部
└ <rxv3>	RXv3 依存部
└ :	
└ <core n>	コアn 依存部

2. 基本実装仕様

2.1 対象マイコン

実装対象のマイコンの基本的な仕様を以下に記す。

項目	内容
CPUコア	RXv3
ROM	4MB（内蔵フラッシュROM）
RAM	1MB（内蔵RAM）

2.2 プロセッサモードと保護レベル

RXv3コアはプロセッサモードとしてスーパーバイザモードとユーザモードの2種類を持っているが、本実装ではスーパーバイザモードで動作し、ユーザモードで動作することはない。

OSが提供する保護レベルは、すべて保護レベル0とみなす。カーネルオブジェクトに対して保護レベル1～3を指定しても保護レベル0を指定されたものとして扱う。

プロファイル TK_MEM_RING0 ～ TK_MEM_RING3 はすべて 0 が返される。

2.3 CPU レジスタ

本マイコンは内部レジスタとして、汎用レジスタ（R1-R15）、PC、PSW、ACC0、ACC1、FPSW、SPを有する。ただし、ACC0、ACC1とFPSWはコンフィグレーション（USE_DSPマクロとUSE_FPUマクロ）によってはタスクコンテキストの対象外となる。また、SPにはISPとUSPが存在するが、タスクコンテキストとして使用されるのはUSPである。

OSのAPI（tk_set_reg、tk_get_reg）を用いて実行中のタスクのコンテキストのレジスタ値を操作できる。APIで使用するマイコンのレジスタのセットは以下のように定義する。

(1) 汎用レジスタ T_REGS

```
typedef struct t_regs {  
    UW    r[15];          /* 汎用レジスタ R1-R15 */  
} T_REGS;
```

(2) 例外時に保存されるレジスタ T_EIT

```
typedef struct t_eit {  
    UW    pc;              /* プログラムカウンタ */  
    UW    psw;             /* プロセッサステータスワード */  
} T_EIT;
```

(3) 制御レジスタ T_CREGS

```
typedef struct t_cregs {  
    UW    acc0[3];         /* アキュムレータ */  
    UW    acc1[3];         /* アキュムレータ */  
    UW    fpsw;            /* 浮動小数点ステータスワード */  
    void   *sp;            /* スタックポインタ */  
} T_CREGS;
```


OSのAPIによって操作されるのは、実際にはスタック上に退避されたレジスタの値である。よって、実行中のタスクに操作することはできない（OS仕様上、自タスクへの操作、またはタスク独立部からのAPI呼出しは禁止されている）。

2.4 低消費電力モードと省電力機能

省電力機能はサポートしていない。プロファイル TK_SUPPORT_LOWPPOWER は FALSE である。

よって、マイコンの低消費電力モードに対応する機能は持たない。

省電力機能のAPI（low_pow、off_pow）は、/kernel/sysdepend/ek_rx72n/power_func.cに空関数として記述されている。

なお、本関数に適切な省電力処理を記述し、プロファイル TK_SUPPORT_LOWPPOWER を TRUE に変更すれば、OSの省電力機能（tk_set_pow）は使用可能となる。

2.5 コプロセッサ対応

本マイコンはコプロセッサ（倍精度浮動小数点レジスタ）として、DR0-DR15、DPSW、DCMR、DECNT、DEPCを有する。

よって、プロファイル TK_SUPPORT_FPU と TK_SUPPORT_COP0 は TRUE 、TK_SUPPORT_COPn（nは1～3）はすべて FALSE である。ただし、CC-RXのオプション設定によって、コプロセッサ（倍精度浮動小数点レジスタ）を未サポートにすることができる（詳細はuTK3.0_EK-RX72N構築手順書を参照のこと）。その場合、プロファイル TK_SUPPORT_FPU と TK_SUPPORT_COP0 は FALSE となる。

コプロセッサに関するAPI（tk_set_cpr、tk_get_cpr）で使用するコプロセッサのセットは以下のよう

倍精度浮動小数点レジスタ T_COPREGS

```
typedef struct t_copregs {
    UD    dr[16];          /* 倍精度浮動小数点データレジスタ DR0-DR15 */
    UW    dpsw;            /* 倍精度浮動小数点ステータスワード DPSW */
    UW    dcmr;            /* 倍精度浮動小数点比較結果レジスタ DCMR */
    UW    decnt;           /* 倍精度浮動小数点例外処理動作制御レジスタ DECNT */
    UW    depc;            /* 倍精度浮動小数点例外プログラムカウンタ DEPC */
} T_COPREGS;
```

3. メモリ

3.1 メモリモデル

RXv3は4G（32bit）のアドレス空間を有するが、MMU（Memory Management Unit：メモリ管理ユニット）は有さず、単一の物理アドレス空間のみである。

本実装では、プログラムは一つの実行オブジェクトに静的にリンクされていることを前提とする。OSとユーザプログラム（アプリケーションなど）は静的にリンクされており、関数呼出しが可能とする。

3.2 マイコンのアドレス・マップ

アドレス・マップは、実装対象のマイコンのアドレス・マップに従う。

以下にRX72N（R5F572NNHxFB）のアドレス・マップを記す。（詳細はマイコンの仕様書を参照のこと）。

アドレス (開始 ～ 終了)	種別	サイズ (KByte)	備考
0x00000000 ～ 0x0007FFFF	内蔵RAM	512	データ領域
0x00100000 ～ 0x00107FFF	データフラッシュメモリ	32	
0x00800000 ～ 0x0087FFFF	内蔵RAM	512	RAMディスク
0xFFC00000 ～ 0xFFFFFFFF	内蔵ROM	4096	プログラム領域

3.3 OSのメモリマップ

本実装ではマイコンの内蔵ROMおよび内蔵RAMを使用する。

OSを含む全てのプログラムのコードは内蔵ROMに配置され、実行される。以下に内蔵ROMおよび内蔵RAMのメモリマップを示す。表中でアドレスに「—」が記載された箇所はデータのサイズによりC言語の処理系にてアドレスが決定され、OS内ではアドレスの指定は行っていない。

(1) 内蔵ROMのメモリマップ

アドレス (開始 ～ 終了)	種別	内容
0xFFC00000 ～ 0xFFC003FF	割込みベクタテーブル	
0xFFC00400 ～ —	プログラムコード	C言語のプログラムコードが配置される領域
—	定数データ	C言語の定数データなどが配置される領域
0xFFFFFFFF80 ～ 0xFFFFFFFF	例外ベクタテーブル	

(2) 内蔵RAMのメモリマップ

アドレス (開始 ～ 終了)	種別	内容
0x00000000 ～ —	プログラムデータ	C言語の変数等が配置される領域
—	OS管理領域	OS内部の動的メモリ管理の領域

—	～ 0x0007FFFF	例外スタック領域	割込みハンドラなどのスタック領域
---	--------------	----------	------------------

3.4 スタック

本実装で使用するスタックには共通仕様に従い以下の種類がある。

(1) タスクスタック

割込みハンドラ以外で使用するスタックであり、タスク毎に 1 本ずつ存在する。

tk_cre_tsk発行時のスタックサイズ (T_CSTK.stksz) で指定する。

(2) 例外スタック

割込みハンドラで使用するスタックであり、システムスタックとは独立したスタック領域が割り当てられる。割込みスタックのサイズは/config/config.h(inc)の CFN_EXC_STACK_SIZE で指定する。

【備考】

各種スタックのサイズ計算方法に関しては、uTK3.0_EK-RX72N構築手順書を参照のこと。

3.5 OS 内の動的メモリ管理

OSのAPIの処理において以下のメモリが動的に確保される。

- メモリプールのデータ領域
- メッセージバッファのデータ領域
- タスクのスタック

ただし、コンフィギュレーション USE_IMALLOC が指定されていない場合は、動的メモリ管理は行われない（初期値は動的メモリ管理を行う）。OS内の動的メモリ管理に使用するOS管理メモリ領域（システムメモリ領域）は、以下のように定められる。

(1) OS管理メモリ領域の開始アドレス

コンフィギュレーション CFN_SYSTEMAREA_TOP の値が 0 以外の場合は、その値が開始アドレスとなる。コンフィギュレーション CFN_SYSTEMAREA_TOP の値が 0 の場合、プログラムが使用しているデータ領域の最終アドレスの次のアドレスが開始アドレスとなる。

(2) OS管理メモリ領域の終了アドレス

コンフィギュレーション CFN_SYSTEMAREA_END の値が 0 以外の場合は、その値が終了アドレスとなる。コンフィギュレーション CFN_SYSTEMAREA_END の値が 0 の場合、例外スタックの開始アドレスの前のアドレスが終了アドレスとなる。

4. 割込みおよび例外

4.1 マイコンの割込みおよび例外

本マイコンには以下の例外が存在する。なお、OS仕様上は例外、割込みをまとめて、割込みと称している。

割込み番号 (ベクタ番号)	割込みの種別	備考
例外ベクタテーブル (0xFFFFFFF)	リセット	OSで使用
例外ベクタテーブル (上記以外)	内部・外部割込み	OSで使用 (無限ループ)
割込みベクタテーブル (0)	BRK命令の実行	OSで使用
割込みベクタテーブル (28)	CMT0のCMIO割込み	OSで使用
割込みベクタテーブル (上記以外)	内部・外部割込み	OSの割込み管理機能で管理

4.2 ベクタテーブル

本マイコンでは前述の各種例外に対応する例外ハンドラのアドレスを設定したベクタテーブルを有する。本実装では、リセット時のベクタテーブルは/kernel/sysdepend/cpu/r5f572n/fixed_vector.src、リセット以外のベクタテーブルは/kernel/sysdepend/cpu/r5f572n/vector.srcに定義される。

4.3 割込み優先度とクリティカルセクション

4.3.1 割込み優先度

RXv3コアは、割込み優先度を4bit (0~15) の15段階 (0は除く) に設定できる (優先度の数字の大きい方が優先度は高い)。

4.3.2 多重割込み対応

本マイコンの割り込み要因プライオリティレジスタ (IPRn) の設定により、多重割込みに対応している。割込みハンドラの実行中に、より優先度の高い割込みが発生した場合は実行中の割込みハンドラに割り込んで優先度の高い割込みハンドラが実行される。

4.3.3 クリティカルセクション

本実装では、クリティカルセクションはCPU内部レジスタのPSWに最高外部割込み優先度 MAX_INT_PRIを設定することにより実現する。MAX_INT_PRI は、本OSが管理する割込みの最高の割込み優先度であり、/include/sys/sysdepend/cpu/r5f572n/sysdef.h (inc) にて以下のように定義される。

```
#define MAX_INT_PRI      (12)           // sysdef.h
MAX_INT_PRI             . EQU      (12) ; sysdef.inc
```

クリティカルセクション中は MAX_INT_PRI 以下の優先度の割込みはマスクされる。

4.3.4 システムタイマの割込み優先度

本実装では、システムタイマとしてCMT0（コンペアマッチタイマのチャンネル0）を使用するが、CMT0の割込みはTIM_INT_PRIの割込み優先度までマスクされた状態で実行される。TIM_INT_PRIはシステムタイマの割込み優先度であり、/include/sys/sysdepend/cpu/r5f572n/sysdef.h(inc)にて以下のように定義される。

```
#define TIM_INT_PRI      (10)          // sysdef.h
TIM_INT_PRI            .EQU    (10)    ; sysdef.inc
```

4.3.5 各割込み優先度の関係

OS管理外の割込み優先度、クリティカルセクションの割込み優先度、システムタイマの割込み優先度は以下の条件が満足されれば変更可能である。

OS管理外の割込み > クリティカルセクション ≥ システムタイマ

4.4 OS内部で使用する割込み

本OSの内部で使用する割込みには、以下のように本マイコンの割込みまたは例外が割り当てられる。該当する割込みまたは例外はOS以外で使用してはならない。

割込み番号 (ベクタ番号)	割込みの種別	備考
例外ベクタテーブル (0xFFFFFEE)	リセット	
割込みベクタテーブル (0)	CMT0のCMI0割込み	システムタイマとして使用
割込みベクタテーブル (28)	BRK命令の実行	割込みハンドリングに使用

4.5 μ T-Kernel/OSの割込み管理機能

本実装の割込み管理機能は、マイコンの内部・外部割込み(OS内部で使用する割込み以外)を対象とし、割込みベクタテーブルの割込みのみ割込みハンドラの管理を行う。例外ベクタテーブルの内部割込みは管理しない。

4.5.1 割込み番号

OSの割込み管理機能が使用する**割込み番号は割込みベクタテーブルのベクタ番号と同一**とする。例えば、IRQ0は割込み番号64となる。

4.5.2 割込みハンドラの優先度

割込みハンドラの優先度（当該割込みの割込み優先順位、4.3.2項参照）は、「4.3.5 各割込み優先度の関係」に記載した「使用可能なOS管理内の割込み優先度」が使用可能である。逆に同項に記載した「OS管理外の割込み優先度」は使用してはならない。

4.5.3 割込みハンドラ属性

本実装では、TA_ASM属性の割込みハンドラはサポートしていない。割込みハンドラはTA_HLNG属性を指定したC言語の関数としてのみ記述可能である。TA_HLNG属性の割込みハンドラは、割込みの発生後、OSの割込み処理ルーチン（BRK命令を使用）を経由して呼び出される。OSの割込み処理ルーチンでは割込みハンドラの実行が行われる。

なお、割込みハンドラでFPSWやACCO、ACCIを利用することは許されない。例え USE_FPU や USE_DSP のコンフィグレーションを有効にしたとしても、割込みハンドラ内での使用は禁止である。同様に割込みハンドラ内で倍精度浮動小数点レジスタを利用ことも許されない。

4.5.4 デフォルトハンドラ

割込みハンドラが未登録の状態では割込みが発生した場合はデフォルトハンドラが実行される。デフォルトハンドラは/kernel/sysdepend/cpu/core/rxv3/interrupt.cのDefault_Handler関数として実装されている。

デフォルトハンドラはプロファイル USE_EXCEPTION_DBG_MSG を有効にすることにより、デバッグ用の情報を出力する（初期設定は有効である）。

必要に応じてユーザがデフォルトハンドラを記述することにより、未定義割込み発生時の処理を行うことができる。

4.6 μ T-Kernel/SM の割込み管理機能

μ T-Kernel/SMの割込み管理機能は、CPUの割込み管理機能および割込み優先順位フラグ・レジスタの制御を行う。各APIの実装方法について以降に記す。

4.6.1 CPU 割込み制御

CPU割込み制御はマイコンのPSW（プロセッサステータスワード）を制御して実現する。

① CPU割込みの禁止（DI）

CPU割込みの禁止（DI）は、PSWのIPLに最高外部割込み優先度 MAX_INT_PRI を設定し、それ以下の優先度の割込みを禁止する。

② CPU割込みの許可（EI）

割込みの許可（EI）は、PSWのIPLの値をDI実行前に戻す。

③ CPU内割込みマスクレベルの設定（SetCpuIntLevel）

CPU内割込みマスクレベルの設定（SetCpuIntLevel）は、PSWのIPLの値を指定した割込みマスクレベルに設定する。割込みマスクレベルは 0 から 15 の値（値の大きい方が高い優先度）が指定可能である。指定したマスクレベル以下の優先度の割込みはマスクされる。また、割込みマスクレベルに 0 が指定された場合は、すべての割り込みはマスクされない。

④ CPU内割込みマスクレベルの参照（GetCpuIntLevel）

CPU内割込みマスクレベルの取得（GetCpuIntLevel）は、PSWのIPLの設定値を参照する。

4.6.2 割込みコントローラ制御

マイコン内蔵の割り込み要因プライオリティレジスタ（IPRn）、割り込み要求許可レジスタ（IERm）、割り込み要求レジスタ（IRn）の制御を行う。

① 割込みコントローラの割込み許可（EnableInt）

割り込み要求許可レジスタ（IERm）を設定し、指定された割込みを許可する。同時に割り込み要因プライオリティレジスタ（IPRn）に指定された割込み優先度を設定する。割込み優先度は 0 から 15 の値が使用可能である。

② 割込みコントローラの割込み禁止（DisableInt）

割り込み要求許可レジスタ（IERm）を設定し、指定された割込みを禁止する。

③ 割込み発生のカリア（ClearInt）

割り込み要求レジスタ（IRn）を設定し、指定された割込みが保留されていればクリアする。

④ 割込みコントローラのEOI 発行（EndOfInt）

本マイコンではEOIの発行は不要である。よって、EOI発行（EndOfInt）は何も実行しないマクロとして定義される。

⑤ 割込み発生のカリア（CheckInt）

割込み発生のカリア（CheckInt）は、割り込み要求レジスタ（IRn）を参照することにより実現する。

⑥ 割込みモード設定（SetIntMode）

未実装である（プロファイル TK_SUPPORT_INTMODE は FLASE である）。

⑦ 割込みコントローラの割込みマスクレベル設定（SetCtrlIntLevel）

本機能はないため、未実装である（プロファイル TK_SUPPORT_CTRLINTLEVEL は FLASE である）。

⑧ 割込みコントローラの割込みマスクレベル取得（GetCtrlIntLevel）

本機能はないため、未実装である（プロファイル TK_SUPPORT_CTRLINTLEVEL は FLASE である）。

4.7 OS 管理外割込み

最高外部割込み優先度 MAX_INT_PRI より優先度の高い割込みは、OS管理外割込みとなる。OS管理外割込みはOS自体の動作よりも優先して実行される。よって、OSから制御することはできない。また、管理外割込みの中でOSのAPIなどの機能を使用することも原則できない。

本実装のデフォルト設定では MAX_INT_PRI は優先度 12 と定義されている。ただし、「4.3.5 各割込み優先度の関係」の内容に従って優先度 1 から 15 に変更可能である。よって、優先度 13 から 15 以外にも優先度 2 から 15 までをOS管理外割込みとすることが可能ある。

4.8 OS 管理外割込みの記述例

OS管理外割込みは、OSの割込み処理ルーチンを介さず、割込み発生時に直接起動されなければならない。このため、以下に記載する2つの手順が必要となる。なお、以下はベクタ番号31、コンペアマッチタイマWのチャンネル1の割込み（CMTW1のCMWI1割込み）を例に紹介する。

4.8.1 ベクタテーブルの改変

ベクタテーブル/ kernel/sysdepend/cpu/r5f572n/vector.srcに登録されている当該ベクタのOS処理ルーチンのベクタアドレスを削除またはコメントアウトする。

【改変前のソースコード】

```
. RVECTOR 30, knl_inthdr_entry30      ; CMTWO  CMWIO
. RVECTOR 31, knl_inthdr_entry31      ; CMTW1  CMWI1
; . RVECTOR 32, knl_inthdr_entry32      ;
```

【改変後のソースコード】

```
. RVECTOR 30, knl_inthdr_entry30      ; CMTWO  CMWIO
; . RVECTOR 31, knl_inthdr_entry31      ; CMTW1  CMWI1
; . RVECTOR 32, knl_inthdr_entry32      ;
```

目的の行の先頭カラムに「;」を入れれば、その行の終わりまでがコメントとなり、フォント色が緑色になる。

4.8.2 OS 管理外割込み関数の記述

OS管理外の割込み関数は、CC-RXコンパイラが持つ#pragma interruptの拡張機能を使って記述する。また、割込み仕様でベクタテーブル指定 (vect=ベクタ番号) と、必要であれば多重割込み許可指定 (enable) やレジスタ括退避機能の利用 (bank=バンク番号) を行う。

(1) ベクタテーブル指定 (vect=ベクタ番号)

ベクタ番号に当該割込みのベクタ番号を指定することでベクタテーブルが生成できる。ベクタ番号は定数、またはiodefine.hのヘッダファイルをインクルードすることでVECTマクロを使うことが可能である。

(2) 多重割込み許可指定 (enable)

必要に応じて多重割込みの許可が可能である。enableを指定すれば多重割込み許可、未指定ならば多重割込み禁止となる。

(3) レジスタ括退避機能の利用 (bank=バンク番号)

必要に応じてレジスタ括退避機能の利用が可能である。bankを指定すれば汎用レジスタ等の退避・回復命令が不要となり、例外スタック領域の削減も可能となります。ただし、バンク番号は割込みレベルと同一の値を指定すること。それ以外の値を指定した際の動作は保証しない。

【OS管理外割込み関数の例】

```
#pragma interrupt cmtw1_cmw11(vect=VECT( CMTW1, CMWI1 ), bank=14, enable)
void cmtw1_cmw11(void)
{
}
}
```


5. 起動および終了処理

5.1 リセット処理

リセット処理は、マイコンのリセットベクタに登録され、マイコンのリセット時に実行される。リセット処理はRXv3コアに固有の処理であるため/kernel/sysdepend/cpu/core/rxv3/resetprg.srcのスタートアップ・ルーチン経由で/kernel/sysdepend/cpu/core/rxv3/reset_hdr.cのReset_Handler関数として実装される。

Reset_Handler関数の処理手順を以下に示す。なお、C言語のグローバル変数領域の初期化はresetprg.srcのスタートアップ・ルーチンで行われている。

(1) ハードウェア初期化 (knl_startup_device)

リセット時の必要最小限のハードウェアの初期化を行う。

knl_startup_device は「5.2 デバイスの初期化および終了処理」を参照のこと。

(2) OS初期化処理 (main) の実行

リセット処理を終了するOSの初期化処理 (main) を実行し、リセット処理は終了する。

5.2 デバイスの初期化および終了処理

デバイスの初期化および終了処理は、以下の関数として実装されている。ユーザのアプリケーションに応じて、関数の処理内容の変更は可能である。ただし、これらの関数はOSの共通部からも呼ばれるため、形式を変更してはならない。

ファイル : /kernel/sysdepend/ek_rx72n/start_dev.c

関数名	内容
knl_startup_device	デバイスのリセット リセット時の必要最小限のハードウェアの初期化を行う。 本実装では処理は未実装である。
knl_shutdown_device	デバイスの停止 周辺デバイスをすべて終了し、マイコンを終了状態とする。 本実装では、割込み禁止状態で無限ループとしている。
knl_restart_device	デバイスの再起動 周辺デバイスおよびマイコンの再起動を行う。 本実装ではデバイスの再起動には対応していない。処理のひな型のみを記述している。

