

μ T-Kernel3.0 デバイスドライバインタフェース 仕様書

Version. 01. 00. 02

2024. 9. 12

目次

1. 概要	6
1.1 目的	6
1.2 実装の基本方針	6
1.3 関連ドキュメント	6
1.4 ソースコード構成	6
1.5 プロジェクト・ファイル	7
2. I2C ドライバ	8
2.1 対象デバイス	8
2.2 デバイス名	8
2.3 固有機能	8
2.4 属性データ	8
2.5 固有データ	9
2.6 事象通知	9
2.7 エラーコード	9
2.8 ヘッダファイルとサービス関数	9
2.9 I2C ドライバが使用する資源	9
2.9.1 使用する資源の概要	9
2.9.2 タスク (iic_tsk)	9
2.9.3 イベントフラグ	10
2.9.4 割込みハンドラ (riic*_eei*_hdr、riic*_rxi*_hdr、riic*_txi*_hdr、riic*_tei*_hdr)	10
2.9.5 グループ割込み (GroupBL1Handler)	12
2.10 I2C ドライバのコンフィグレーション	12
2.10.1 I2C ドライバのコンフィグレーション・ファイル	12
2.10.2 タスクの優先度と割込みハンドラの割込み優先度	13
2.10.3 RIIC チャネル関連	13
2.10.4 端子関連	14
2.11 スタックサイズ	14
2.11.1 スタック見積もりツール	14
2.11.2 iic_tsk のスタックサイズ	15
2.11.3 割込みハンドラのスタックサイズ	15
2.11.4 グループ割込み	15
2.11.5 処理関数のスタックサイズ	17
2.12 サンプルプログラム	17
2.12.1 サンプルプログラムの概要	17
2.12.1 光センサとジェスチャーセンサのサンプルプログラム	17

2.12.2	EEPROM のサンプルプログラム	20
3.	簡易 I2C ドライバ	23
3.1	対象デバイス	23
3.2	デバイス名	23
3.3	固有機能	24
3.4	属性データ	24
3.5	固有データ	24
3.6	事象通知	24
3.7	エラーコード	25
3.8	ヘッダファイルとサービス関数	25
3.9	簡易 I2C ドライバが使用する資源	25
3.9.1	使用する資源の概要	25
3.9.2	タスク (siic_tsk)	25
3.9.3	イベントフラグ	25
3.9.4	割込みハンドラ (sci*_rx*_hdr、sci*_tx*_hdr、sci*_tei*_hdr)	26
3.9.5	グループ割込み (GroupBL0Handler、GroupBL1Handler、GroupAL0Handler)	29
3.10	簡易 I2C ドライバのコンフィグレーション	29
3.10.1	簡易 I2C ドライバのコンフィグレーション・ファイル	29
3.10.2	タスクの優先度と割込みハンドラの割込み優先度	30
3.10.3	SCI チャンネル関連	30
3.11	スタックサイズ	31
3.11.1	スタック見積もりツール	31
3.11.2	siic_tsk のスタックサイズ	32
3.11.3	割込みハンドラのスタックサイズ	32
3.11.4	グループ割込み	32
3.11.5	処理関数のスタックサイズ	33
3.12	サンプルプログラム	34
4.	シリアルドライバ	35
4.1	対象デバイス	35
4.2	デバイス名	35
4.3	固有機能	36
4.4	属性データ	36
4.5	固有データ	37
4.6	事象通知	37
4.7	エラーコード	37
4.8	ヘッダファイルとサービス関数	37
4.9	シリアルドライバが使用する資源	37
4.9.1	使用する資源の概要	37