ファイル : /kernel/sysdepend/ek_rx72n/devinit.c

関数名	内容
knl_init_device	デバイスの初期化 周辺デバイスの初期化を行う。

	本実装では簡易I2Cのデバイスドライバが実装済みである。
knl_start_device	デバイスの実行 デバイスドライバの登録、実行を行う。 本実装では処理は未実装である(※)。
knl_finish_device	デバイスの終了 デバイスドライバを終了する。 本実装では処理は未実装である(※)。

※ 本実装では、デバイスドライバを登録していないため、関数は何の処理も行っていない。
 デバイスドライバを登録する場合は、上記の関数に必要な処理を記述する。記述内容は、基本実装仕様書を参照のこと。

6. その他の実装仕様

6.1 タスク

6.1.1 タスク属性

タスク属性のハードウェア依存仕様を以下に示す。

属性	可否	説明
TA_COP0	○	本マイコンは倍精度浮動小数点コプロセッサを有する。
TA_FPU	○	
TA_COPn (nは1～3)	×	本マイコンは上記以外のコプロセッサは持っていない。

6.1.2 タスクの処理ルーチン

タスクの処理ルーチンの実行開始時の各レジスタの状態は以下である。これ以外のレジスタの値は不定である。

レジスタ	値	補足
PSW	0x00030000	割込み許可
FPSW	0x00000100	USE_FPU が 1 の場合
R1	第一引数	stacd タスク起動コード
R2	第二引数	*exinf タスク拡張情報
USP	タスクスタックの先頭アドレス	

また、タスク属性として TA_FPU または TA_COP0 を指定したタスクの処理ルーチンの実行開始時の倍精度浮動小数点レジスタの状態は以下である。これ以外の倍精度浮動小数点レジスタの値は不定である。

レジスタ	値	補足
DPSW	0x00000100	
DCMR	0x00000000	
DECNT	0x00000001	

6.2 時間管理機能

6.2.1 システムタイマ

本実装では、マイコン内蔵のインターバル・タイマをシステムタイマとして使用する。

システムタイマのティック時間は1ミリ秒から50ミリ秒の間で設定できる。

ティック時間の標準の設定値は1ミリ秒である。

6.2.2 タイムイベントハンドラ

タイムイベントハンドラの実行中の割込み優先度は、システムタイマの割込み優先度 TIM_INT_PRI と同じであり、タイムイベントハンドラの実行中は TIM_INT_PRI 以下の優先度の割込みはマスクされる。TIM_INT_PRI の設定値に関しては「4.3.4 システムタイマの割込み優先度」を参照。

6.3 T-Monitor 互換ライブラリ

6.3.1 ライブラリ初期化

T-Monitor互換ライブラリを使用するにあたって、ライブラリの初期化関数を実行する必要がある。
本初期化関数はOSの起動処理の中で実行される。

6.3.2 コンソール入出力

APIによるコンソール入出力の仕様を以下に示す。

項目	内容
デバイス	内蔵SCI Channel 2
ボーレート	115,200bps
データ形式	data 8bit, stop 1bit, no parity

7. Light センサ

7.1 Light センサの接続回路図

図7.1にLightセンサ（ISL29034）の接続回路図を示します。

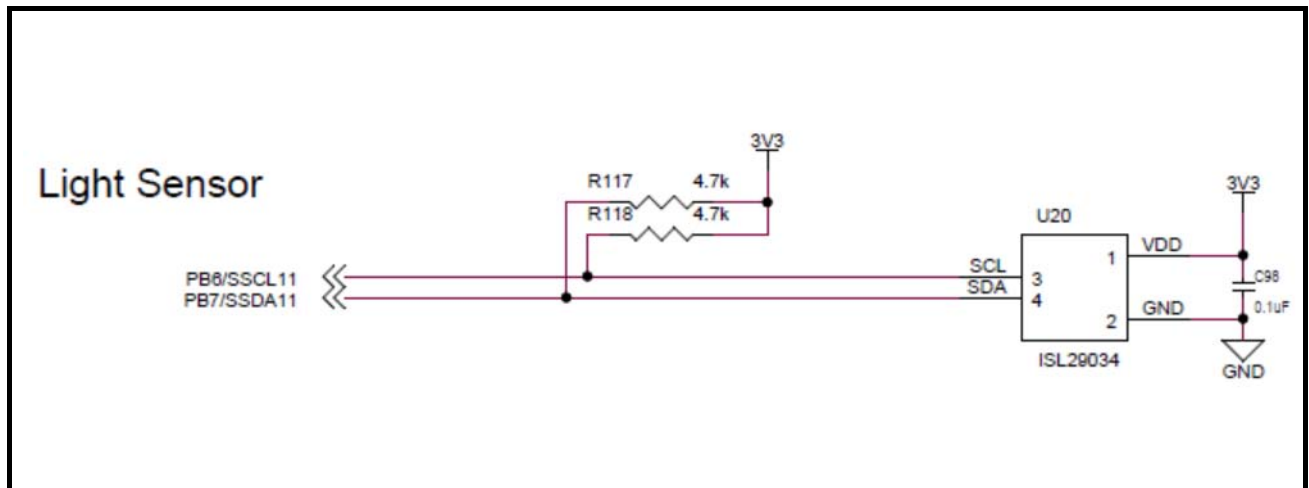


図7.1 Lightセンサの接続回路図

7.2 Light センサの制御方法

LightセンサはI2Cデバイスとして、RX72N内蔵のSCI11に接続されています。EK-RX72Nの実装では簡易I2Cデバイスドライバを介して制御することが可能です。簡易I2Cデバイスドライバの詳細に関しては、「uTK3.0_デバイスドライバインタフェース仕様書」を参照ください。

5.2節で紹介したようにEK-RX72Nの実装では簡易I2Cデバイスドライバは初期状態で利用可能です。チャンネル番号はSCI11ですから、“siicl”のデバイス名称でデバイスドライバをオープンすることになります。以下、簡易I2Cデバイスドライバの概要を示します。

7.2.1 対象デバイス

内蔵SCIチャンネルを簡易I2Cモードで利用するデバイスです。

7.2.2 デバイス名

デバイス名は“siicl”です。

7.2.3 固有機能

なし

7.2.4 属性データ

なし

7.2.5 固有データ

以下の固有データをサポートします。

開始番号 (start) : デバイスアドレス (0x44)

サイズ (size) : デバイスと送受信するデータのサイズ

データ (buf) : 送受信するデータの先頭アドレス

7.2.6 事象通知

未サポート

7.2.7 エラーコード

μT-Kernel3.0仕様書のデバイス管理機能の項を参照ください。簡易I2C固有の特殊なエラーコードは存在しません。

7.3 Light センサ (ISL29034) のレジスタ

7.3.1 デバイスアドレス

ISL29034のデバイスアドレスは“1000100”、16進数では 0x44 です。

7.3.2 レジスタマップ

ISL29034のレジスタマップを表7.1に示します。

表7.1 レジスタマップ

名称	アドレス	レジスタビット								初期値	R/W
		B7	B6	B5	B4	B3	B2	B1	B0		
COMMAND- I	0x00	OP			-	-	-	-	-	0x00	R/W
COMMAND- II	0x01	-	-	-	-	RES		RANGE		0x00	R/W
DATA _{LSB}	0x02	D7	D6	D5	D4	D3	D2	D1	D0	0x00	R
DATA _{MSB}	0x03	D15	D14	D13	D12	D11	D10	D9	D8	0x00	R
ID	0x0F	BOUT	-	1	0	1	-	-		00000000	R/W

初期状態、ISL29034はIDのB7にあるBOUTが“1”のBrownout状態となっています。使用する際はBOUTを“0”にリセットする必要があります。その後、COMMAND- I とCOMMAND- IIにより、使用用途を決定します。表7.2～表7.4にOP、RANGE、RESの設定値と意味を示します。

表7.2 OP (Operating Modes Bits)

B7	B6	B5	動作
0	0	0	パワーダウン
0	0	1	周囲光の単発測定

0	1	0	赤外光の単発測定
1	0	1	周囲光の連続測定
1	1	0	赤外光の連続測定
上記以外			設定不可

表7.3 RANGE (Range Register Bits)

B1	B0	フルスケールのルクス範囲
0	0	1,000
0	1	4,000
1	0	16,000
1	1	64,000

表7.4 RES (ADC Resolution Data Width)

B3	B2	ADCの分解能	変換時間
0	0	16 Bit	0.022 ms
0	1	12 Bit	0.352 ms
1	0	8 Bit	5.6 ms
1	1	4 Bit	105 ms

DATA_{LSB}とDATA_{MSB}の2つの読み取り専用レジスタはADCの変換結果の上位バイトと下位バイトを保持します。上位バイトはDATA_{MSB}でアクセスでき、下位バイトはDATA_{LSB}でアクセスできます。16ビット分解能でのデータはD0からD15、12ビットの分解能でのデータはD0からD11、8ビット分解能でのデータはD0からD7、4ビット分解能でのデータはD0からD3となります。レジスタは変換時間毎に更新されます。

7.3.3 シリアルインタフェース

ISL29034のシリアルインタフェースを図7.2から図7.4に示します。

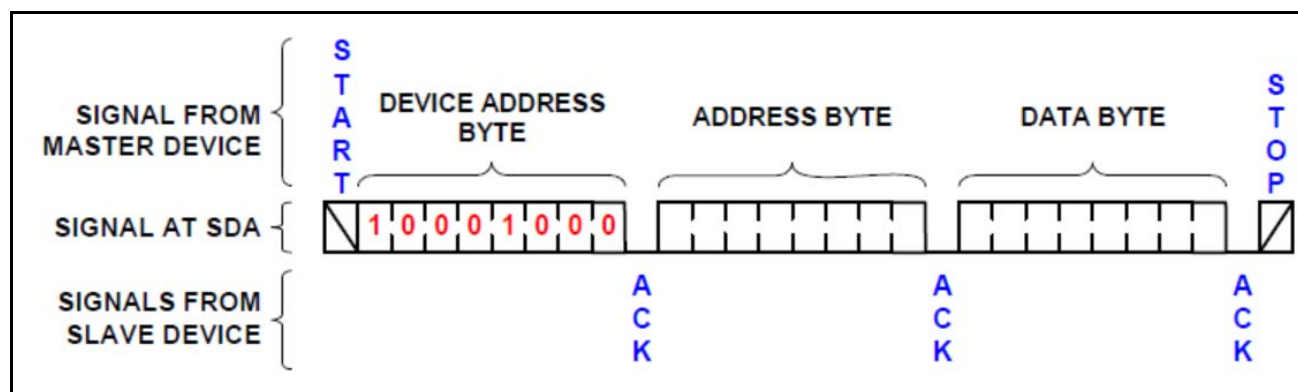


図 7.2 1バイトの書き込みシーケンス

書き込みシーケンスは1バイトのみの書き込みであり、デバイスアドレス (0x44) に続き、レジスタアドレス、書き込みデータを ISL29034 に送信します。tk_wri_dev や tk_swri_dev のシステムコールに置き換えると、デバイスアドレスを第2引数の start に指定し、レジスタアドレスと書き込みデータを第3引数で指定する buf 内に格納し、第4引数の size には2を指定することになります。

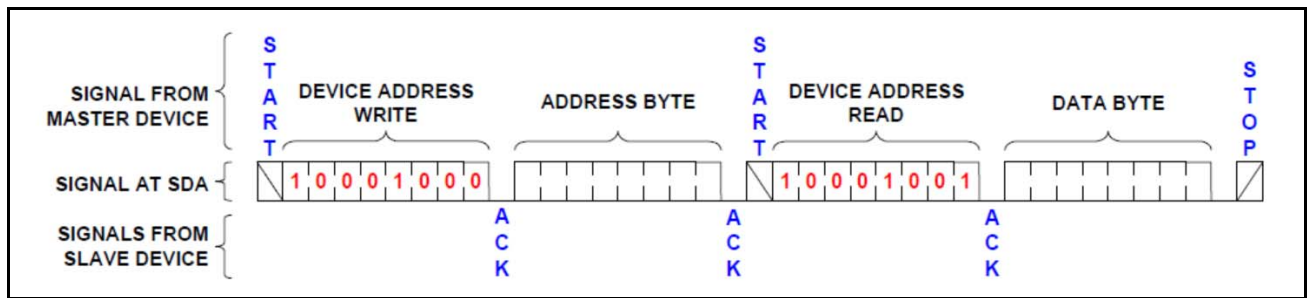


図 7.3 1 バイトの読み込みシーケンス

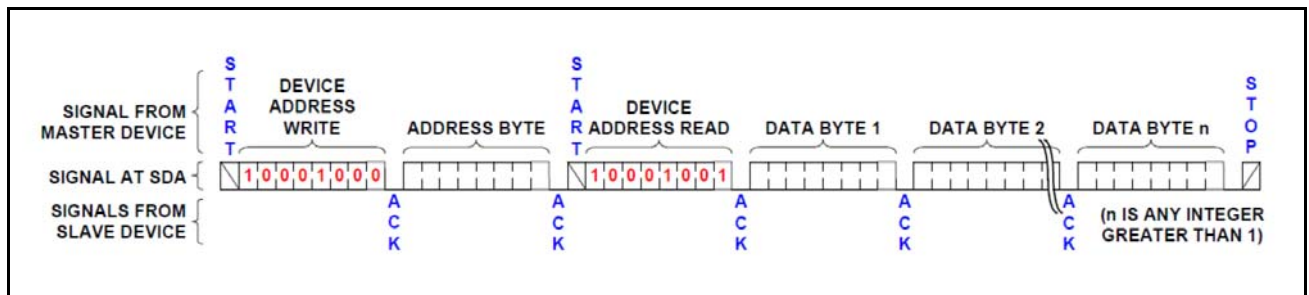


図 7.4 複数バイトの読み込みシーケンス

読み込みは1バイトも複数バイトも同じシーケンスです。まず、書き込みシーケンスにより、読み込みを行うレジスタアドレスを送信します。具体的にはデバイスアドレス（0x44）に続き、レジスタアドレスを ISL29034 に送信します。tk_wri_dev や tk_swri_dev のシステムコールに置き換えると、デバイスアドレスを第2引数の start に指定し、レジスタアドレスを第3引数で指定する buf 内に格納し、第4引数の size には1を指定することになります。

その後、読み込みシーケンスを発行します。読み込みシーケンスではデバイスアドレス（0x44）を ISL29034 に送信すると、上記の書き込みシーケンスで指定したレジスタアドレスからの内容が、STOP シーケンスを発行するまで ISL29034 から送られてきます。tk_rea_dev や tk_srea_dev のシステムコールに置き換えると、デバイスアドレスを第2引数の start に指定し、第3引数の buf には受信するデータの格納先バッファのアドレスを指定します。第4引数の size には受信するデータの長さを指定します。この第4引数で指定したサイズ分のデータを受信すると、デバイスドライバ内部で自動的に STOP シーケンスが発行され、読み込みシーケンスが終了します。

7.4 サンプルプログラム

EK-RX72N搭載のISL29034センサを赤外光の連続測定モードに設定し、測定結果をターミナルに500msの間隔で表示するサンプル（usermain_ek_rx72n_i2c.c）です。

usermain_ek_rx72n_i2c.c

```
EXPORT void sensor_tsk(INT stacd, void *exinf)
{
  ID dd;
  UB buf[2];
  SZ asize;
  INT data;
```



```

dd = tk_opn_dev( "siicl", TD_UPDATE ); // Open Simple IIC Driver
tk_swri_dev( dd, 0x44, "¥x0F¥x28", 2, &asize ); // Initialize ISL29034
tk_swri_dev( dd, 0x44, "¥x00¥xC0", 2, &asize ); // Set Measures IR Mode
while( 1 ) {
    tk_swri_dev( dd, 0x44, "¥x02", 1, &asize ); // Select Data(LSB)
    tk_srea_dev( dd, 0x44, &buf[0], 1, &asize ); // Read D7-D0
    tk_swri_dev( dd, 0x44, "¥x03", 1, &asize ); // Select Data(MSB)
    tk_srea_dev( dd, 0x44, &buf[1], 1, &asize ); // Read D15-D8
    tk_srea_dev( dd, 0x44, buf, 2, &asize ); // Read D7-D0, D15-D8
    data = ( buf[1] << 8 ) + buf[0]; // Make Sensor Value
    tm_printf( "%d¥n", data ); // Output Console
    tk_dly_tsk( 500 ); // Wait 500ms
}
}

```

デバイス名称 "siicl" をオープンします。その後、ISL29034をリセットし、赤外光の連続測定モードに設定します。

無限ループ内ではDATA_{LSB}のレジスタアドレスを指定し、そこから2バイト (DATA_{LSB} とDATA_{MSB}) を読み込み、ターミナルに表示します。なお、リストでは2バイトの連続読み込みを行っていますが、コメントのように1バイトずつの読み込みも可能です。

実行結果

```

1015
1017
1015
998
153
38
35
27
495
1038
1032
2535
8250
20649
22798
23730
26754
20025
1019
1019
1019
1021
939
963

```

8. タッチパネル

8.1 タッチパネルの接続回路図

図8.1にタッチパネル（FT5206）の接続回路図を示します。

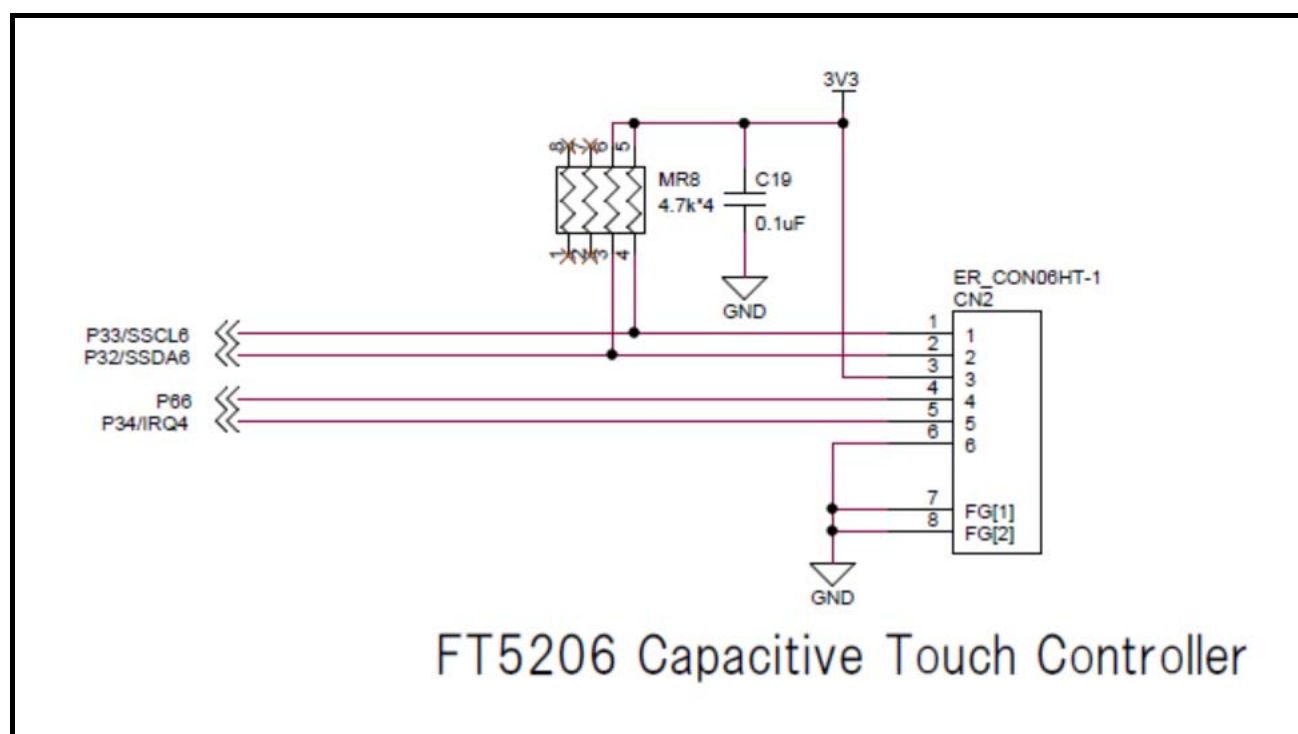


図8.1 タッチパネルの接続回路図

8.2 タッチパネルの制御方法

タッチパネルはI2Cデバイスとして、RX72N内蔵のSCI6に接続されています。EK-RX72Nの実装では簡易I2Cデバイスドライバを介して制御することが可能です。簡易I2Cデバイスドライバの詳細に関しては、「uTK3.0_デバイスドライバインタフェース仕様書」を参照ください。

5.2節で紹介したようにEK-RX72Nの実装では簡易I2Cデバイスドライバは初期状態で利用可能です。チャンネル番号はSCI6ですから、“siicg”のデバイス名称でデバイスドライバをオープンすることになります。以下、簡易I2Cデバイスドライバの概要を示します。

また、P66はホスト（RX72N）からタッチパネルのWAKE信号、IRQ4はタッチパネルからホストへのINT信号として使用します。

8.2.1 対象デバイス

内蔵SCIチャンネルを簡易I2Cモードで利用するデバイスです。

8.2.2 デバイス名

デバイス名は“siicg”です。

8.2.3 固有機能

なし

8.2.4 属性データ

なし

8.2.5 固有データ

以下の固有データをサポートします。

開始番号 (start) : デバイスアドレス (0x38)

サイズ (size) : デバイスと送受信するデータのサイズ

データ (buf) : 送受信するデータの先頭アドレス

8.2.6 事象通知

未サポート

8.2.7 エラーコード

μT-Kernel3.0仕様書のデバイス管理機能の項を参照ください。簡易I2C固有の特殊なエラーコードは存在しません。

8.3 タッチパネル (FT5206) のレジスタ

8.3.1 デバイスアドレス

FT5206のデバイスアドレスは“0111000”、16進数では 0x38 です。

8.3.2 レジスタマップ

FT5206のレジスタマップ (タッチパネル制御に必要な部分のみ) を表8.1に示します。

表8.1 レジスタマップ

名称	アドレス	レジスタビット								R/W
		B7	B6	B5	B4	B3	B2	B1	B0	
GEST_ID	0x01		Gesture ID							R
TD_STATUS	0x02					Number of touch points				R
TOUCH1_XH	0x03	1 st Event Flag				1 st Touch X Position[11:8]				R
TOUCH1_XL	0x04	1 st Touch X Position[7:0]								R
TOUCH1_YH	0x05	1 st Touch ID				1 st Touch Y Position[11:8]				R
TOUCH1_YL	0x06	1 st Touch Y Position[7:0]								R
TOUCH2_XH	0x09	2 nd Event Flag				2 nd Touch X Position[11:8]				R
TOUCH2_XL	0x0A	2 nd Touch X Position[7:0]								R

TOUCH2_YH	0x0B	2 nd Touch ID			2 nd Touch Y Position[11:8]	R
TOUCH2_YL	0x0C	2 nd Touch Y Position[7:0]				R
TOUCH3_XH	0x0F	3 rd Event Flag			3 rd Touch X Position[11:8]	R
TOUCH3_XL	0x10	3 rd Touch X Position[7:0]				R
TOUCH3_YH	0x11	3 rd Touch ID			3 rd Touch Y Position[11:8]	R
TOUCH3_YL	0x12	3 rd Touch Y Position[7:0]				R
TOUCH4_XH	0x15	4 th Event Flag			4 th Touch X Position[11:8]	R
TOUCH4_XL	0x16	4 th Touch X Position[7:0]				R
TOUCH4_YH	0x17	4 th Touch ID			4 th Touch Y Position[11:8]	R
TOUCH4_YL	0x18	4 th Touch Y Position[7:0]				R
TOUCH5_XH	0x1B	5 th Event Flag			5 th Touch X Position[11:8]	R
TOUCH5_XL	0x1C	5 th Touch X Position[7:0]				R
TOUCH5_YH	0x1D	5 th Touch ID			5 th Touch Y Position[11:8]	R
TOUCH5_YL	0x1E	5 th Touch Y Position[7:0]				R

タッチパネル上でのジェスチャー動作は GEST_ID (0x01) の Gesture ID に反映されます。表8.2に Gesture ID の一覧を示します。ジェスチャー動作だけを読み取るのであれば、Gesture ID のみを参照するだけです。

表8.2 Gesture ID

値	動作
0x01	Move Up
0x14	Move Left
0x18	Move Down
0x1C	Move Right
0x48	Zoom In
0x49	Zoom Out
0x00	No Gesture

タッチポイントの個数は TD_STATUS (0x02) の Number of touch points に反映されます。1 ～ 5 が有効な値となります（1 ならば1箇所、5 ならば5箇所タッチしている）。また、この値に従って、TOUCHn_XH、TOUCHn_XL、TOUCHn_YH、TOUCHn_YL (nは1～5) に有効なデータがあるかどうかが決まります。以下、TOUCHn_XH、TOUCHn_XL、TOUCHn_YH、TOUCHn_YLの各フィールドの意味を示します。

■Event Flag

タッチポイントのイベント内容です。Put Up が検出されることは少ないようです。また、未使用の値を読み込むこともあります（未使用値の場合は Put Up と判断しても良いかも知れません）。

表8.3 Event Flag

値		イベント
B7	B6	
0	0	Put Down
0	1	Put Up
1	0	Contact
1	1	未使用

■Touch ID

FT5206が割り振るタッチポイントのIDです。通常は 0、1、2 の順番で割り振られ、0 ～ 4 が正常値ですが、時々15 となることもあります。0 ～ 4 以外のデータは破棄した方が良いでしょう。

■Touch X Position

タッチポイントのX座標です。横480ピクセルのTFT液晶なので最大値は479となります。

■Touch Y Position

タッチポイントのY座標です。縦272ラインのTFT液晶なので最大値は271となります。

8.3.3 WAKE 信号

WAKE信号はホスト（RX72N）からFT5206へのウェイクアップ信号です。タッチパネルを使用する際はWAKE信号をMin. 5ms間 “Low” 出力を行う必要があります。WAKE信号はP66に接続されています。

8.3.4 INT 信号

INT信号はFT5206からホスト（RX72N）への割り込み信号です。有効なタッチデータの更新毎にパルスが生成されます。この様子を図8.2に示します。タッチパネルを使用する際はINT信号の立下りエッジを検出する度にシリアルインタフェースでタッチデータを読み込むことになります。INT信号はIRQ4に接続されています。

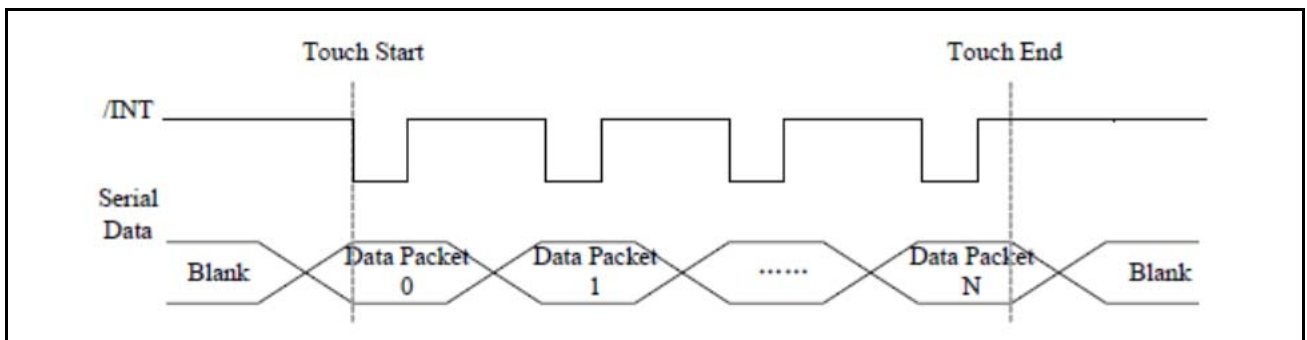


図8.2 INT信号

8.3.5 シリアルインタフェース

FT5206のシリアルインタフェースを図8.3に示します。FT5206を通常動作で使用する際はレジスタの


```

EXPORT ID ObjID[OBJ_KIND_NUM]; // IDテーブル

EXPORT void touch_tsk(INT stacd, void *exinf)
{
ID dd;
UB num, buf[4];
SZ asize;
INT i, x, y, id;

dd = tk_opn_dev( "siicg", TD_UPDATE ); // Open Simple IIC Driver
PORT6. PODR. BIT. B6 = 1; // Output Value is High
PORT6. PDR. BIT. B6 = 1; // P66 is Output (Wakeup Signal)
ICU. IRQCR[4]. BYTE = 0x04; // IRQ4 is Falling Edge
tk_dis_dsp( ); // Dispatch Disable
MPC. PWPR. BIT. BOWI = 0; // PFSWE Write Enable
MPC. PWPR. BIT. PFSWE = 1; // PmnPFS Write Enable
MPC. P34PFS. BYTE = 0x40; // P34 is IRQ4 Pin
MPC. PWPR. BYTE = 0x80; // Write Disable
tk_ena_dsp( ); // Dispatch Enable
PORT3. PMR. BIT. B4 = 1; // P34 is Peripheral Pin
EnableInt( VECT( ICU, IRQ4 ), 1 ); // IRQ4 Interrupt Level is 4
PORT6. PDR. BIT. B6 = 0; // Wakeup Signal is Low
tk_dly_tsk( 5 ); // 5ms Wait
PORT6. PDR. BIT. B6 = 1; // Wakeup Signal is High
while( 1 ) {
tm_putstring( "Gesture Mode. %n%r", // Gesture Mode
while( 1 ) {
if( tk_slp_tsk( 1000 * 10 ) == E_TMOUT ) // Wait INT Signal
break; // Mode Change
tk_swri_dev( dd, 0x38, "%x01", 1, &asize ); // Select GEST_ID
tk_srea_dev( dd, 0x38, &num, 1, &asize ); // Read Gesture ID
switch( num ) { // Gesture Check
case 0x10: tm_putstring( "Move UP" ); break;
case 0x14: tm_putstring( "Move Left" ); break;
case 0x18: tm_putstring( "Move Down" ); break;
case 0x1C: tm_putstring( "Move Right" ); break;
case 0x48: tm_putstring( "Zoom In" ); break;
case 0x49: tm_putstring( "Zoom Out" ); break;
}
if( num )
tm_putstring( "%n%r", // CR, LF
}
tm_putstring( "Touch Point Mode. %n%r", // Touch Point Mode
while( 1 ) {
if( tk_slp_tsk( 1000 * 10 ) == E_TMOUT ) // Wait INT Signal
break; // Mode Change
tk_swri_dev( dd, 0x38, "%x02", 1, &asize ); // Select TD_STATUS
tk_srea_dev( dd, 0x38, &num, 1, &asize ); // Read Number
if( num < 1 || num > 5 ) // Check Number
continue;
for( i=0 ; i<num ; i++ ) {
buf[0] = i * 6 + 3; // Make Register Address
tk_swri_dev( dd, 0x38, buf, 1, &asize ); // TOUCHn_XH
tk_srea_dev( dd, 0x38, buf, 4, &asize ); // Touch Point
switch( buf[0] & 0xC0 ) { // Event Check
case 0x00: tm_putstring( "Put Down " ); break;
case 0x40: tm_putstring( "Put Up " ); break;
case 0x80: tm_putstring( "Contact " ); break;
}
}
}
}
}

```

```

        case 0xC0: tm_putstring(""); break;
    }
    id = buf[2] >> 4; // Make Touch ID
    x = ((buf[0] & 0x0F) << 8) + buf[1]; // Make X Position
    y = ((buf[2] & 0x0F) << 8) + buf[3]; // Make Y Position
    tm_printf(" #%%d ID = %%2d, x =%%4d, y =%%4d¥n¥r", i+1, id, x, y)
    }
}

EXPORT void irq4_hdr(UINT intno)
{
    tk_wup_tsk( ObjID[TOUCH_TSK] ); // Wakeup Touch Task
}

```

デバイス名称 "siicg" をオープンします。その後、P66のWAKE信号を "High" 出力で出力ポートに初期化します。また、INT信号を利用するため、P34をIRQ4端子で使用し、立下りエッジで割込みが発生するようにします。なお、IRQ4の割込みが発生すると irq4_hdr割込みハンドラが起動するように初期化してあります。その後、WAKE信号を5ms間 "Low" 出力し、FT5206をWake Upします

無限ループ内では、先に「ジェスチャー」モードを行います。tk_slp_tskを使って、10秒間、IRQ4割込みが発生しなければ、「タッチパネル」モードに移行します。同様に「タッチパネル」モードでもtk_slp_tskを使って、10秒間、IRQ4割込みが発生しなければ、「ジェスチャー」モードに移行します。

「ジェスチャー」モードでは、GEST_IDを指定して、レジスタ値を読み込み、上下左右、ズームイン・アウトの動作を検出していれば、それをターミナルに表示します。

「タッチパネル」モードでは、TD_STATUSを指定して、レジスタ値を読み込み、タッチポイント個数を調べます。正常な個数（1～5）の範囲であれば、その個数分、タッチポイントの情報を読み込み、イベント内容、タッチポイントの番号、タッチID、X座標、Y座標をターミナルに表示します。

実行結果

```

Gesture Mode.
Move UP
Move Left
Move Right
Move Down
Zoom In
Zoom In
Zoom In
Zoom Out
Zoom Out
Touch Point Mode.
Put Down #1 ID = 0, x = 321, y = 140
Contact #1 ID = 0, x = 321, y = 140
Contact #1 ID = 0, x = 321, y = 140
Contact #1 ID = 0, x = 321, y = 140
Put Down #1 ID = 0, x = 175, y = 111
Contact #1 ID = 0, x = 175, y = 111
Put Down #2 ID = 1, x = 317, y = 140

```



```

Contact #1 ID = 0, x = 175, y = 111
Contact #2 ID = 1, x = 317, y = 140
Contact #1 ID = 0, x = 175, y = 111
Contact #2 ID = 1, x = 317, y = 140
Contact #1 ID = 0, x = 177, y = 111
Contact #2 ID = 1, x = 317, y = 140
Contact #1 ID = 0, x = 176, y = 111
Contact #1 ID = 0, x = 176, y = 111
Put Down #1 ID = 0, x = 61, y = 43
Contact #1 ID = 0, x = 61, y = 43
Contact #1 ID = 0, x = 61, y = 43
Put Down #2 ID = 1, x = 398, y = 188
Contact #1 ID = 0, x = 61, y = 43
Contact #2 ID = 1, x = 397, y = 188
Contact #1 ID = 0, x = 61, y = 43
Contact #2 ID = 1, x = 397, y = 188
Contact #1 ID = 0, x = 61, y = 43
Contact #2 ID = 1, x = 397, y = 189
Contact #1 ID = 0, x = 61, y = 43
Contact #2 ID = 1, x = 397, y = 189
Contact #1 ID = 0, x = 61, y = 43
Contact #2 ID = 1, x = 397, y = 189
Contact #1 ID = 0, x = 61, y = 43
Contact #1 ID = 0, x = 61, y = 44
Contact #1 ID = 0, x = 61, y = 44
Put Down #1 ID = 0, x = 79, y = 47
Put Down #2 ID = 1, x = 216, y = 134
Put Down #3 ID = 2, x = 385, y = 177
Contact #1 ID = 0, x = 79, y = 47
Contact #2 ID = 1, x = 216, y = 134
Contact #3 ID = 2, x = 384, y = 177
Contact #1 ID = 0, x = 79, y = 47
Contact #2 ID = 1, x = 216, y = 134
Contact #3 ID = 2, x = 384, y = 177
Contact #1 ID = 0, x = 79, y = 47
Contact #2 ID = 1, x = 216, y = 134
Contact #3 ID = 2, x = 384, y = 176
Contact #1 ID = 0, x = 79, y = 47
Contact #2 ID = 1, x = 215, y = 134
Contact #1 ID = 0, x = 79, y = 47
Contact #2 ID = 1, x = 216, y = 134
Contact #1 ID = 0, x = 79, y = 47
Contact #2 ID = 1, x = 216, y = 134
Gesture Mode.

```

9. TFT ディスプレイ

9.1 TFT ディスプレイの接続回路図

図9.1にTFTディスプレイ（ER-TFT043-3）の接続回路図を示します。

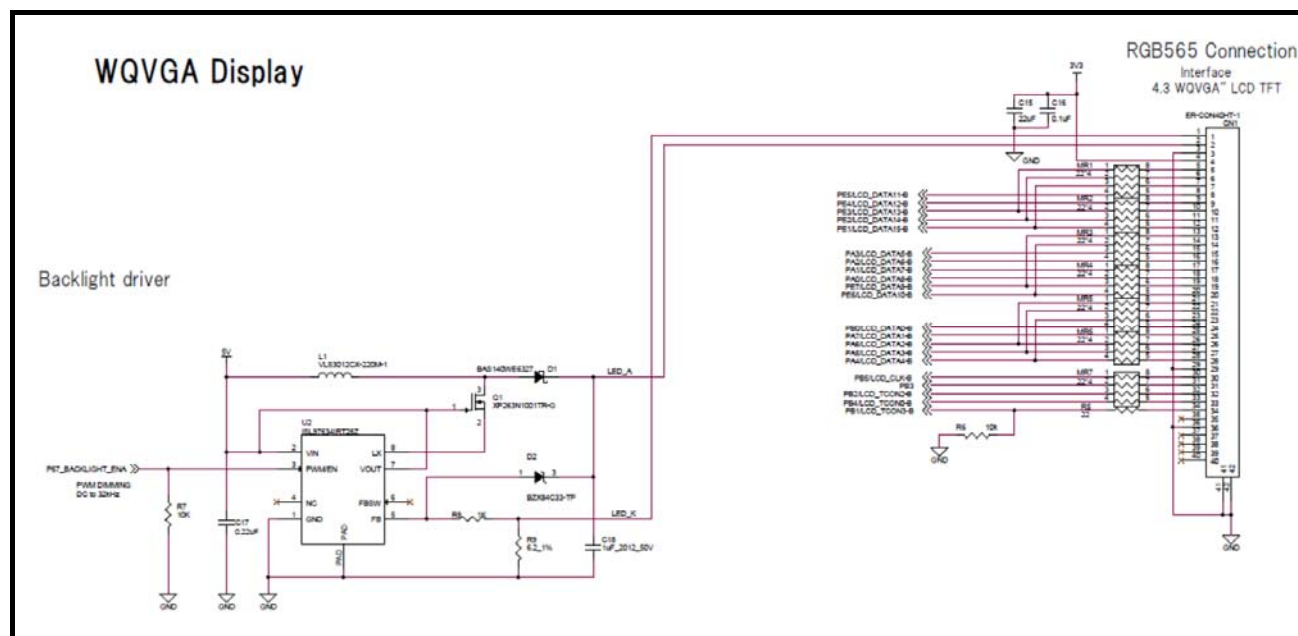


図9.1 タッチパネルの接続回路図

9.2 TFT ディスプレイの制御方法

TFTディスプレイの制御に μ T-Kernelのデバイスドライバは使用しません。ライブラリ的に利用可能なサンプルとして提供します。サンプルではタスク内で処理を行っていますが、タスクとして処理する必要もなく、また割り込みハンドラも利用していません。必要に応じて改造ください。

それと以降で紹介する理由により、**サンプルは独立したプロジェクトとして作成されています。以下のプロジェクト・アイコン（EX_RC72N_TFT.mtpj）から起動してください。**



EK_RX72N_TFT.
mtpj

9.2 画面の定義

RX72NのハードウェアマニュアルにおけるグラフィックLCDコントローラ（GLCDC）の章に記載がある通り、GLCDCのすべての動作の基本となる信号は、バックグラウンド画面生成部で生成され、グラフィック1、グラフィック2、出力制御部に順次伝播するVS（垂直同期）信号、HS（水平同期）信号、VE（垂

直表示有効) 信号、HE (水平表示有効) 信号を基準に動作します。表9. 1にER-TFT043-3搭載の ILI6480BQパネルドライバのデータシートに記載されているAC特性を示します。

表9. 1 ILI6480BQパネルドライバのAC特性

Signal	Symbol	Min	Typ	Max	Unit	サンプル値
DCLK clock time	Tclk	30			MHz	30
VSD width	Tvwh	1	–	–	Th	
VSD display area	Tvd	272			H	272
VSD period time	Tv	277	288	400	H	288
VSD back porch	Tvb	3	8	31	H	8
VSD front porch	Tvfp	2	8	97	H	7
HSD width	Thwh	1	–	–	DCLK	
HSD display area	Thd	480			CLK	480
HSD period time	Thb	520	525	800	CLK	525
HSD back porch	Thbp	36	40	255	CLK	37
HSD front porch	Thfp	4	5	65	CLK	4

各信号の関係を図9. 2に示します。

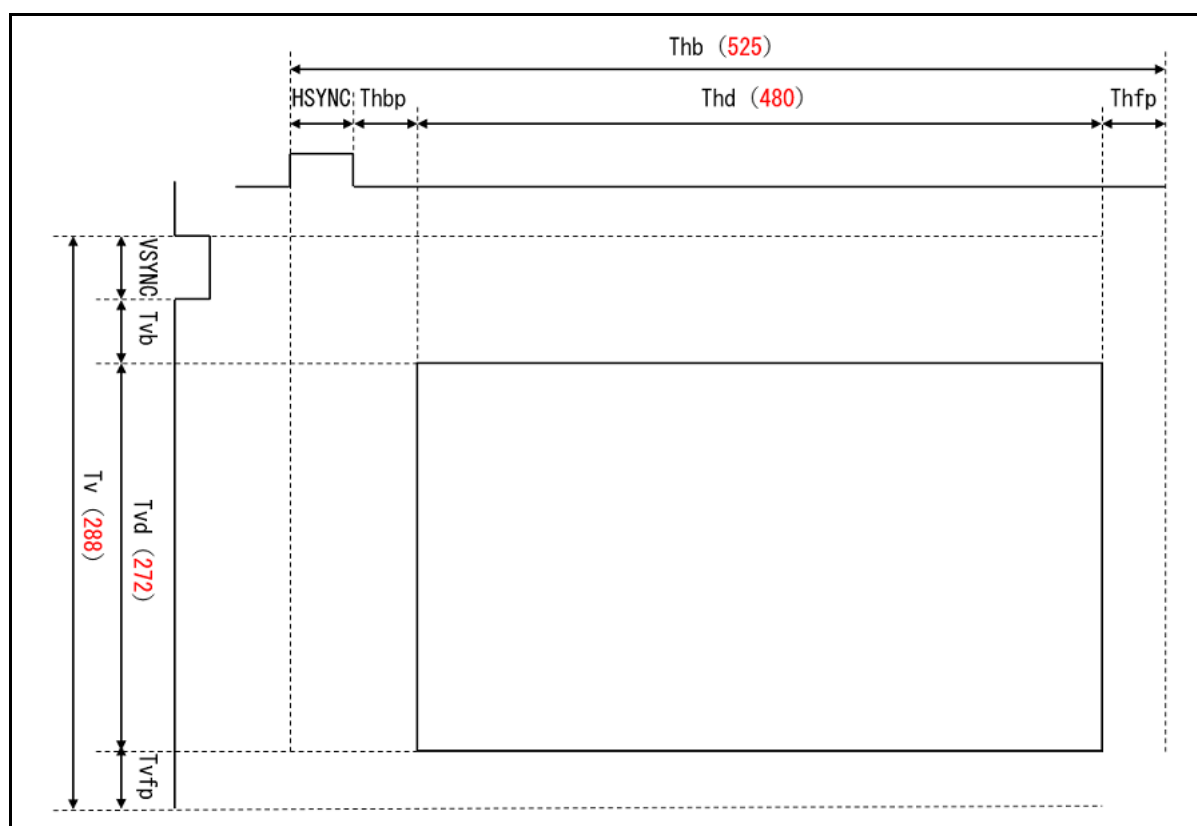


図9. 2 各信号の関係

縦・横のdisplay areaは272×480で決まりです。また、縦・横のperiod timeはTypの平均値を利用するものとして、288×525とします。残りの値に関しては、RX72N内蔵のGLCDCが要求する条件も関係することから、出来るだけTypの平均値に近づけた値（無理な場合はMinの最小値に近づけた値）を利用するものとして、GLCDCにおける画面の定義を図9.3に示します。

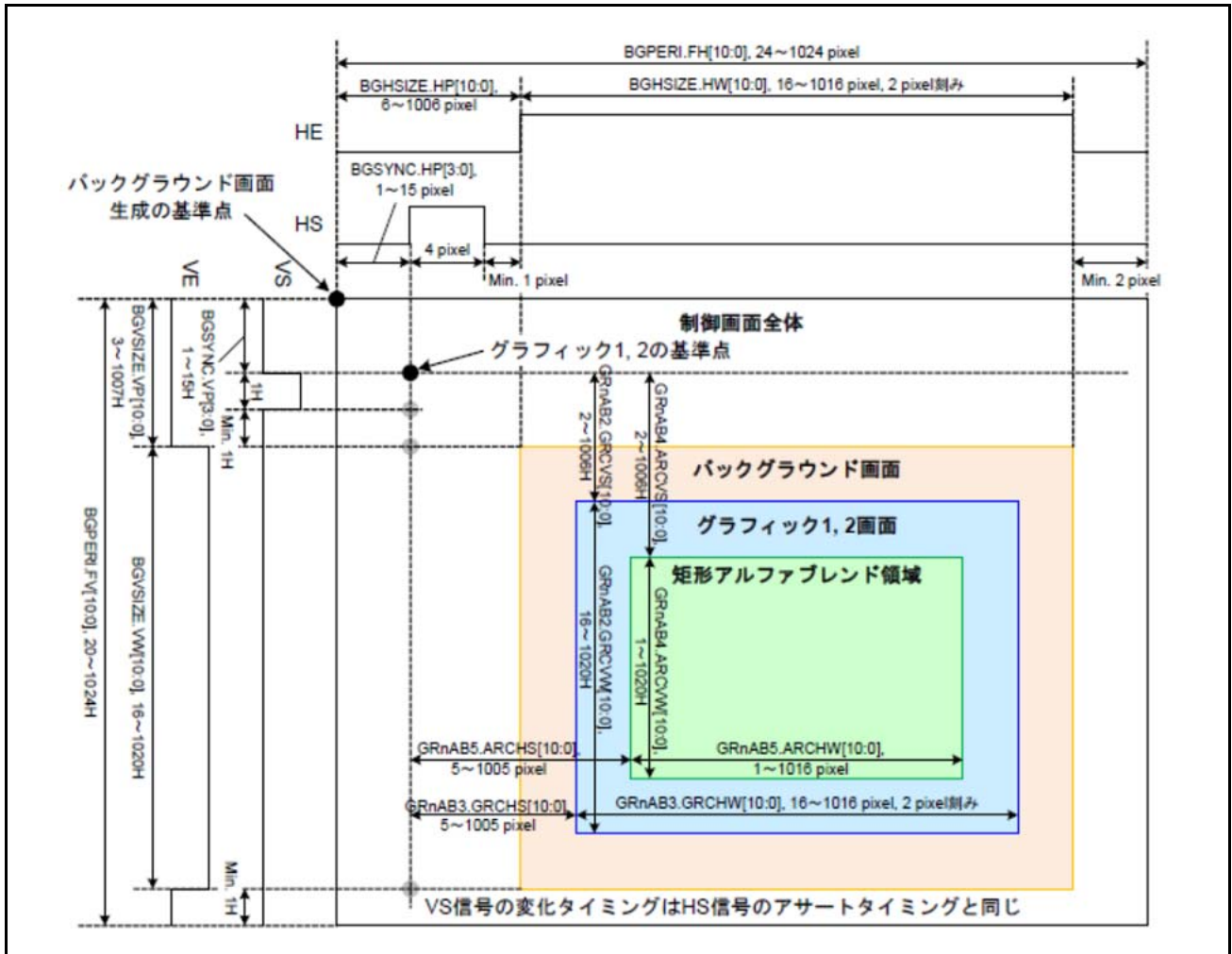


図9.3 GLCDCにおける画面の定義

図9.2と図9.3の内容から、BGPERI.FV と BGPERI.FH は 288 と 525、BGVSIZ.VW と BGHSIZE.HW は 272 と 480 となります。同様にGLCDCでは、VSYNC と HSYNC は 1 と 4 の固定値です。

図9.2で考えて、垂直方向の全ての値をTypの平均値とすると $VSYNC + Tv_b + Tv_d + Tv_{fp} = 1 + 8 + 272 + 8 = 289$ となり、 Tv のTypである 288 より 1 Line多いことが分かります。 Tv_b と Tv_{fp} では Tv_{fp} の方がMinの値が小さいため、 Tv_{fp} は 7 とします。これらの値を図9.3に当てはめ、一番下段の値をMinの 1 とし、

- ・ BGSYNC.VP は $Tv_{fp} - 1 = 7 - 1 = 6$
- ・ BGVSIZ.VP は $BGSYNC.VP + 1 + Tv_b = 6 + 1 + 8 = 15$

としました。

同様に水平方向の全ての値をMinの最小値とすると $HSYNC + Th_{pb} + Th_d + Th_{fp} = 4 + 36 + 480 + 4$

= 524 となり、Thb のTypである 525 より 1 Pixel少ないことが分かります。Thpb と Thfp では Thpb の方がTypの値の方が大きいため、Thpb は 37 とします。これらの値を図9. 3に当てはめ、一番右側の値をMinの 2 とし、

- ・ **BGSYNC. HP** は $\text{Thfp} - 2 = 4 - 2 = 2$
- ・ **BGHSIZE. HP** は $\text{BGSYNC. HP} + 4 + \text{Thbp} = 2 + 4 + 37 = 43$

としました。

以上の内容により、画面の定義に必要なGLCDCのレジスタ、BGPERI、BGSYNC、BGVSIZE、BGHSIZE への設定値が決定したことになります。

9.3 出力信号

図9. 4にGLCDCの出力制御部の構成を示します。

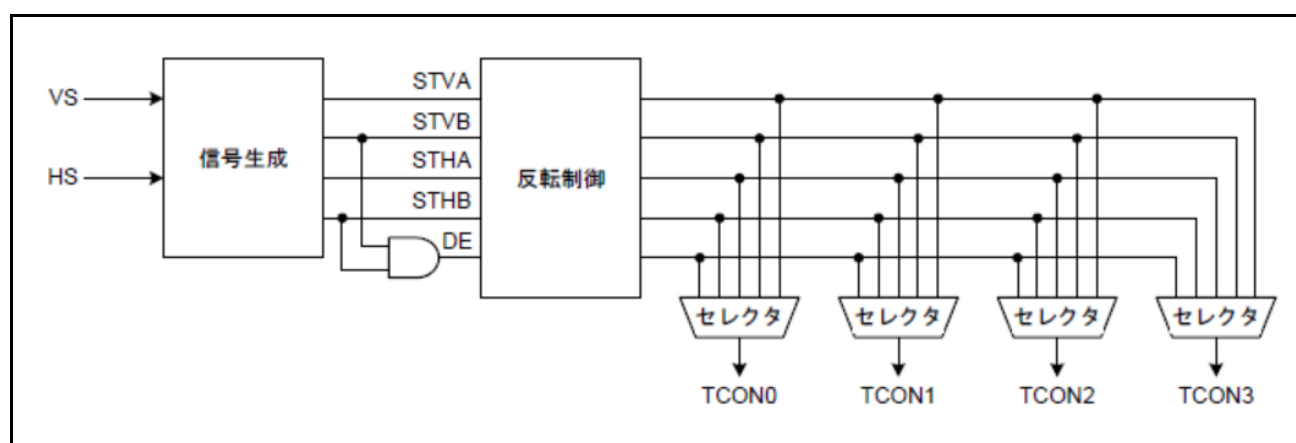


図9. 4 反転制御と出力信号選択部の構成

図9. 4から分かるようにGLCDCはTCON0端子からTCON3端子の合計4本の信号が出力可能です。一方でER-TFT043-3には表9. 2に示す3本のTCON端子と1本の制御端子が接続されています。

表9. 2 ER-TFT043-3の端子構成

Pin No	Pin Name	Descriptions	RX72N端子
31	DISP	Disp="on" when pin 31 is connected with high-level. Disp="off" when pin 31 is connected with low-level.	PB3
32	HSYNC	Horizontal sync input in RGB mode. (Short to GND if not used)	TCON2
33	VSYNC	Vertical sync input in RGB mode. (Short to GND if not used)	TCON0
34	DE	Data Enable	TCON3

DISP端子にはI/Oポート（PB3）から直接Highレベルの信号を出力します。HSYNC端子とVSYNC端子にはTCON2端子とTCON0端子からHSとVSの同期を出力します。両同期信号のタイミングは表9. 1の HSD width と VSD width が示す通り、1 DCLK (Pixel) と 1 Th (Line) のアクティブ信号を出力します。なお、コピーライトの関係で各信号のタイミングチャートを載せられませんが、図9. 3のHSとVSとは極性が逆

ですから、反転出力する必要があります。また、図9.4に示す通り、DE端子はSTVBとSTHBの論理積であり、それをTCON3端子から出力します。STVBとSTHBには図9.3のHEとVEの画面アクティブ信号を出力します。以上のことから、

9.4 フレームバッファ

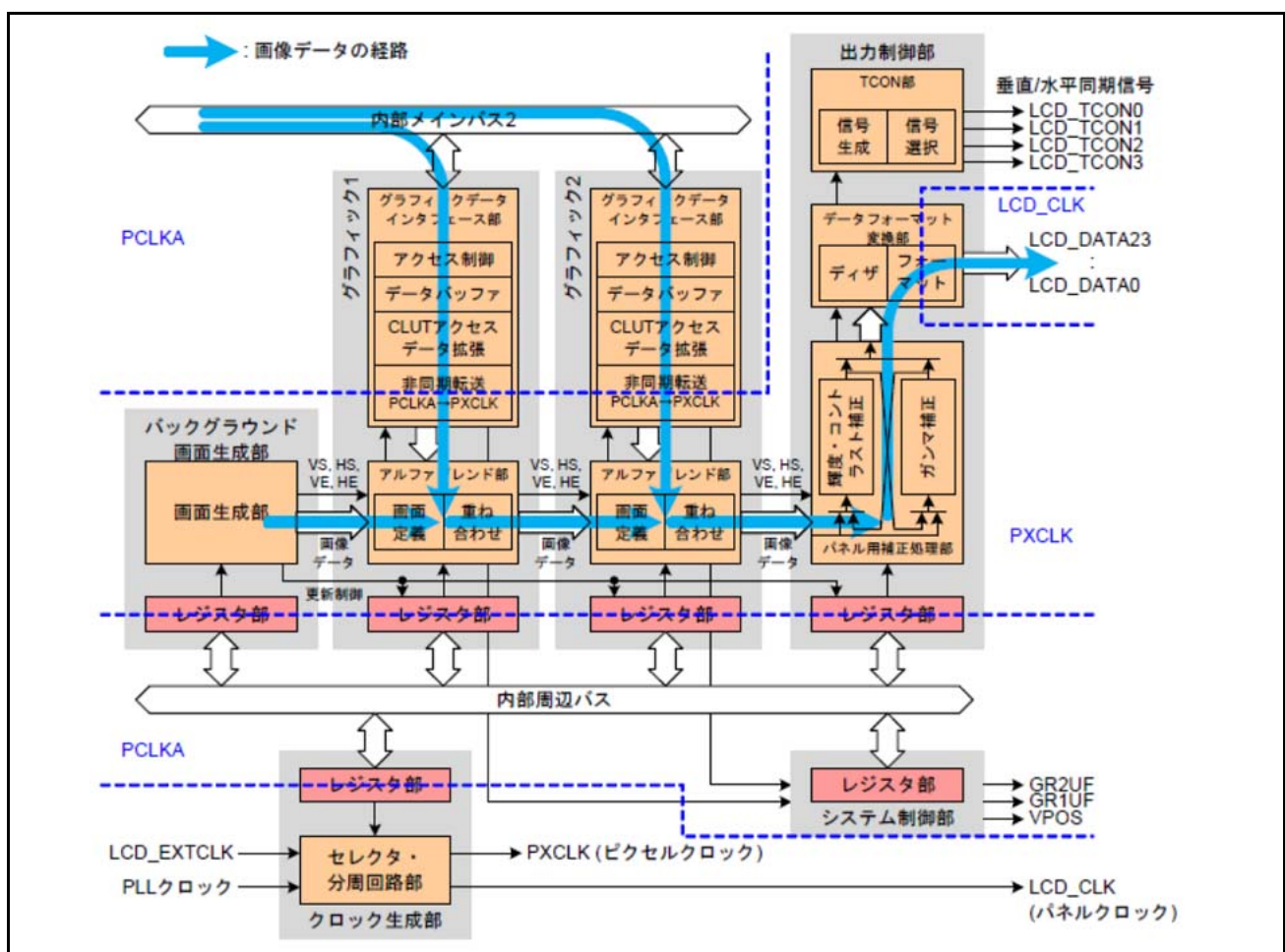


図9.4から分かるように画像データの経路は3種類あります。最下層はバックグラウンド画面です。た

だし、バックグラウンド画面は、GBCOLORレジスタに設定する単一色のみ表示可能であり、ピクセル単位に表示色を変えることはできません。次のグラフィック 1 画面とグラフィック 2 画面は単一色の表示に加え、内部メインバス 2 を介したフレームバッファの内容を表示することや、下層画像および仮想画面とアルファブレンドを施した画像を表示することも可能です。なお、**本サンプルではカラー Lookup テーブル (CLUT) とガンマ補正は利用しておらず、カラーフォーマットは RGB (565) のみです。**

RGB (565) の場合、1 ピクセル 16 ビットの 2 バイト表現です。液晶画面が 480 × 272 ですから、フレームバッファは $2 \times 480 \times 272 = 261,120$ の 255 KByte です。表示画像を変更することを行うのであれば当然 RAM 領域にフレームバッファを配置することになります。RX72N の内蔵 RAM は 2 つに分かれており、 μ T-Kernel では利用していない 0x800000 ~ 0x87FFFF の内蔵 RAM 上にフレームバッファを配置することにしました。以下にフレームバッファの宣言を示します。

```
#pragma address frmbuf1=0x00800100
UH frmbuf1[272][480];
#pragma address frmbuf2=0x00840100
UH frmbuf2[272][480];
```

各フレームバッファは #pragma address 指定により配置場所を指定してあります。通常であれば、区切りの良い 0x800000 と 0x840000 番地に配置すべきですが、以降で記載する理由により、256 バイトずらして配置しています。

9.5 グラフィック 1、2 画面

グラフィック 1、2 画面はバックグラウンド画面の一部に設定することも可能ですが、サンプルではバックグラウンド画面と同じ 480 × 272 ピクセルで定義しました。グラフィック 1 画面にはフレームバッファ 1 (frmbuf1) を表示、グラフィック 2 画面にはフレームバッファ 2 (frmbuf2) を表示するものとし、各レジスタを初期化します。図 9.3 を参照しながら、設定値を確認してください。

- ・ GR1FLM2 は frmbuf1 フレームバッファのアドレス
- ・ GR1FLM3.LNOFF は 480 × 2 の 960 (設定値は RX72N のハードウェアマニュアルに記載があります)
- ・ GR1FLM5.LNNUM は 272 - 1 の 271、GR1FLM5.DATANUM は 480 × 2 / 64 - 1 の 14 (GR1FLM3 と同様)
- ・ GR1FLM6.FORMAT は RGB (565) の 0
- ・ GR1AB2.GRCVS は 9 の 9 Line、GR1AB2.GRCVW は 272 の 272 Line
- ・ GR1AB3.GRCHS は 41 の 41 Pixel、GR1AB3.GRCHW は 480 の 480 Pixel
- ・ GR2FLM2 は frmbuf2 フレームバッファのアドレス
- ・ GR2FLM3.LNOFF は 480 × 2 の 960 (設定値は RX72N のハードウェアマニュアルに記載があります)
- ・ GR2FLM5.LNNUM は 272 - 1 の 271、GR2FLM5.DATANUM は 480 × 2 / 64 - 1 の 14 (GR2FLM3 と同様)
- ・ GR2FLM6.FORMAT は RGB (565) の 0
- ・ GR2AB2.GRCVS は 9 の 9 Line、GR2AB2.GRCVW は 272 の 272 Line
- ・ GR2AB3.GRCHS は 41 の 41 Pixel、GR2AB3.GRCHW は 480 の 480 Pixel
- ・ GR1FLMRD.RENB に 1 を設定し、フレームバッファ (frmbuf1) の内容を読み込ませる

- ・ **GR2FLMRD.RENB** に **1** を設定し、フレームバッファ（frmbuf2）の内容を読み込ませ
 なお、バックグラウンド、グラフィック 1、グラフィック 2 のどの画面が表示されるかは、**GR1AB1**
 と **GR2AB1** の **DISPSEL** で決まります。

9.6 矩形アルファブレンド

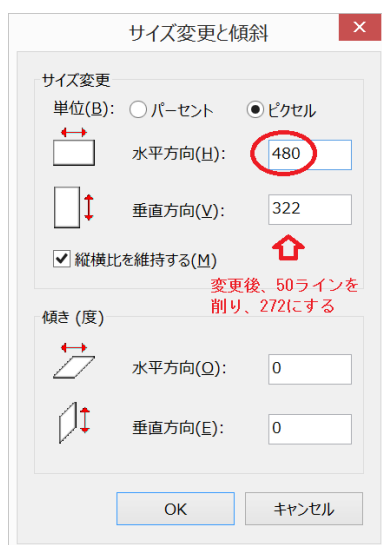
サンプルでは矩形アルファブレンドを利用し、グラフィック 1 とグラフィック 2 をブレンド表示しています。矩形アルファブレンドは表示画面の一部分だけをブレンド表示することも可能ですが、サンプルでは全画面を矩形アルファブレンド領域としています。図9.3を参照しながら、設定値を確認してください。

- ・ **GR2AB4.ARCVW** は **9** の 9 Line, **GR2AB4.ARCVS** は **272** の 272 Line
 - ・ **GR2AB5.ARCHW** は **41** の 41 Pixel, **GR2AB5.ARCHS** は **480** の 480 Pixel
- なお、矩形アルファブレンドのブレンド内容は **GR2AB6** で決まります。

9.7 表示する画像の問題点

9.4節で紹介した通り、表示する画像のフレームバッファは255KByteあります。問題は、その容量の画像をROM化するデータとして確保することがCS+の無償評価版では作成できないことです。**無償評価版はプログラム可能なROMサイズに制限があり（確か64KByte）、255KByteもある画像データはリンクできません。**従って、ROM化するためには画像データはバイナリファイル等でプログラムとは別のロードモジュールとしてダウンロードする必要があります。そこでサンプルでは、画像データはバイナリファイルとし、直接RAM領域に配置したフレームバッファにダウンロードする方法を取っています。その結果、E2 Liteのエミュレータ経由での実行は可能ですが、エミュレータを外した単独実行には対応していません。勿論、**画像データをROM化して単独実行することも可能です。もし、そのようなプログラムの作成が必要であれば、第10章の連絡先までお問い合わせください。**

以下、画像データをEK-RX72Nで表示する方法を紹介します。まずは表示したい**画像データを480×272のサイズにサイズ変更、または切り出してください。**例えば、Windows標準のペイントであれば、ホーム・メニューのサイズ変更からサイズを変更します。



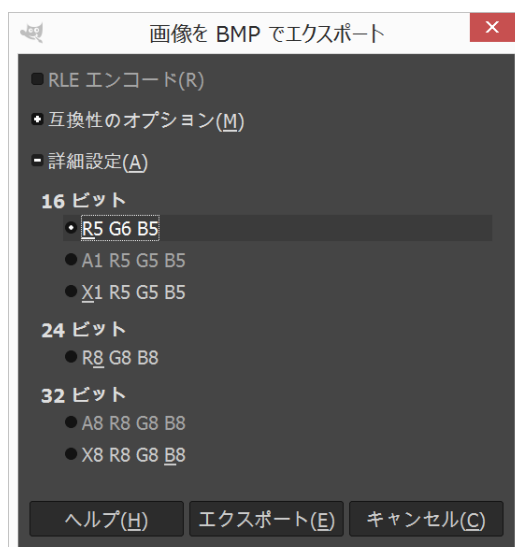
サイズを変更し、480×272になったら、当該の画像データを「24ビット ビットマップ (*.bmp)」として保存します。次に保存した画像データのフォーマットを24ビットビットマップからRGB (565)に変更します。画像処理ソフトをお持ちであれば簡単ですが、持っていられない方のために変換方法を紹介します。筆者はフリーソフトであるGIMP (GNU Image Manipulation Program) を利用しました。

Webサイトで検索すれば、直ぐに見つかると思います。筆者がダウンロードして利用したのは2. 10. 38 (リビジョン1) でした。起動後は目的の画像データを読み込んでください。読込んだら、必要に応じて画像データの反転等を行ってください。多分、上下左右の反転が必要だと思います。

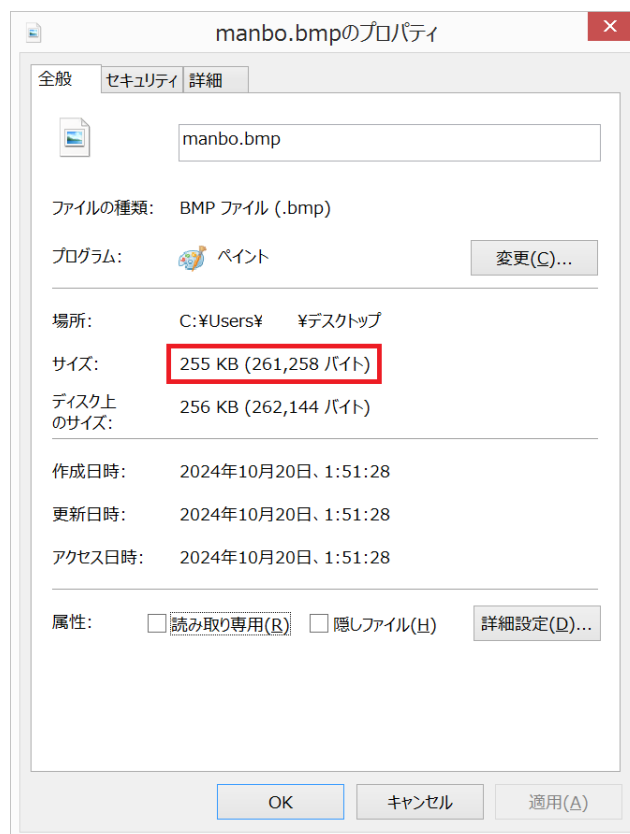
反転作業が完了したら、ファイル・メニューの「名前を付けてエクスポート」を実行します。そうすると以下のダイアログが表示されますから、「ファイル形式の選択」に必ずチェックを入れて、「エクスポート」を実行します（ファイル名は自由ですが、拡張子はbmpとしてください）。



そうすると以下のダイアログが表示されますから、「16ビット」の「R5 G6 B5」のRGB (565)を選択して「エクスポート」します。



フレームバッファに格納したいRGB(565)の画像データは、1ピクセルが16ビットの2バイトですから、 $2 \times 480 \times 272 = 261,120$ バイトです。そこで保存した画像データのサイズを確認してみます。以下に保存した画像データのプロパティを示します。



見ての通り、 $261,258 - 261,120 = 138$ バイト、容量が大きいことが分かります。これは保存した画像データ内にフォーマット形式等を示す情報が含まれていることを意味します。そこで保存した画像データをコマンドプロンプトのcertutilコマンドでダンプしてみます。

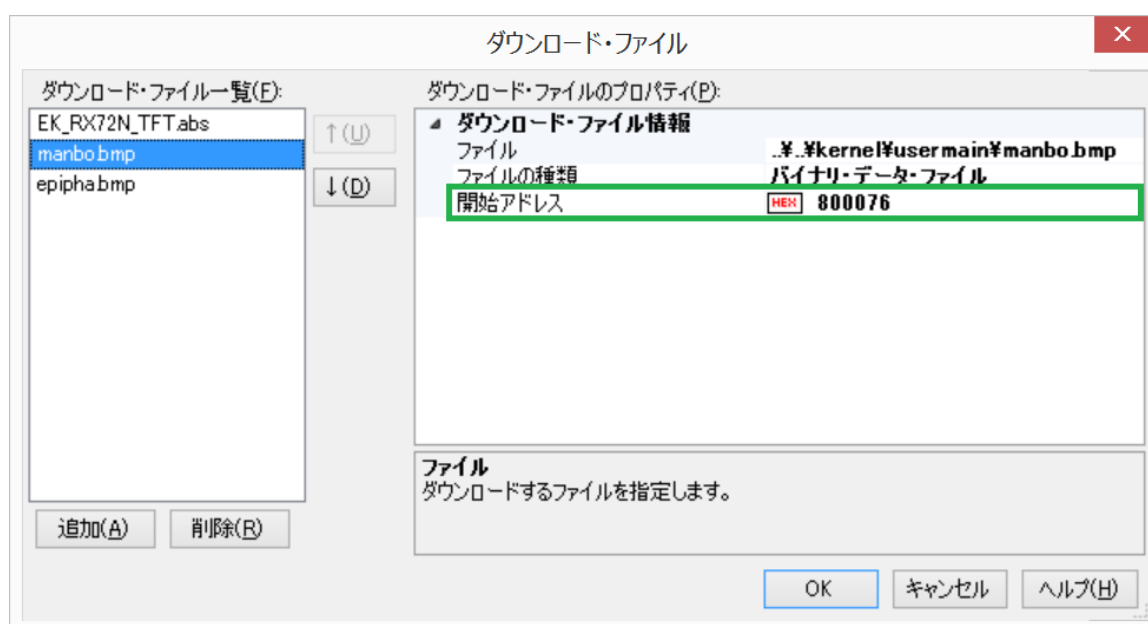
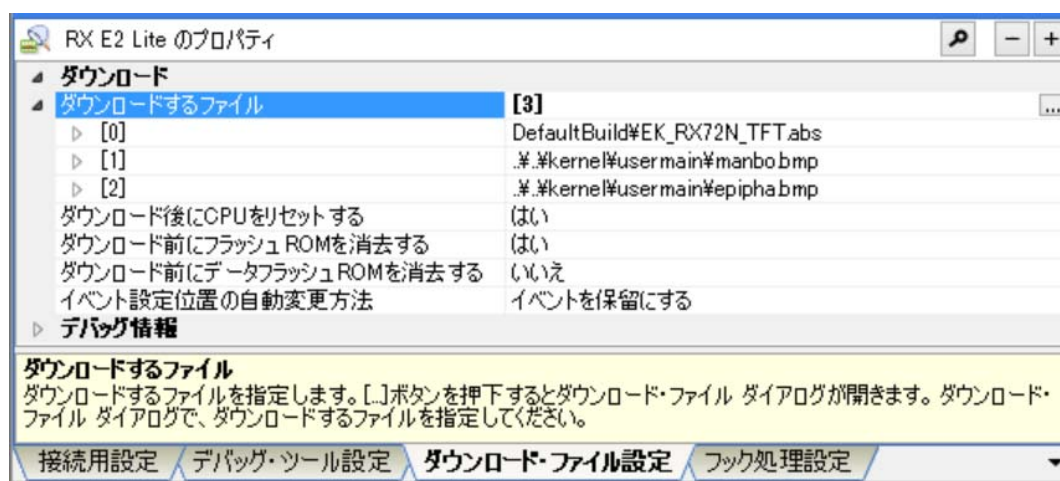
```
000000 ...
03fc8a
000000 42 4d 8a fc 03 00 00 00 00 00 8a 00 00 00 7c 00 BM.....|.
000010 00 00 e0 01 00 00 10 01 00 00 01 00 10 00 03 00 .....
000020 00 00 00 fc 03 00 13 0b 00 00 13 0b 00 00 00 00 .....
000030 00 00 00 00 00 00 00 f8 00 00 e0 07 00 00 1f 00 .....
000040 00 00 00 00 00 00 42 47 52 73 00 00 00 00 00 00 .....BGRs.....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff .....
000090 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff .....
0000a0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff .....
```

保存した画像データの最初の138バイトがフォーマット情報（緑枠の部分）です。例えば、最初の2バイトがフォーマット形式を意味する“BM”、次の4バイトがリトルエンディアンで格納されているフ

ファイルサイズ 0x0003fc8a です。また、0x12番地の 0x000001e0 が 480 のピクセル数、0x16番地の 0x00000110 が 272 のライン数等です。つまり、**最初の138バイトがフォーマット情報、それ以降がフレームバッファに格納すべき画像データということになります。**

バイナリエディタをお持ちであれば、最初の138バイトをカットすることでフレームバッファを区切りの良い番地に配置することが可能です。しかしながら、本実装では画像データは加工せず、そのまま利用することにしました。その対策が9.4節で紹介したフレームバッファのアドレスです。GLCDCの制約により、フレームバッファは64バイト境界に配置する必要があるため、フレームバッファは区切りの良い番地から256バイトずらして配置していました。そのずらした部分にフォーマット情報を格納（無駄ですが）しようという考えです。フォーマット情報は138バイト（16進0x8A）ですから、画像データを $256 - 138 = 118$ バイト（16進0x76）からダウンロードすれば、丁度画像データの本体がフレームバッファに格納されることになります。

そこでCS+のデバッグツール（RX E2 Lite）のプロパティで以下のような設定を行っています。ダウンロードするファイルの[1]である画像データのファイルの設定を示します。



ダウンロードの開始アドレスを0x800076番地にすることにより、画像データのフォーマット情報が隙間の138バイトを埋めて、画像データの本体はフレームバッファの配置アドレスである0x800100番地にダウンロードされます。

以上のようにサンプルでは特殊な設定で画像データをRAM上にダウンロードしています。画像データをROM化して単独実行するとか、本来あるべき（移植性上の問題がない）設定としたいのであれば、第10章の宛先まで問い合わせをお願い致します。

9.8 サンプルプログラム

EK-RX72N搭載のLCDパネル（ER-TFT043-3）に画像を表示し、輝度補正、コントラスト補正を行った後、矩形アルファブレンドで2つの画像を入れ替えるサンプル（usermain_ek_rx72n_display.c）です。

usermain_ek_rx72n_display.c

```
#define VPFP          ( 6UL)          // 6 Line
#define HPFP          ( 2UL)          // 2 Pixel
#define VPBP          ( 8UL)          // 8 Line
#define HPBP          (37UL)          // 37 Pixel
#define BGPERRI_FV    (VPFP+1+VPBP+BGVSIZE_VW+1) // 288 Line
#define BGPERRI_FH    (HPFP+4+HPBP+BGHSIZE_HW+2) // 525 Pixel
#define BGVSIZE_VP    (VPFP+1+VPBP)    // 15 Line
#define BGVSIZE_VW    (272UL)          // 272 Line
#define BGHSIZE_HP    (HPFP+4+HPBP)    // 43 Pixel
#define BGHSIZE_HW    (480UL)          // 480 Pixel
#define VPWH          (1UL)           // 1 Line
#define HPWH          (1UL)           // 1 Pixel

#pragma address frmbuf1=0x00800100
UH frmbuf1[272][480];
#pragma address frmbuf2=0x00840100
UH frmbuf2[272][480];

EXPORT void display_tsk(INT stacd, void *exinf)
{
    INT i;

    tk_dis_dsp( ); // Dispatch Disable
    SYSTEM.PRCR.WORD = 0xA502; // Protect Disable
    MSTP( GLCDC ) = 0; // Enable GLCDC
    SYSTEM.PRCR.WORD = 0xA500; // Protect Enable
    tk_ena_dsp( ); // Dispatch Enable
    PORT6.PODR.BIT.B7 = 1; // Set Back Light ON
    PORT6.PDR.BIT.B7 = 1; // Back Light Enable
    PORTB.PODR.BIT.B3 = 1; // Set Output Value(DISP ON)
    PORTB.PDR.BIT.B3 = 1; // PB3 is Output(DISP Enable)
    tk_dis_dsp( ); // Dispatch Disable
    MPC.PWPR.BIT.BOWI = 0; // PFSWE Write Enable
    MPC.PWPR.BIT.PFSWE = 1; // PmnPFS Write Enable
    MPC.PE7PFS.BYTE = 0x25; // PE7 is LCD_DATA9-B
    MPC.PE6PFS.BYTE = 0x25; // PE6 is LCD_DATA10-B
    MPC.PE5PFS.BYTE = 0x25; // PE5 is LCD_DATA11-B
    MPC.PE4PFS.BYTE = 0x25; // PE4 is LCD_DATA12-B
    MPC.PE3PFS.BYTE = 0x25; // PE3 is LCD_DATA13-B
    MPC.PE2PFS.BYTE = 0x25; // PE2 is LCD_DATA14-B
```

```

MPC. PE1PFS. BYTE = 0x25; // PE1 is LCD_DATA15-B
MPC. PB5PFS. BYTE = 0x25; // PB5 is LCD_CLK-B
MPC. PB4PFS. BYTE = 0x25; // PB4 is LCD_TCON0-B
MPC. PB2PFS. BYTE = 0x25; // PB2 is LCD_TCON2-B
MPC. PB1PFS. BYTE = 0x25; // PB1 is LCD_TCON3-B
MPC. PB0PFS. BYTE = 0x25; // PB0 is LCD_DATA0-B
MPC. PA7PFS. BYTE = 0x25; // PA7 is LCD_DATA1-B
MPC. PA6PFS. BYTE = 0x25; // PA6 is LCD_DATA2-B
MPC. PA5PFS. BYTE = 0x25; // PA5 is LCD_DATA3-B
MPC. PA4PFS. BYTE = 0x25; // PA4 is LCD_DATA4-B
MPC. PA3PFS. BYTE = 0x25; // PA3 is LCD_DATA5-B
MPC. PA2PFS. BYTE = 0x25; // PA2 is LCD_DATA6-B
MPC. PA1PFS. BYTE = 0x25; // PA1 is LCD_DATA7-B
MPC. PA0PFS. BYTE = 0x25; // PA0 is LCD_DATA8-B
MPC. PWPR. BYTE = 0x80; // Write Disable
PORTE. PMR. BYTE |= 0xFE; // PE7-PE1 is Peripheral Pin
PORTA. PMR. BYTE = 0xFF; // PA7-PA0 is Peripheral Pin
PORTB. PMR. BYTE |= 0x37; // PB5, PB4, PB2-PB0 is Peripheral Pin
tk_ena_dsp( ); // Dispatch Enable
GLCDC. BGEN. LONG = 0x00010000; // Software Reset
GLCDC. PANELCLK. LONG = 0x01100108; // Use PLL, 8 Division (30MHz)
GLCDC. PANELCLK. BIT. CLKEN = 1; // Enable LCD_CLK Output
while( ! GLCDC. BGMON. BIT. SWRST ) ; // Wait Software Reset End
GLCDC. OUTSET. LONG = 0x00002000; // Set Data Format is RGB(565)
GLCDC. PANELDTHA. LONG = 0x00020000;
GLCDC. BGPERI. LONG = ( BGPERI_FV << 16 ) + BGPERI_FH;
GLCDC. BGSYNC. LONG = ( VPFP << 16 ) + HPFP;
GLCDC. BGVSIZ. LONG = ( BGVSIZ_VP << 16 ) + BGVSIZ_VW;
GLCDC. BGHSIZ. LONG = ( BGHSIZ_HP << 16 ) + BGHSIZ_HW;
// GLCDC. TCONTIM. LONG = ( 0 << 16 ) + 0;
GLCDC. TCONSTVA1. LONG = ( 0 << 16 ) + VPWH;
GLCDC. TCONSTVB1. LONG = ( ( BGVSIZ_VP - VPFP ) << 16 ) + BGVSIZ_VW;
GLCDC. TCONSTVA2. LONG = ( 1 << 4 ) + 1;
GLCDC. TCONSTHA1. LONG = ( 0 << 16 ) + HPWH;
GLCDC. TCONSTHB1. LONG = ( ( BGHSIZ_HP - HPFP ) << 16 ) + BGHSIZ_HW;
GLCDC. TCONSTHA2. LONG = ( 1 << 4 ) + 2;
GLCDC. TCONSTHB2. LONG = ( 0 << 4 ) + 7;
GLCDC. BRIGHT1. LONG = 0x200;
GLCDC. BRIGHT2. LONG = ( 0x200 << 16 ) + 0x200;
GLCDC. CONTRAST. LONG = ( 0x80 << 16 ) + ( 0x80 << 8 ) + 0x80;
GLCDC. CLKPHASE. LONG = 0x00001000;
// GLCDC. BGCOLOR. LONG = 0x00000000;
// GLCDC. GR1BASE. LONG = 0x00000000;
GLCDC. GR1FLM2 = (UW) frmbuf1;
GLCDC. GR1FLM3. LONG = ( ( BGHSIZ_HW * 2 ) << 16 );
GLCDC. GR1FLM5. LONG = ( ( BGVSIZ_VW - 1 ) << 16 ) + BGHSIZ_HW * 2 / 64 - 1;
// GLCDC. GR1FLM6. LONG = 0 << 27;
GLCDC. GR1AB1. LONG = 0x00000002;
GLCDC. GR1AB2. LONG = ( ( VPBP + 1 ) << 16 ) + BGVSIZ_VW;
GLCDC. GR1AB3. LONG = ( ( HPBP + 4 ) << 16 ) + BGHSIZ_HW;
GLCDC. GR1FLMRD. LONG = 0x00000001;
// GLCDC. GR2BASE. LONG = 0x00000000;
GLCDC. GR2FLM2 = (UW) frmbuf2;
GLCDC. GR2FLM3. LONG = ( ( BGHSIZ_HW * 2 ) << 16 );
GLCDC. GR2FLM5. LONG = ( ( BGVSIZ_VW - 1 ) << 16 ) + BGHSIZ_HW * 2 / 64 - 1;
// GLCDC. GR2FLM6. LONG = 0 << 27;
GLCDC. GR2AB1. LONG = 0x00001003;
// GLCDC. GR2AB1. LONG = 0x00000001;

```

```

GLCDC. GR2AB2. LONG = ( ( VPBP + 1 ) << 16 ) + BGVSIZ_VW;
GLCDC. GR2AB3. LONG = ( ( HPBP + 4 ) << 16 ) + BGHSIZ_HW;
GLCDC. GR2AB4. LONG = ( ( VPBP + 1 ) << 16 ) + BGVSIZ_VW;
GLCDC. GR2AB5. LONG = ( ( HPBP + 4 ) << 16 ) + BGHSIZ_HW;
GLCDC. GR2FLMRD. LONG = 0x00000001;

GLCDC. BGEN. LONG = 0x00010101; // Enable Background Generating Block Operation
tk_dly_tsk( 3000 ); // Wait 3000 ms
GLCDC. DTCTEN. LONG = 0x00000001; // Enable Detection of Specified Notification
tm_putstring( " Brightness Adjustment" );
for( i=0x1FF ; 0<=i ; i-- ) { // Brightness Counter
    if( !( i & 3 ) )
        tm_printf( "%r%t%t%t%+5d", i - 0x200 );
    GLCDC. STCLR. LONG = 0x00000001; // Clear VPOS Flag
    while( GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Clear
    while( ! GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Set
    GLCDC. BRIGHT1. LONG = i; // Set G Brightness Adjustment
    GLCDC. BRIGHT2. LONG = ( i << 16 ) + i; // Set R, B Brightness Adjustment
    GLCDC. OUTVEN. LONG = 0x00000001; // Reflection Register Values
    tk_dly_tsk( 5 ); // Wait 5ms
}
for( i=1 ; 0x400>i ; i++ ) { // Brightness Counter
    if( !( i & 3 ) )
        tm_printf( "%r%t%t%t%+5d", i - 0x200 );
    GLCDC. STCLR. LONG = 0x00000001; // Clear VPOS Flag
    while( GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Clear
    while( ! GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Set
    GLCDC. BRIGHT1. LONG = i; // Set G Brightness Adjustment
    GLCDC. BRIGHT2. LONG = ( i << 16 ) + i; // Set R, B Brightness Adjustment
    GLCDC. OUTVEN. LONG = 0x00000001; // Reflection Register Values
    tk_dly_tsk( 5 ); // Wait 5ms
}
for( i=0x3FE ; 0x200<=i ; i-- ) { // Brightness Counter
    if( !( i & 3 ) )
        tm_printf( "%r%t%t%t%+5d", i - 0x200 );
    GLCDC. STCLR. LONG = 0x00000001; // Clear VPOS Flag
    while( GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Clear
    while( ! GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Set
    GLCDC. BRIGHT1. LONG = i; // Set G Brightness Adjustment
    GLCDC. BRIGHT2. LONG = ( i << 16 ) + i; // Set R, B Brightness Adjustment
    GLCDC. OUTVEN. LONG = 0x00000001; // Reflection Register Values
    tk_dly_tsk( 5 ); // Wait 5ms
}
tm_putstring( "%n%r Contrast Adjustment" );
for( i=0x7F ; 0<=i ; i-- ) { // Contrast Counter
    if( !( i & 3 ) )
        tm_printf( "%r%t%t%t0.%03d", i * 1000 / 128 );
    GLCDC. STCLR. LONG = 0x00000001; // Clear VPOS Flag
    while( GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Clear
    while( ! GLCDC. STMON. BIT. VPOS ) ; // Wait VPOS Flag is Set
    GLCDC. CONTRAST. LONG = ( i<<16 ) + ( i<<8 ) + i; // Set Contrast
    GLCDC. OUTVEN. LONG = 0x00000001; // Reflection Register Values
    tk_dly_tsk( 20 ); // Wait 20ms
}
for( i=1 ; 0x100>i ; i++ ) { // Contrast Counter
    if( !( i & 3 ) && i > 0x80 )
        tm_printf( "%r%t%t%t1.%03d", ( i - 128 ) * 1000 / 128 );
    else if( !( i & 3 ) && i < 0x80 )

```



```

        tm_printf("¥r¥t¥t¥t0.¥03d", i * 1000 / 128 );
        GLCDC. STCLR. LONG = 0x00000001;           // Clear VPOS Flag
        while( GLCDC. STMON. BIT. VPOS ) ;         // Wait VPOS Flag is Clear
        while( ! GLCDC. STMON. BIT. VPOS ) ;       // Wait VPOS Flag is Set
        GLCDC. CONTRAST. LONG = ( i<<16 ) + ( i<<8 ) + i; // Set Contrast
        GLCDC. OUTVEN. LONG = 0x00000001;         // Reflection Register Values
        tk_dly_tsk( 20 );                          // Wait 20ms
    }
    for( i=0xFE ; 0x80<=i ; i-- ) {                // Brightness Counter
        if( !( i & 3 ) )
            tm_printf("¥r¥t¥t¥t1.¥03d", ( i - 128 ) * 1000 / 128 );
            GLCDC. STCLR. LONG = 0x00000001;       // Clear VPOS Flag
            while( GLCDC. STMON. BIT. VPOS ) ;     // Wait VPOS Flag is Clear
            while( ! GLCDC. STMON. BIT. VPOS ) ;   // Wait VPOS Flag is Set
            GLCDC. CONTRAST. LONG = ( i<<16 ) + ( i<<8 ) + i; // Set Contrast
            GLCDC. OUTVEN. LONG = 0x00000001;     // Reflection Register Values
            tk_dly_tsk( 20 );                      // Wait 20ms
        }
        tm_putstring("¥r¥n Alpha Blending¥r¥n");
        GLCDC. GR2AB6. LONG = 0x00010000;         // Set Alpha Blending Parameter
        while( 1 ) {
            GLCDC. BGEN. LONG = 0x00010101;       // Enable Alpha Blending
            while( ! GLCDC. GR2MON. BIT. ARCST ) ; // Wait Alpha Blending Started
            while( GLCDC. GR2MON. BIT. ARCST ) ;   // Wait Alpha Blending Stopped
            tk_slp_tsk( 3000 );                    // Wait 3000 ms
            GLCDC. GR2AB6. LONG ^= 0x01000000;     // Change Increase or Decrease
        }
    }
}

```

GLCDCの初期化部分に関しては、9.2節から9.6節で設定値を紹介していますので説明を省略します。初期化後はBGENを設定して、バックグラウンド画面の生成を許可します。表示画面はグラフィック1とグラフィック2の矩形アルファブレンドですが、グラフィック2のアルファ値が0なので液晶に表示されるのはグラフィック1の画像です。

その後、3個のfor文で輝度補正を行います。R、G、Bの全てのチャンネルを輝度0から-512、-512から+511、+511から0に変化させます。

その後、3個のfor文でコントラスト補正を行います。R、G、Bの全てのチャンネルのコントラストを1.000から0.000、0.000から1.992、1.992から1.000に変化させます。

その後、矩形アルファブレンドでグラフィック1とグラフィック2の入れ替え表示を無限に行います。なお、本サンプルはリトルエンディアンでしか動作しません。ビッグエンディアンだと正しい動作ができません。

備考

本サンプルではJARのWebサイトで公開されていた壁画のデータを流用させて頂いております。もし、著作権の問題等がある場合は第10章に記載の問い合わせ先までご連絡ください。問題があれば直ちに画像データを変更させて頂きます。

9.9 キャラクタ・コンソールドライバ

TFTディスプレイをキャラクタ・コンソールとして利用するためのドライバを提供します。以下、キャラクタ・コンソールドライバの仕様を紹介します。

9.9.1 対象デバイス

EK-RX72NのTFTディスプレイが対象となります。

コンソールの仕様は、17行×60字、1文字は16×8ピクセルです。17行を超えると、1行ずつスクロールします。

9.9.2 デバイス名

デバイス名は、“SCREEN”です。

<dev_tft.h>をインクルードすることでTFT_DEVNMマクロで参照できます。

9.9.3 固有機能

EK-RX72NのTFTディスプレイをキャラクタ・コンソールとして利用します。

9.9.4 属性データ

なし

9.9.5 固有データ

start : 0 に固定です。0 以外の場合は E_PAR となります。

buf : ディスプレイに表示する文字列（ASCIIコード）の格納先アドレスを指定します。
size=0 の場合、文字列の終端には '¥0' を設定してください。

size : 表示する文字列のサイズを指定します。
プラスの値が指定された場合は、その文字数分を表示します。
0 が指定された場合は、文字列中に '¥0' を検出するまで表示します。
マイナスの値が指定された場合は、E_PAR となります。

9.9.6 事象通知

なし

9.9.7 エラーコード

μT-Kernel仕様書のデバイス管理機能の項を参照。キャラクタ・コンソールドライバ固有の特殊なエラーコードは存在しません。

9.9.8 ヘッダファイルとサービス関数

ヘッダファイルは dev_tft.h です。

キャラクタ・コンソールドライバを登録するためのサービス関数の仕様は以下の通りです。


```
ER ercd = tftDrvEntry(void);
```

9.9.9 キャラクタ・コンソールドライバが使用する資源

キャラクタ・コンソールドライバは、 μ T-Kernelの資源（T-Kernel/OS）は利用しません。デバイスドライバのIDを1つだけ利用します。

9.9.10 キャラクタ・コンソールドライバのコンフィグレーション

（1）キャラクタ・コンソールドライバのコンフィグレーション・ファイル

キャラクタ・コンソールドライバのコンフィグレーション・ファイルは、mtkernel_3¥device¥tft¥sysdepend¥ターゲット¥tft_config.h です。

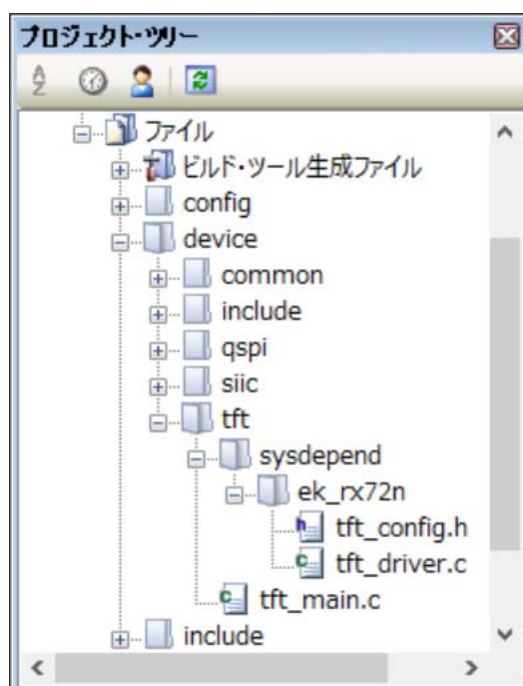


図9.1 キャラクタ・コンソールドライバのコンフィグレーション・ファイル

（2）DMACのチャンネル番号

● TFTP_CFG_DMA_CHANNEL

フレームバッファに対するデータ転送に利用するDMACのチャンネル番号です。

システムの都合に応じて、0 ～ 7 の範囲で変更できます（範囲以外の値の場合は 7 の指定となります）。デフォルトでは 7 に設定されています。

```
/* TFTPDMA channel. */  
#define TFTP_CFG_DMA_CHANNEL
```

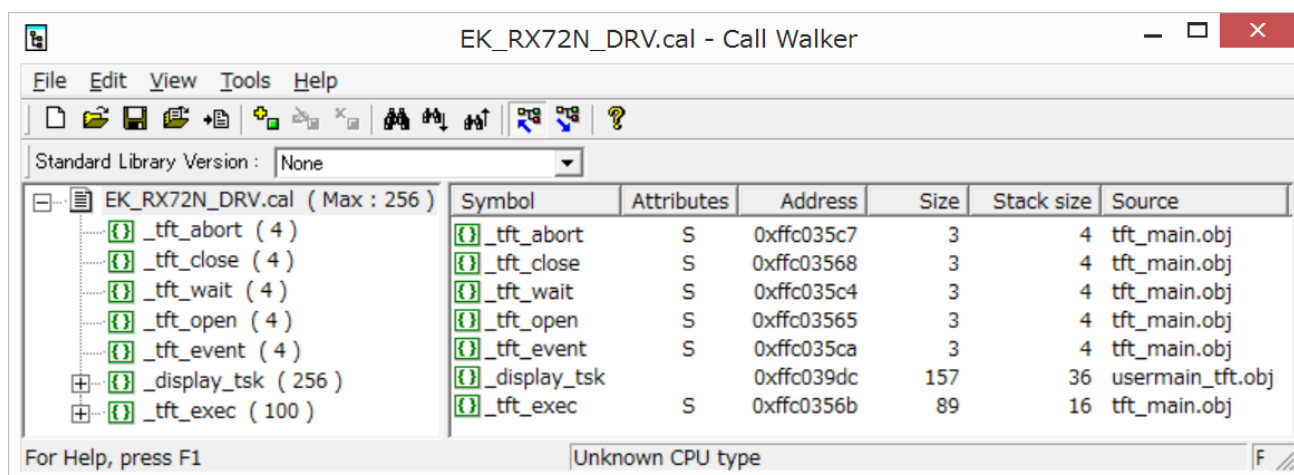
(7)

9.9.11 スタックサイズ

（1）スタック見積もりツール

本実装においてもスタック見積もりツールを使用することが可能です。スタック見積もりツールの起動方法やリアルタイムOSオプションの設定等はターゲット毎の構築仕様書の第5章を参照ください。

以降はスタック見積もりツールを使用することを前提に各スタックサイズを紹介します。以下にスタック見積もりツールの表示例を示します。



Symbol	Attributes	Address	Size	Stack size	Source
_tft_abort (4)	[] _tft_abort	S	0xffc035c7	3	4 tft_main.obj
_tft_close (4)	[] _tft_close	S	0xffc03568	3	4 tft_main.obj
_tft_wait (4)	[] _tft_wait	S	0xffc035c4	3	4 tft_main.obj
_tft_open (4)	[] _tft_open	S	0xffc03565	3	4 tft_main.obj
_tft_event (4)	[] _tft_event	S	0xffc035ca	3	4 tft_main.obj
_display_tsk (256)	[] _display_tsk		0xffc039dc	157	36 usermain_tft.obj
_tft_exec (100)	[] _tft_exec	S	0xffc0356b	89	16 tft_main.obj

図9.2 キャラクタ・コンソールドライバのスタック見積もり結果

(2) 処理関数のスタックサイズ

μT-Kernel/SMから呼び出されるデバイスドライバの処理関数（tft_open、tft_close、tft_exec、tft_wait、tft_abort、tft_event）の中でシステムコールの加算条件を超えるものはありません。このため現状のスタック解析結果では処理関数が使用するサイズは表示されないようにしてあります。詳しくはEK-RX72N構築仕様書の第5章を参照してください。

9.9.12 サンプルプログラム（usermain_tft.c）

usermain関数では、サービス関数を使ってキャラクタ・コンソールドライバを登録し、キャラクタ表示を行うディスプレイタスク（display_tsk）を生成・起動し、自身はtk_slp_tskで待ち状態となります。

usermain関数

```
#include <string.h>
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
#include "dev_tft.h"

EXPORT INT usermain( void )
{
    T_CTSK t_ctsk;
    ID objid;

    tftDrvEntry( );
    t_ctsk.tasktr = TA_HLNG | TA_DSNAME;
    t_ctsk.stksz = 1024;
    // Entry TFT Driver
    // Set Task Attribute
    // Set Task Stack size
}
```

```

t_ctsk.itskpri = 10;           // Set Task Priority
t_ctsk.task = display_tsk;    // Set Task Function Address
strcpy( t_ctsk.dsname, "display" ); // Set Object Name
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) // Create TFT Display Task
    goto ERROR;
if( tk_sta_tsk( objid, 0 ) != E_OK ) // Start TFT Display Task
    goto ERROR;
while( 1 ) tk_slp_tsk(TMO_FEVR); // Task Waiting
ERROR:
return 0;
}

```

ディスプレイタスク (display_tsk) は、キャラクタ・コンソールに対して、59文字の文字列を1文字ずつ減らしながら、合計58回表示します。その時の表示方法は、size=0 で '¥0' 文字を表示の終了として使用し、CRとLFは文字列中に予め入れてあります。

次にキャラクタ・コンソールに対して、36文字の文字列を1文字ずつ減らしながら、合計35回表示します。その時の表示方法は、size で表示する文字数を指定し、CRとLFは別のシステムコールで表示しています。これら2つの処理を無限に行います。

ディスプレイタスク (display_tsk)

```

CONST VB *buf[] = { "!¥"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUVWXYZ¥n¥r",
                    "[¥¥]^_`abcdefghijklmnopqrstuvwxyz{|}~" };

EXPORT void display_tsk(INT stacd, void *exinf)
{
ID dd;
SZ asize;
INT i;
    if( ( dd = tk_opn_dev( TFT_DEVMN, TD_WRITE ) ) <= E_OK ) // Open TFT Driver
        goto Err;
    while( 1 ) {
        for( i=0 ; i<58 ; i++ ) {
            tk_swri_dev( dd, 0, buf[0]+i, 0, &asize ); // String Output
            tm_printf(" %2d¥n", asize); // Output Act Size
            tk_dly_tsk( 1000 ); // Wait 1000ms
        }
        for( i=0 ; i<35 ; i++ ) {
            tk_swri_dev( dd, 0, buf[1], 35-i, &asize ); // String Output
            tm_printf(" %2d¥n", asize); // Output Act Size
            tk_swri_dev( dd, 0, "¥n¥r", 2, &asize ); // CR, LF Output
            tk_dly_tsk( 1000 ); // Wait 1000ms
        }
    }
Err:
    tk_ext_tsk(); // Exit
}

```

10. シリアルフラッシュ

10.1 シリアルフラッシュの接続回路図

図10.1にシリアルフラッシュ（MX25L3233F）の接続回路図を示します。

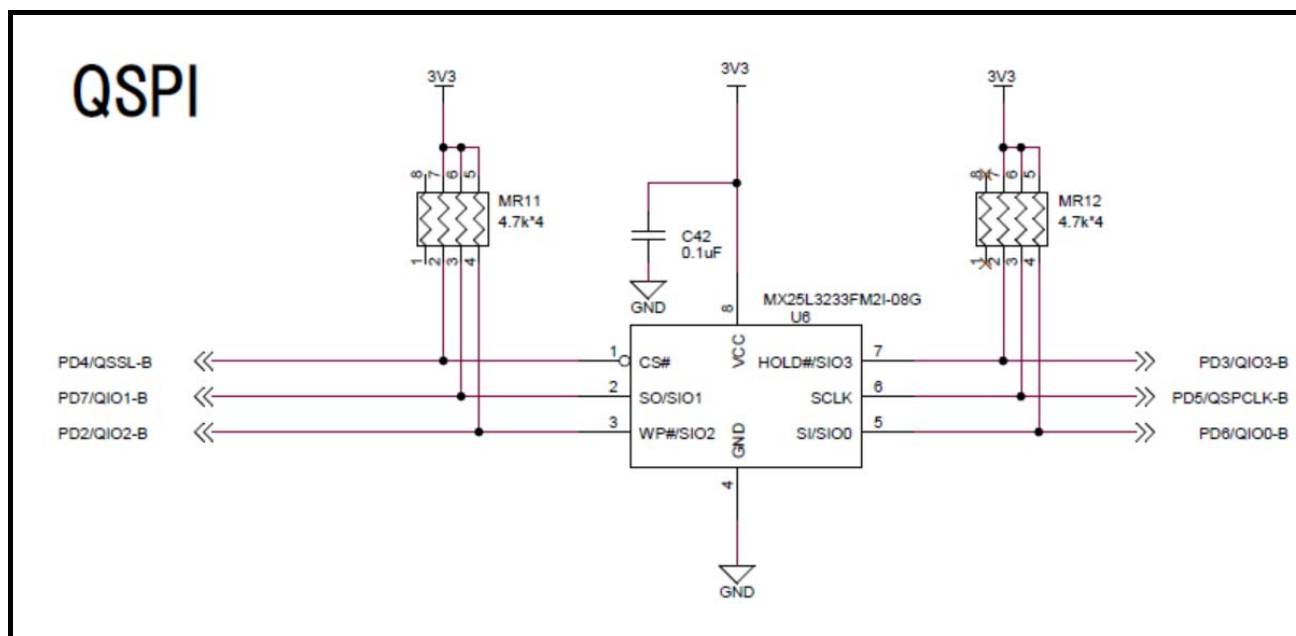


図10.1 シリアルフラッシュの接続回路図

10.2 シリアルフラッシュ（MX25L3233F）

シリアルフラッシュはQSPIデバイスとして、RX72N内蔵のQSPIに接続されています。今回、EK-RX72Nに搭載されたシリアルフラッシュ専用のデバイスドライバ（以下、本ドライバと記す）を実装しました。以下、本ドライバを使用することを前提としたシリアルフラッシュの制御方法を紹介します。

10.2.1 メモリ構成

シリアルフラッシュ（MX25L3233F）のメモリ構成を表10.1に示します。

表10.1 シリアルフラッシュ（MX25L3233F）のメモリ構成

ブロック（64KB）	ブロック（32KB）	セクタ（4KB）	アドレス
0	0	0	0x000000 - 0x000FFF
		:	:
		7	0x007000 - 0x007FFF
	1	8	0x008000 - 0x008FFF
		:	:
1	2	15	0x00F000 - 0x00FFFF
		16	0x010000 - 0x010FFF
		:	:
	3	23	0x017000 - 0x017FFF
		24	0x018000 - 0x018FFF
		:	:
		31	0x01F000 - 0x01FFFF

2	4	32	0x020000 – 0x020FFF
		:	:
		39	0x027000 – 0x027FFF
	5	40	0x028000 – 0x028FFF
		:	:
		47	0x02F000 – 0x02FFFF
:	:	:	:
:	:	:	:
61	122	976	0x3D0000 – 0x3D0FFF
		:	:
		983	0x3D7000 – 0x3D7FFF
	123	984	0x3D8000 – 0x3D8FFF
		:	:
		991	0x3DF000 – 0x3DFFFF
62	124	992	0x3E0000 – 0x3E0FFF
		:	:
		999	0x3E7000 – 0x3E7FFF
	125	1000	0x3E8000 – 0x3E8FFF
		:	:
		1007	0x3EF000 – 0x3EFFFF
63	126	1008	0x3F0000 – 0x3F0FFF
		:	:
		1015	0x3F7000 – 0x3F7FFF
	127	1016	0x3F8000 – 0x3F8FFF
		:	:
		1023	0x3FF000 – 0x3FFFFFFF

10.2.2 レジスタマップ

シリアルフラッシュ（MX25L3233F）のレジスタを表10.2と表10.5に示します。

表10.2 Status Register

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
名称	SRWD	QE	BP3	BP2	BP1	BP0	WEL	WIP

SRWD (Status Register Write Protect) ビット

SRWDビットは不揮発性ビットでデフォルト値は“0”です。SRWDビットが“1”の場合、Status Registerへの書き込み命令（WRSR命令）は無効となります。逆にSRWDビットが“0”の場合、Status Registerへの書き込み命令（WRSR命令）は有効となります。

QE (Quad Enable) ビット

QEビットはクアドモードの有効/無効を設定するものであり、本ドライバでは常に“1”の有効となります。QEビットが“1”の場合、クアドモードコマンドがシングルモードコマンドとデュアルモードコマンドとともにサポートされます。WP#/SI02端子とHOLD#/SI03端子は、それぞれSI02端子とSI03端子として機能し、それらの代替端子機能は無効になります。

BP3、BP2、BP1、BP0 (Block Protect) ビット

BP3、BP2、BP1、BP0ビットは不揮発性ビットであり、プログラム/消去命令からデバイスを保護する領域（「表10.3と表10.4 保護領域」で定義）を示します。BP3、BP2、BP1、BP0ビットに書き込むには、WRSR命令を実行する必要があります。これらのビットは、PP（ページプログラム）、SE（セクタ消去64K/32K）、BE（ブロック消去）、およびCE（チップ消去）命令からメモリを保護する領域を定義します（全てのブロック保護ビットが“0”に設定されている場合にのみ、CE命令を実行できます）。BP3、BP2、BP1、BP0ビットはデフォルトで“0”です。つまり、保護されていません。

表10.3 保護領域（TBビット = “0”）

Status Registerビット				保護領域 (ブロック64K Byte)
BP3	BP2	BP1	BP0	
0	0	0	0	なし
0	0	0	1	1ブロック、ブロック63
0	0	1	0	2ブロック、ブロック62、63
0	0	1	1	4ブロック、ブロック60～63
0	1	0	0	8ブロック、ブロック56～63
0	1	0	1	16ブロック、ブロック48～63
0	1	1	0	32ブロック、ブロック32～63
0	1	1	1	64ブロック、全ブロック
1	0	0	0	64ブロック、全ブロック
1	0	0	1	64ブロック、全ブロック
1	0	1	0	64ブロック、全ブロック
1	0	1	1	64ブロック、全ブロック
1	1	0	0	64ブロック、全ブロック
1	1	0	1	64ブロック、全ブロック
1	1	1	0	64ブロック、全ブロック
1	1	1	1	64ブロック、全ブロック

表10.4 保護領域（TBビット = “1”）

Status Registerビット				保護領域 (ブロック64K Byte)
BP3	BP2	BP1	BP0	
0	0	0	0	なし
0	0	0	1	1ブロック、ブロック0
0	0	1	0	2ブロック、ブロック0、1
0	0	1	1	4ブロック、ブロック0～3
0	1	0	0	8ブロック、ブロック0～7
0	1	0	1	16ブロック、ブロック0～15
0	1	1	0	32ブロック、ブロック0～31
0	1	1	1	64ブロック、全ブロック
1	0	0	0	64ブロック、全ブロック
1	0	0	1	64ブロック、全ブロック
1	0	1	0	64ブロック、全ブロック
1	0	1	1	64ブロック、全ブロック
1	1	0	0	64ブロック、全ブロック
1	1	0	1	64ブロック、全ブロック
1	1	1	0	64ブロック、全ブロック
1	1	1	1	64ブロック、全ブロック

WEL (Write Enable Latch) ビット

WELビットはデバイスがプログラムおよび消去命令を受け入れるかどうかを示すビットですが、**本ビットはドライバ側で制御するため、ユーザは操作できません。**

WIP (Write in Progress) ビット

WIPビットはデバイスがプログラム/消去/書き込みStatus Registerの命令進行中にビジー状態であるかどうかを示しますが、**本ビットはドライバ側で制御するため、ユーザは操作できません。**

表10.5 Configuration Register

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
名称	未使用	DC	未使用	未使用	TB	未使用	未使用	ODS

DC (Dummy Cycle) ビット

DCビットは2READ命令と4READ命令におけるダミーサイクルと周波数を表しますが、**本ドライバでは2READ命令と4READ命令をサポートしていないため、無効ビットとなります。**

TB (Top/Bottom selected) ビット

TBビットはOTP (One Time Program) ビットです。TBビットは、メモリ構成のトップまたはボトムから始まるBPビット (BP3、BP2、BP1、BP0) によって保護領域を構成するために使用されます。TBビットはデフォルトで“0”に設定されており、これはトップ領域の保護を意味します。“1”に設定すると、保護領域はボトム領域に変更されます。TBビットを書き込むには、WRSR命令を実行する必要があります。

ODSビット

ODSビットは出力ドライバ強度を示しますが、**本ビットはドライバ側で制御するため、ユーザは操作できません。**

10.2.3 コマンド説明

(1) WREN (Write Enable) 命令

WREN命令は、WELビットを設定します。PP、SE、BE、BE32K、CE、WRSRなどの命令の前には、WREN命令が必要です。ただし、**本ドライバでは、PP、SE、BE、BE32K、CE、WRSRなどの命令を発行すれば、自動的にWREN命令が発行される仕組みになっています。**

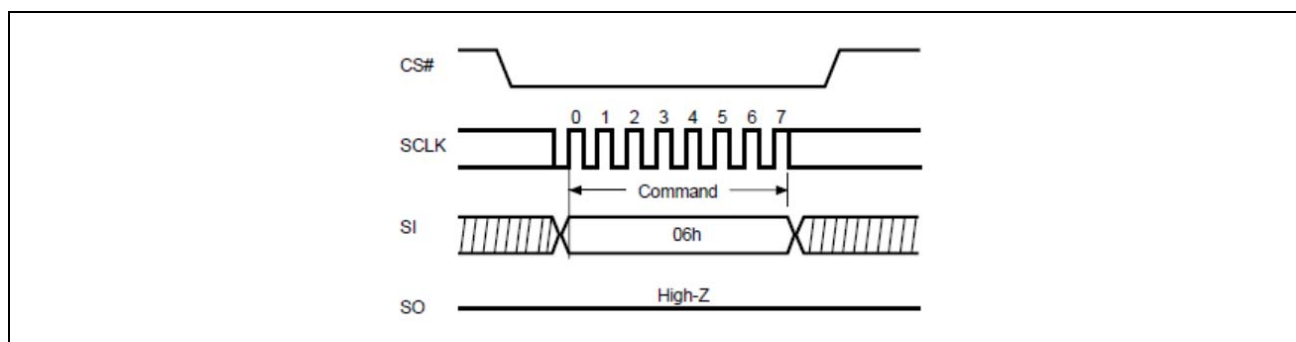


図10.2 WREN命令シーケンス

(2) WRDI (Write Disable) 命令は、WELビットをリセットします。

WRDI命令は、WELビットをリセットします。ただし、本ドライバでは、必要に応じて自動的にWRDI命令が発行される仕組みになっています。

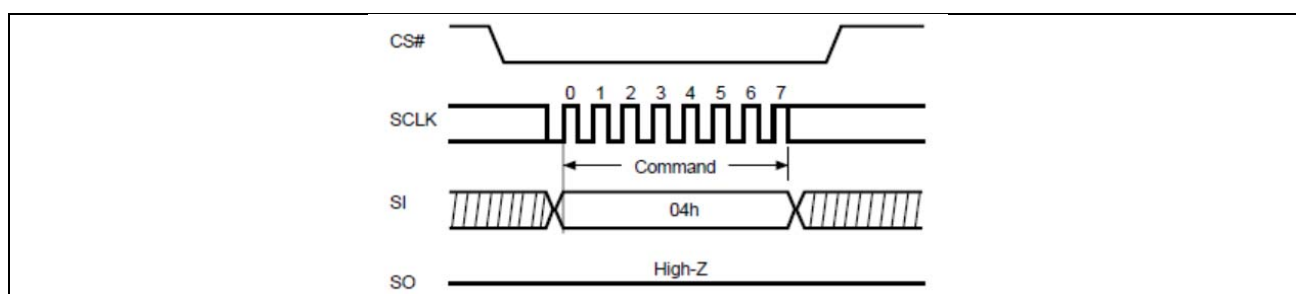


図10.3 WRDI命令シーケンス

(3) RDSR (Read Status Register) 命令

RDSR命令は、Status Registerの値を読みみます。本ドライバでは、開始番号 (start) でDN_RDSR (-102) を指定した tk_rea_dev か tk_srea_dev でRDSR命令を発行することができます。

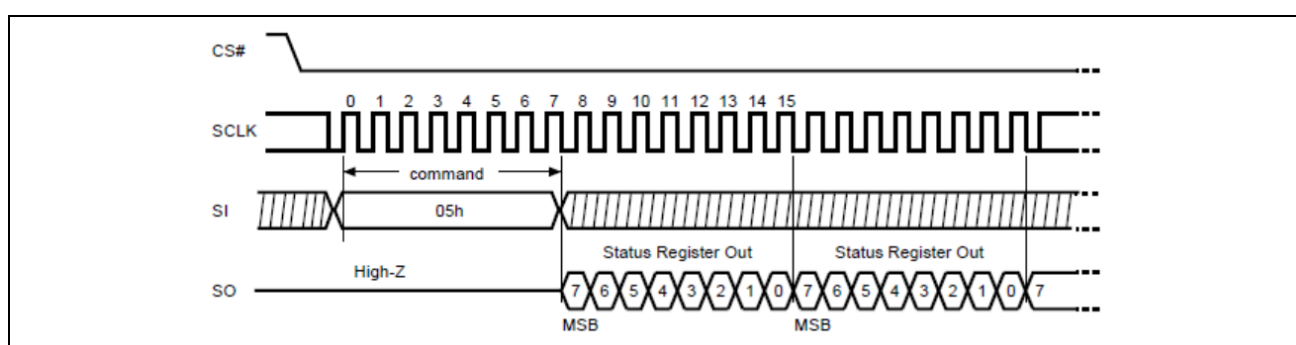


図10.4 RDSR命令シーケンス

(4) RDCR (Read Configuration Register) 命令

RDCR命令は、Configuration Registerの値を読みみます。本ドライバでは、開始番号 (start) でDN_RDCR (-103) を指定した tk_rea_dev か tk_srea_dev でRDCR命令を発行することができます。

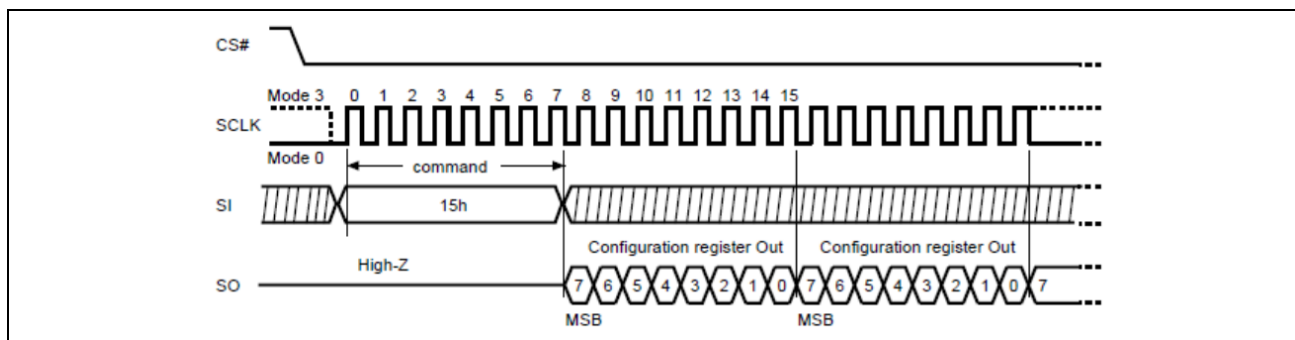


図10.5 RDCR命令シーケンス

(5) WRSR (Write Status Register) 命令

WRCR命令は、Status RegisterとConfiguration Registerに値を書込みます。この命令により、ブロック保護 (BP3、BP2、BP1、BP0、TB) ビットの値を変更して、メモリの保護領域を定義できます。本ドライバでは、開始番号 (start) でDN_WRSR (-104) を指定した `tk_wri_dev` か `tk_swri_dev` でWRSR命令を発行することができます。

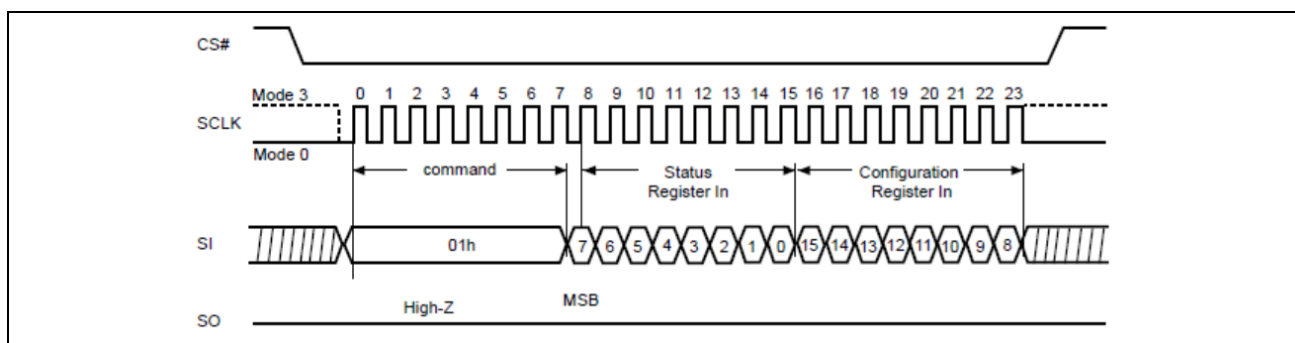


図10.6 WRSR命令シーケンス

(6) FAST_READ (Read Data Bytes at Higher Speed) 命令

FAST_READ命令は、データをSO端子のみで読むためのものです。読むアドレスは任意の場所を指定できます。各バイトデータがシフトアウトされた後、読むアドレスは自動的に次の上位アドレスに増加するため、1回のFAST_READ命令でメモリ全体を読み取ることができます。最高アドレスに達すると、読むアドレスは0にロールオーバーします。本ドライバでは、読むアドレスはリードアドレス (変数) として管理しており、開始番号 (start) でDN_RDADDR (-101) を指定した `tk_wri_dev` か `tk_swri_dev` で任意のアドレスを指定できます。また、開始番号 (start) でDN_FREAD (1) を指定した `tk_rea_dev` か `tk_srea_dev` により、リードアドレス (変数) でFAST_READ命令を発行することができます。読んだデータ分、リードアドレス (変数) は上位アドレスに増加します。

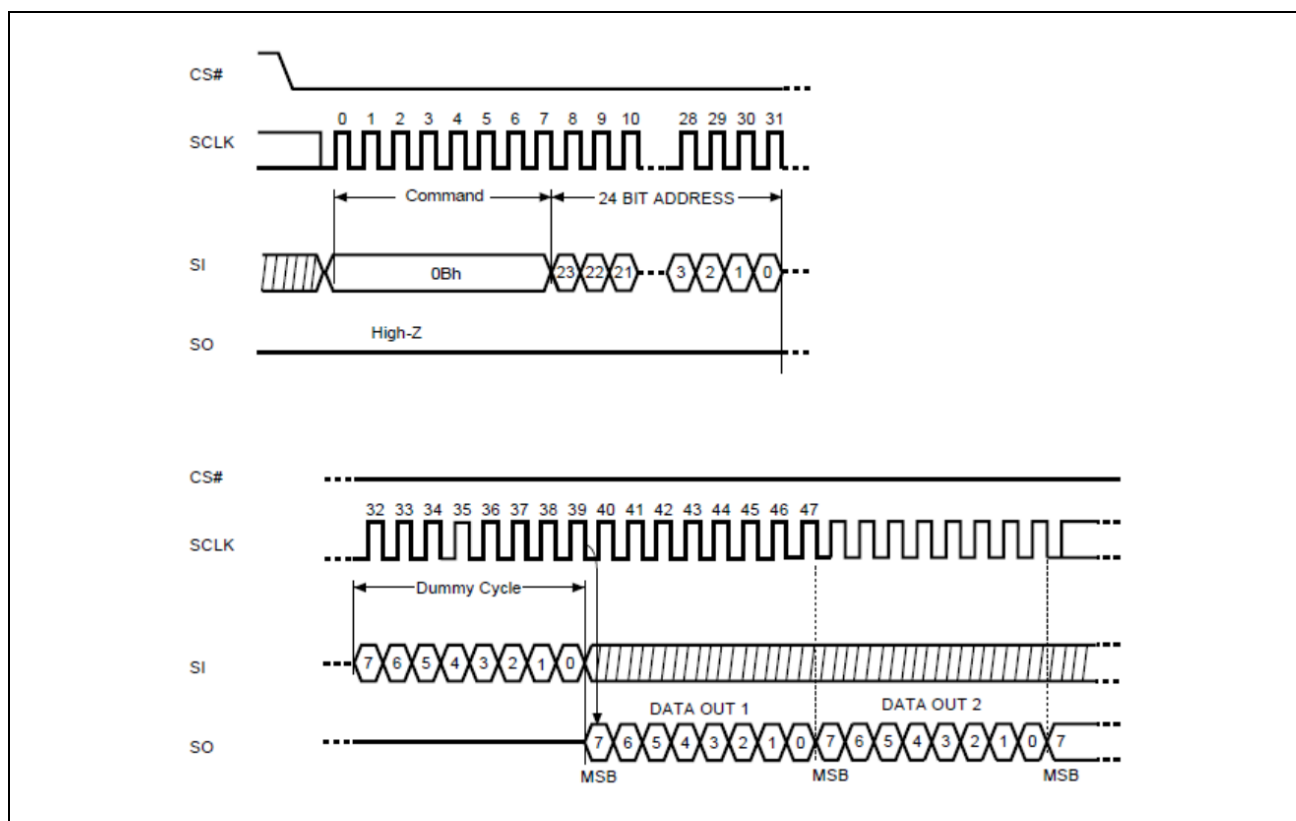


図10.7 FAST_READ命令シーケンス

(7) DREAD (Dual Read Mode) 命令

DREAD命令は、データをSI00端子とSI01端子で読込むためのものです。読込むアドレスはFAST_READ命令と同じです。本ドライバでは、読込むアドレスはリードアドレス（変数）として管理しており、開始番号（start）でDN_RDADDR（-101）を指定した `tk_wri_dev` か `tk_swri_dev` で任意のアドレスを指定できます。また、開始番号（start）でDN_DREAD（2）を指定した `tk_rea_dev` か `tk_srea_dev` により、リードアドレス（変数）でDREAD命令を発行することができます。読込んだデータ分、リードアドレス（変数）は上位アドレスに増加します。

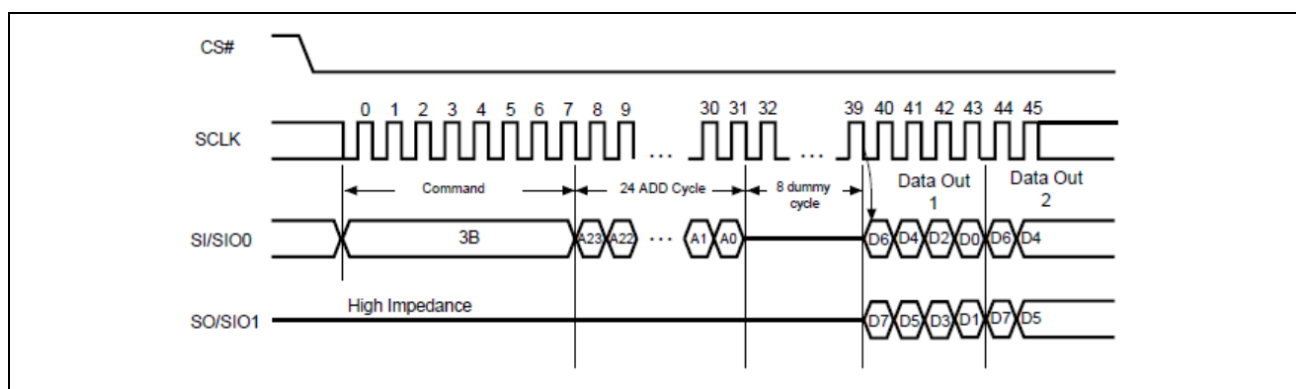


図10.8 DREAD命令シーケンス

(8) QREAD (Quad Read Mode) 命令

QREAD命令は、データをSI00端子からSI03端子で読むためのものです。読むアドレスはFAST_READ命令と同じです。本ドライバでは、読むアドレスはリードアドレス（変数）として管理しており、開始番号（start）でDN_RDADDR（-101）を指定した tk_wri_dev か tk_swri_dev で任意のアドレスを指定できます。また、開始番号（start）でDN_QREAD（3）を指定した tk_rea_dev か tk_srea_dev により、リードアドレス（変数）でQREAD命令を発行することができます。読んだデータ分、リードアドレス（変数）は上位アドレスに増加します。

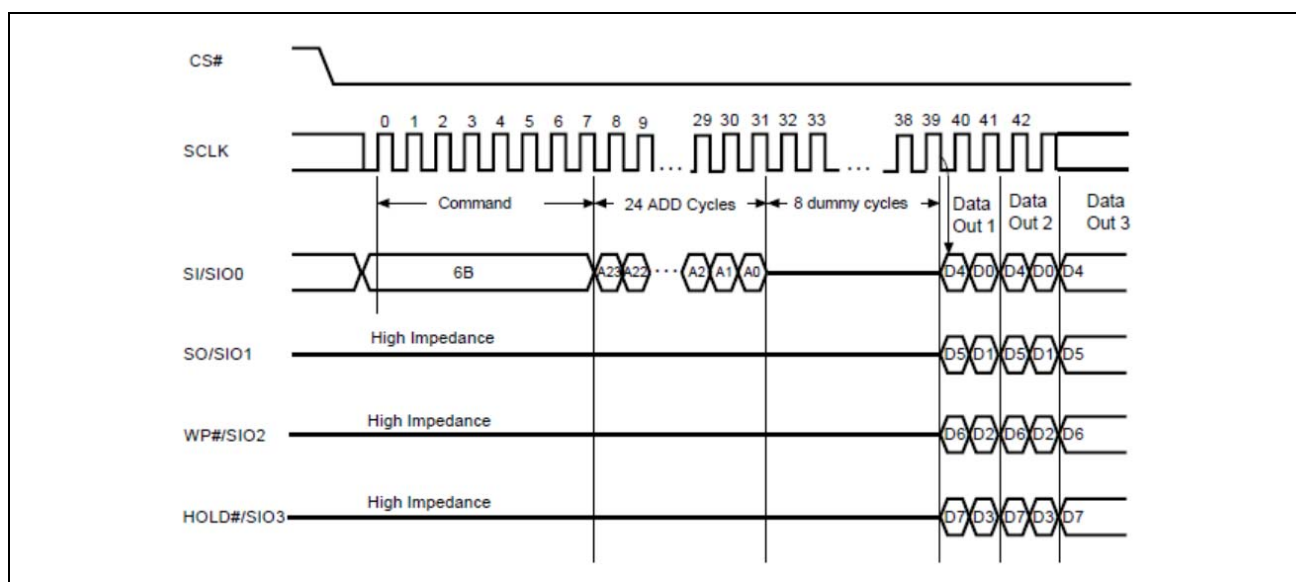


図10.9 QREAD命令シーケンス

(9) SE (Sector Erase) 命令

SE命令は、選択したセクタのデータを消去して“1”にします。この命令は任意の4K バイトのセクタに使用できます。本ドライバでは、開始番号（start）でDN_SE（-105）を指定した tk_wri_dev か tk_swri_dev でSE命令を発行することができます。その際、消去するセクタ内の任意のアドレスを格納したUW型の変数のアドレスをデータ（buf）に指定してください。

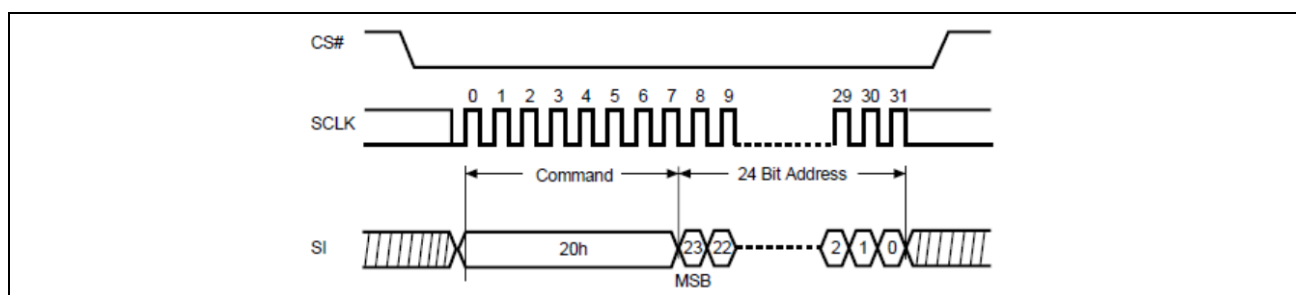


図10.10 SE命令シーケンス

(10) BE32K (Block Erase 32K) 命令

BE32K命令は、選択したブロックのデータを消去して“1”にします。この命令は任意の32K バイトのブロックに使用できます。本ドライバでは、開始番号（start）でDN_BE32K（-106）を指定した

tk_wri_dev か tk_swri_dev でBE32K命令を発行することができます。その際、消去するブロック内の任意のアドレスを格納したUW型の変数のアドレスをデータ（buf）に指定してください。

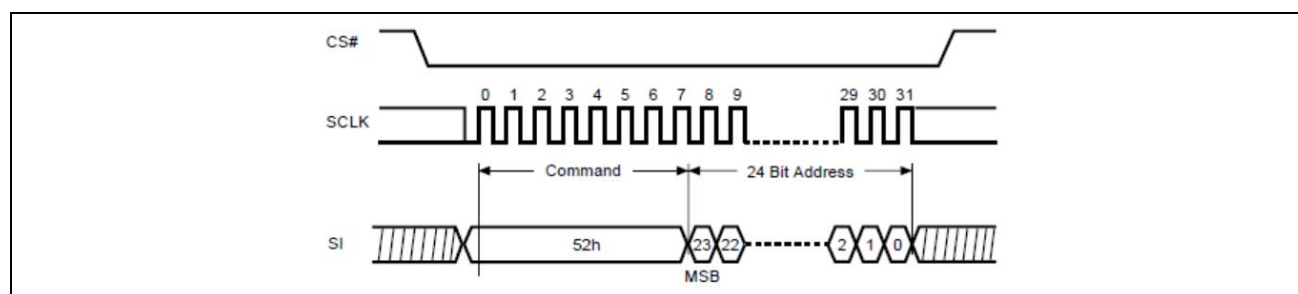


図10.11 BE32K命令シーケンス

(11) BE (Block Erase) 命令

BE命令は、選択したブロックのデータを消去して“1”にします。この命令は任意の64K バイトのブロックに使用できます。本ドライバでは、開始番号（start）でDN_BE（-107）を指定した tk_wri_dev か tk_swri_dev でBE命令を発行することができます。その際、消去するブロック内の任意のアドレスを格納したUW型の変数のアドレスをデータ（buf）に指定してください。

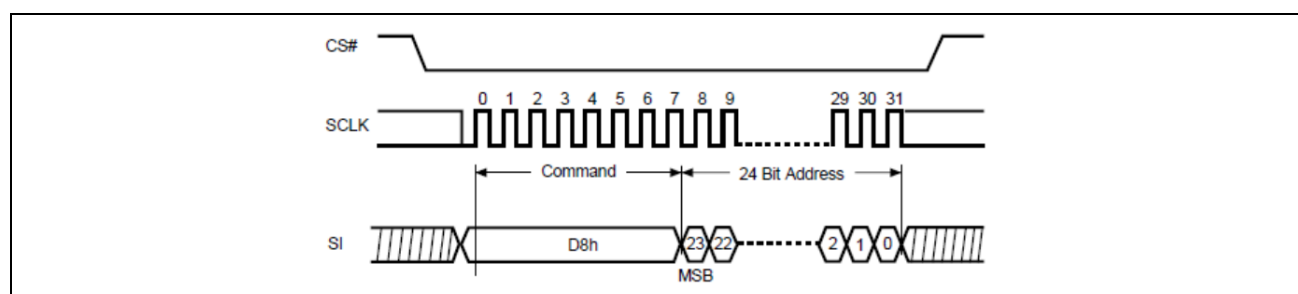


図10.12 BE命令シーケンス

(12) CE (Chip Erase) 命令

CE命令は、チップ全体のデータを消去して“1”にします。本ドライバでは、開始番号（start）でDN_CE（-108）を指定した tk_wri_dev か tk_swri_dev でCE命令を発行することができます。

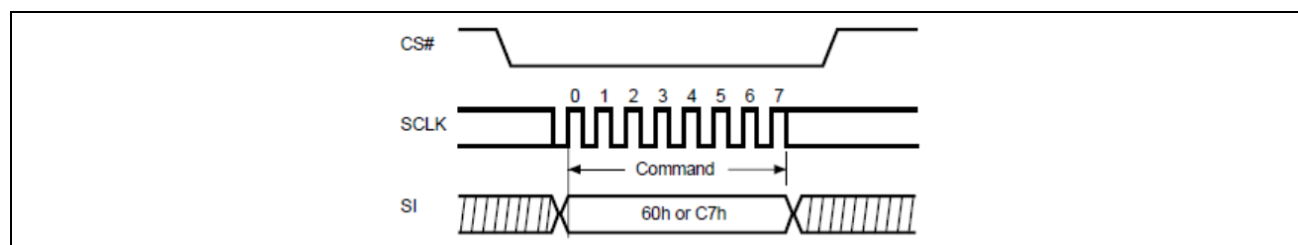


図10.13 CE命令シーケンス

(13) PP (Page Program) 命令

PP命令は、メモリビットを“0”にプログラムします。プログラムするデバイスには、1 ～ 256バイ

トを送信できます。なお、PP命令では256バイトのページの境界を越えることはできません。本ドライバでは、書込むアドレスはライトアドレス（変数）として管理しており、開始番号（start）でDN_WTADDR（-100）を指定した tk_wri_dev か tk_swri_dev で任意のアドレスを指定できます。また、開始番号（start）でDN_PP（0）を指定した tk_wri_dev か tk_swri_dev により、ライトアドレス（変数）でPP命令を発行することができます。書込んだデータ分、ライトアドレス（変数）は上位アドレスに増加します。ただし、256バイトのページ境界を超える書込みはできません。ページ境界を超える場合は E_PAR となります。

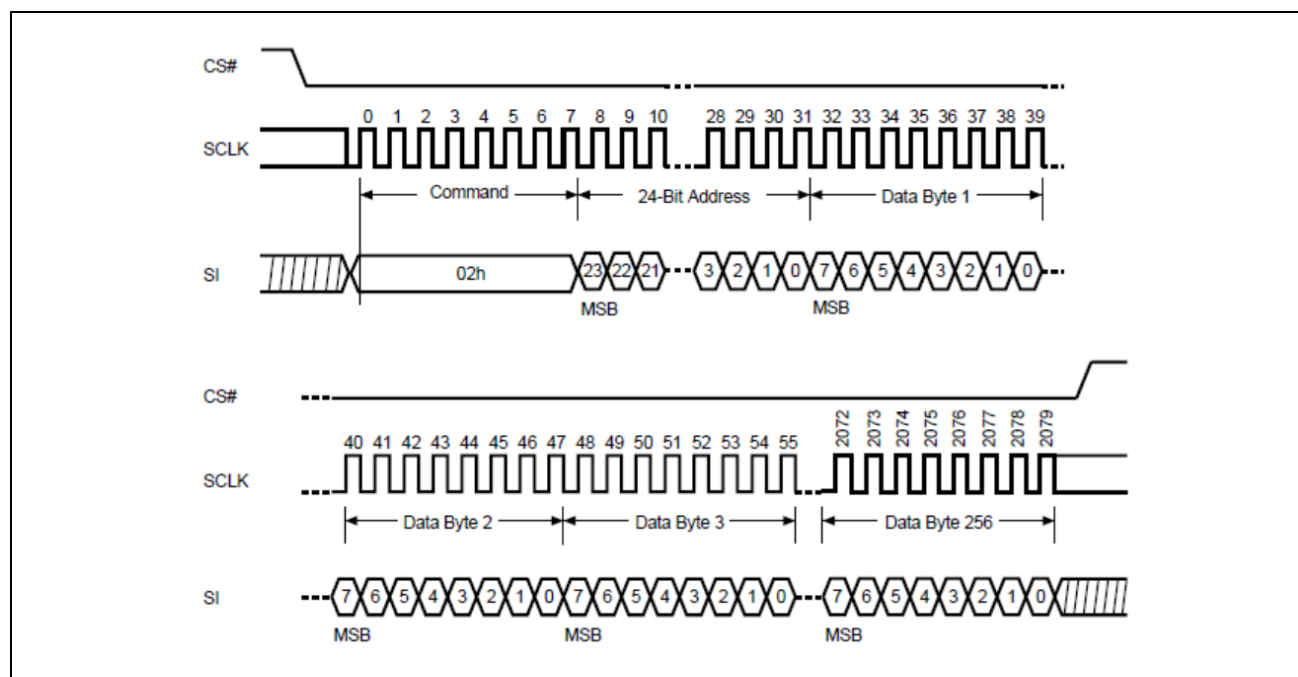


図10.14 PP命令シーケンス

10.3 シリアルフラッシュドライバ

10.3.1 対象デバイス

内蔵QSPIに接続されたシリアルフラッシュ（MX25L3233F）が対象です。

10.3.2 デバイス名

デバイス名は“QSPISF”です。

10.3.3 固有機能

なし

10.3.4 属性データ

以下の属性データをサポートします。

- シリアルフラッシュの書込みアドレスを設定する。

W

- 開始番号 (start) : DN_WTADDR (-100)
 サイズ (size) : sizeof(UW)
 データ (buf) : ライトアドレスを格納したUW型の変数のアドレス
- シリアルフラッシュの読み込みアドレスを設定する。 W
- 開始番号 (start) : DN_RDADDR (-101)
 サイズ (size) : sizeof(UW)
 データ (buf) : リードアドレスを格納したUW型の変数のアドレス
- Status Registerの値を読み込む。 R
- 開始番号 (start) : DN_RDSR (-102)
 サイズ (size) : sizeof(UB)
 データ (buf) : Status Registerの値を読み込むUB型の変数のアドレス
- Configuration Registerの値を読み込む。 R
- 開始番号 (start) : DN_RDCR (-103)
 サイズ (size) : sizeof(UB)
 データ (buf) : Configuration Registerの値を読み込むUB型の変数のアドレス
- Status RegisterとConfiguration Registerに値を書込む。 W
- 開始番号 (start) : DN_WRSR (-104)
 サイズ (size) : sizeof(UB) × 2
 データ (buf) : Status RegisterとConfiguration Registerの値を格納したUB型の2つの配列のアドレス
- 指定したアドレスが属するセクション (4K Byte) を消去する。 W
- 開始番号 (start) : DN_SE (-105)
 サイズ (size) : sizeof(UW)
 データ (buf) : 消去するセクションのアドレスを格納したUW型の変数のアドレス
- 指定したアドレスが属するブロック (32K Byte) を消去する。 W
- 開始番号 (start) : DN_BE32K (-106)
 サイズ (size) : sizeof(UW)
 データ (buf) : 消去するブロック (32K Byte) のアドレスを格納したUW型の変数のアドレス
- 指定したアドレスが属するブロック (64K Byte) を消去する。 W
- 開始番号 (start) : DN_BE (-107)
 サイズ (size) : sizeof(UW)
 データ (buf) : 消去するブロック (64K Byte) のアドレスを格納したUW型の変数のアドレス
- チップ全体 (4M Byte) を消去する。 W
- 開始番号 (start) : DN_CE (-108)
 サイズ (size) : 無効 (値をチェックしない)
 データ (buf) : 無効 (値をチェックしない)

10.3.5 固有データ

以下の固有データをサポートします。

■ ページプログラムをシリアルフラッシュに書き込みを行う。

W

開始番号 (start) : DN_PP (0)

サイズ (size) : 書込むデータのサイズ

データ (buf) : 書込むデータを格納したバッファのアドレス

書込むアドレスはDN_WTADDRで設定したライトアドレスから行われ、書込んだサイズ分、ライトアドレスは自動的に増加します。

■ Fastリード (S0端子のみ) でシリアルフラッシュの読み込みを行う。

R

開始番号 (start) : DN_FREAD (1)

サイズ (size) : 読み込むデータのサイズ

データ (buf) : 読み込むデータを格納するバッファのアドレス

読み込むアドレスはDN_RDADDRで設定したリードアドレスから行われ、読み込んだサイズ分、リードアドレスは自動的に増加します。

■ Dualリード (SI00端子とSI01端子) でシリアルフラッシュの読み込みを行う。

R

開始番号 (start) : DN_DREAD (2)

サイズ (size) : 読み込むデータのサイズ

データ (buf) : 読み込むデータを格納するバッファのアドレス

読み込むアドレスはDN_RDADDRで設定したリードアドレスから行われ、読み込んだサイズ分、リードアドレスは自動的に増加します。

■ Quadリード (SI00端子からSI03端子) でシリアルフラッシュの読み込みを行う。

R

開始番号 (start) : DN_QREAD (3)

サイズ (size) : 読み込むデータのサイズ

データ (buf) : 読み込むデータを格納するバッファのアドレス

読み込むアドレスはDN_RDADDRで設定したリードアドレスから行われ、読み込んだサイズ分、リードアドレスは自動的に増加します。

10.3.6 事象通知

未サポート

10.3.7 エラーコード

μT-Kernel3.0仕様書のデバイス管理機能の項を参照ください。QSPI固有の特殊なエラーコードは存在しません。

10.3.8 ヘッダファイルとサービス関数

ヘッダファイルは dev_qspi.h です。

シリアルフラッシュドライバを登録するためのサービス関数の仕様は以下の通りです。

ER ercd = qspiDrvEntry(void);

10.3.9 シリアルフラッシュドライバが使用する資源

(1) 使用する資源の概要

シリアルフラッシュドライバは処理を実施するに当たり、1つのタスクと1つのイベントフラグを利用します。サービス関数のAPIを呼び出すと生成されます。

(2) タスク (qspi_tsk)

QSPIの通信処理やユーザアプリケーションからのシステムコールの受け付けを行います。

項目	値
拡張情報	0
タスク属性	TA_HLNG TA_DSNAME TA_USERBUF
タスク起動アドレス	qspi_tsk
タスク起動時優先度	QSPI_CFG_TASK_PRIORITY
スタックサイズ	260
DSオブジェクト名称	"qspi_t"

備考：

TA_DSNAME、TA_USERBUFのタスク属性はカーネルのコンフィグレーションによって未サポートとなる場合があります。

(3) イベントフラグ

QSPIの通信処理やユーザアプリケーションからのシステムコールの受け付け状況の管理を行います。

項目	値
拡張情報	未使用
タスク属性	TA_PRI TA_WMUL TA_DSNAME
DSオブジェクト名称	"qspi_f"

備考：

TA_DSNAMEのイベントフラグ属性はカーネルのコンフィグレーションによって未サポートとなる場合があります。

10.3.10 シリアルフラッシュドライバのコンフィグレーション

(1) シリアルフラッシュドライバのコンフィグレーション・ファイル

シリアルフラッシュドライバのコンフィグレーション・ファイルは、mtkernel_3¥device¥qspi ¥sysdepend¥ターゲット¥qspi_config.h です。

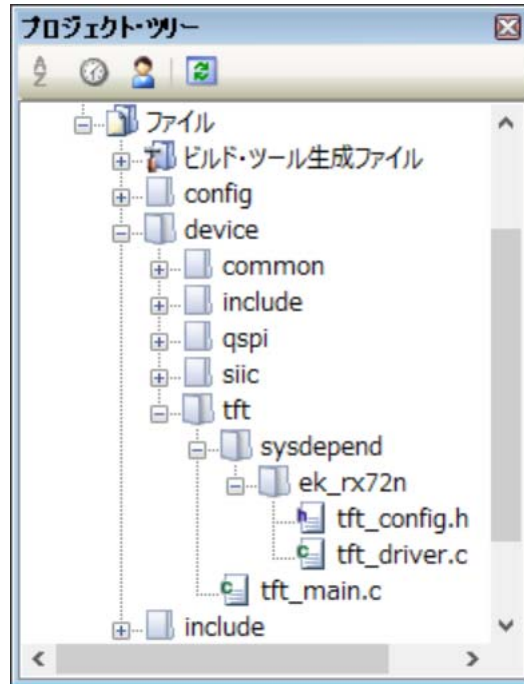


図10.15 シリアルフラッシュドライバのコンフィグレーション・ファイル

(2) タスクの優先度

● QSPI_CFG_TASK_PRIORITY

QSPIの通信処理を実行するタスク (qspi_tsk) のタスク優先度です。

システムの都合に応じて、1 ~ CFN_MAX_TSKPRI (タスク優先度の最大値) の範囲で変更できます。最低でもシリアルフラッシュを利用するタスクの優先度よりも高く設定することを推奨します。デフォルトは 5 に設定されています。

```
/* QSPI control task priority. */
#define QSPI_CFG_TASK_PRIORITY (5)
```

10.3.11 スタックサイズ

(1) スタック見積もりツール

本実装においてもスタック見積もりツールを使用することが可能です。スタック見積もりツールの起動方法やリアルタイムOSオプションの設定等はターゲット毎の構築仕様書の第5章を参照ください。

以降はスタック見積もりツールを使用することを前提に各スタックサイズを紹介します。以下にスタック見積もりツールの表示例を示します。

Symbol	Attributes	Address	Size	Stack size	Source
_qspi_wait	S	0xffc03819	31	12	qspi_main.obj
_qspi_exec	S	0xffc037ba	95	16	qspi_main.obj
_qspi_abort	S	0xffc03838	55	16	qspi_main.obj
_qspi_open	S	0xffc037b4	3	4	qspi_main.obj
_qspi_event	S	0xffc0386f	3	4	qspi_main.obj
_qspi_tsk		0xffc03872	189	44	qspi_main.obj
_qspi_close	S	0xffc037b7	3	4	qspi_main.obj

図10.16 シリアルフラッシュドライバのスタック見積もり結果

(2) qspi_tsk のスタックサイズ

qspi_tskタスクが独自に使用するスタックサイズは148バイトです。コンパイラのバージョン等が変化しても、それほど大きな違いはないはずです。このサイズにタスクコンテキストのサイズを加えても、**現状のサンプルで確保されている260バイトは超えない**と考えて構いません。

ただし、カーネル管理外割込みを複数レベル使用すると260バイトを超える可能性があります。もし、カーネル管理外割込みを複数レベル使用の場合はターゲット毎の構築仕様書の第5章の内容に従ってスタックサイズを算出し、mtkernel_3¥device¥qspi¥qspi_main.cで生成・確保されているスタックサイズを変更してください。

```
u.t_ctsk.stksz = 260;           // Set Task StackSize
u.t_ctsk.itspri = QSPI_CFG_TASK_PRIORITY; // Set Task Priority
```

(3) 処理関数のスタックサイズ

μT-Kernel/SMから呼び出されるデバイスドライバの処理関数 (qspi_open、qspi_close、qspi_exec、qspi_wait、qspi_abort、qspi_event) の中でシステムコールの加算条件を超えるものはありません。このため現状のスタック解析結果では処理関数が使用するサイズは表示されないようにしてあります。詳しくはEK-RX72N構築仕様書の第5章を参照してください。

10.3.12 サンプルプログラム (usermain_qspi.c)

usermain関数では、サービス関数を使ってシリアルフラッシュドライバを登録し、シリアルフラッシュに対してデータの書込み・読込みを行うシリアルフラッシュタスク (sf_tsk) を生成・起動し、自身はtk_slp_tskで待ち状態となります。

usermain関数

```
#include <string.h>
#include <tk/tkernel.h>
#include <tm/tmonitor.h>
```

```

#include "dev_qspi.h"

EXPORT INT usermain( void )
{
    T_CTSK t_ctsk;
    ID objid;

    qspiDrvEntry( );
    t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
    t_ctsk.stksz = 1024;
    t_ctsk.itskpri = 10;
    t_ctsk.task = display_tsk;
    strcpy( t_ctsk.dsname, "display" );
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK )
        goto ERROR;
    if( tk_sta_tsk( objid, 0 ) != E_OK )
        goto ERROR;
    while( 1 ) tk_slp_tsk(TMO_FEVR);
ERROR:
    return 0;
}

```

シリアルフラッシュタスク (sf_tsk) は、シリアルフラッシュに書込むデータをbuf配列の0行目 (256バイト) に作成したら、シリアルフラッシュドライバをオープンします。オープン後はライトアドレスとリードアドレスを共にフラッシュメモリの0番地に設定します。

そして、シリアルフラッシュの0番地からの256バイトをbuf配列の1行目 (256バイト) をFAST_READ命令で読み込み、コンソールに表示します。その後、buf配列の0行目をフラッシュメモリに書込みます。書込み後は、再びリードアドレスをフラッシュメモリの0番地に設定し、シリアルフラッシュの0番地からの256バイトをbuf配列の1行目にDREAD命令で読み込み、コンソールに表示し、正しく書込まれたことを確認します。

確認後は、セクションイレースで0番地を含むセクションをイレースし、リードアドレスをフラッシュメモリの0番地に設定後、シリアルフラッシュの0番地からの256バイトをbuf配列の1行目にQREAD命令で読み込み、正しくイレースされたことを確認します。

シリアルフラッシュタスク (sf_tsk)

```

UB buf[2][256];

EXPORT void sf_tsk(INT stacd, void *exinf)
{
    ID dd;
    SZ asize;
    UW addr = 0;
    INT i, j;
    for( i=0 ; i<256 ; i++ )
        buf[0][i] = i;
    if( ( dd = tk_opn_dev( QSPISF_DEVNM, TD_UPDATE ) ) <= E_OK )
        goto Err;
    tk_swri_dev( dd, DN_WTADDR, &addr, sizeof(UW), &asize );
    tk_swri_dev( dd, DN_RDADDR, &addr, sizeof(UW), &asize );
    tm_putstring( " Read Data\r\n\r" );
    tk_srea_dev( dd, DN_FREAD, buf[1], sizeof(buf[1]), &asize );
    for( i=0 ; i<16 ; i++ ) {

```

```

        for( j=0 ; j<16 ; j++ )
            tm_printf(" %02X", buf[1][i*16+j]);          // Output Dump Code
        tm_putstring("\n\r");                             // Output CR, LF
    }

    tm_putstring(" Write Data\n\r");
    tk_swri_dev( dd, DN_PP, buf[0], sizeof(buf[0]), &asize ); // Write Data
    tk_swri_dev( dd, DN_RDADDR, &addr, sizeof(UW), &asize ); // Set Read Address
    tm_putstring(" Read Data\n\r");
    tk_srea_dev( dd, DN_DREAD, buf[1], sizeof(buf[1]), &asize ); // Dual Read
    for( i=0 ; i<16 ; i++ ) {                               // Make Write Data
        for( j=0 ; j<16 ; j++ )
            tm_printf(" %02X", buf[1][i*16+j]);          // Output Dump Code
        tm_putstring("\n\r");                             // Output CR, LF
    }
    tm_putstring(" Section Erase\n\r");
    tk_swri_dev( dd, DN_SE, &addr, sizeof(UW), &asize ); // Section Erase
    tk_swri_dev( dd, DN_RDADDR, &addr, sizeof(UW), &asize ); // Set Read Address
    tm_putstring(" Read Data\n\r");
    tk_srea_dev( dd, DN_QREAD, buf[1], sizeof(buf[1]), &asize ); // Quad Read
    for( i=0 ; i<16 ; i++ ) {                               // Make Write Data
        for( j=0 ; j<16 ; j++ )
            tm_printf(" %02X", buf[1][i*16+j]);          // Output Dump Code
        tm_putstring("\n\r");                             // Output CR, LF
    }
}
Err:
    tk_ext_tsk();                                          // Exit
}

```

11. 問い合わせ先

本実装に関する問い合わせや他のRXファミリへのポーティングに関する相談は以下のメールアドレス宛にお願い致します。

yuji_katori@yahoo.co.jp

トロンフォーラム学術・教育WGメンバ
鹿取 祐二（かとり ゆうじ）

なお、上記のメールアドレスは余儀なく変更される場合がありますが、その際はご了承ください。

以上