4.9.2	タスク (sci_tsk)	37
4.9.3	イベントフラグ	38
4.9.4	割込みハンドラ (sci*_rx*_hdr、sci*_tx*_hdr、sci*_tei*_hdr、sci*_eri*_hdr)	38
4.9.5	グループ割込み (GroupBL0Handler、GroupBL1Handler、GroupAL0Handler)	42
4.10	シリアルドライバのコンフィグレーション	42
4.10.1	シリアルドライバのコンフィグレーション・ファイル	42
4.10.2	タスクの優先度と割込みハンドラの割込み優先度	43
4.10.3	SCI チャンネル関連	44
4.11	スタックサイズ	44
4.11.1	スタック見積もりツール	44
4.11.2	sci_tsk のスタックサイズ	45
4.11.3	割込みハンドラのスタックサイズ	45
4.11.4	グループ割込み	45
4.11.5	処理関数のスタックサイズ	46
4.12	サンプルプログラム	46
5.	データフラッシュドライバ	48
5.1	対象デバイス	48
5.2	デバイス名	48
5.3	固有機能	48
5.4	属性データ	48
5.5	固有データ	48
5.6	事象通知	48
5.7	エラーコード	49
5.8	ヘッダファイルとサービス関数	49
5.9	データフラッシュドライバが使用する資源	49
5.9.1	データフラッシュドライバが使用する資源の概要	49
5.9.2	割込みハンドラ (fcu_frdyi_hdr)	49
5.10	データフラッシュドライバのコンフィグレーション	50
5.10.1	データフラッシュドライバのコンフィグレーション・ファイル	50
5.10.2	割込みハンドラの割込み優先度	50
5.11	スタックサイズ	50
5.11.1	スタック見積もりツール	50
5.11.2	割込みハンドラのスタックサイズ	51
5.11.3	処理関数のスタックサイズ	51
5.12	サンプルプログラム	51
6.	本実装に関する考察	55
6.1	排他制御	55
6.2	データフラッシュドライバ	56

7. 問い合わせ先	56
-----------------	----

1. 概要

1.1 目的

本仕様書はルネサスエレクトロニクス社のRX用に改変した μ T-Kernel 3.0 (V3.00.00) のソースコードに搭載する各種デバイスドライバのインタフェース仕様を記載したものです。

1.2 実装の基本方針

各種デバイスドライバの実装の基本方針を以下に示します。

- μ T-Kernel 3.0仕様のデバイス管理機能に準拠した構造とする。
- 処理用タスクと割込みハンドラを利用した汎用デバイスドライバとして実装する。一部、処理用タスクが存在しない単純デバイスドライバとして実装する。
- デバイスドライバのインタフェースライブラリは使用しないで実装する。理由はデバイスドライバが使用する資源を極力削減するためである。基本的にはドライバ毎にイベントフラグを1つだけ使用した構造とする。

1.3 関連ドキュメント

以下に本構築仕様書の関連するドキュメントを示します。

分類	名称	発行
OS仕様	μ T-Kernel 3.0仕様書 (Ver. 3.00.00)	TRONフォーラム TEF020-S004-3.00.00
T-Monitor	T-Monitor仕様書	TRONフォーラム TEF-020-S002-01.00.01
共通実装仕様	uTK3.0共通実装仕様書	

1.4 ソースコード構成

本実装のソースコードのディレクトリ構成を以下に示します。

└ device	デバイスドライバ・ミドルウェア
└ common	デバイスドライバ共通ライブラリ
└ include	デバイスドライバ用のインクルードファイル
└ *****	各種デバイスドライバ用ディレクトリ
└ sysdepend	実装依存定義
└ include	アプリケーション用のインクルードファイル

機種依存定義 sysdepend ディレクトリは以下のように構成されます。

└ sysdepend	実装依存定義
-------------	--------

└ <ターゲット1>	ターゲット1 依存部
└ :	
└ <ターゲットn>	ターゲットn 依存部
└ <core>	コア共通実装定義
└└ <コア1>	コア1 依存部
└└ :	
└└ <コアn>	コアn 依存部

各デバイスドライバのソースコードを他のターゲットに移植する場合は、機種依存定義 sysdepend ディレクトリにある core ディレクトリ内のソースコードをご利用ください。

1.5 プロジェクト・ファイル

本実装を利用するためのプロジェクト・ファイルは「mtkernel_3/ide/cs」のフォルダにあります。

アイコンの名称が ****_DRV.mtpj となっているものがデバイスドライバを利用可能な μ T-Kernel3.0 のプロジェクト・ファイルです。ダブルクリックすればCS+が起動されます。

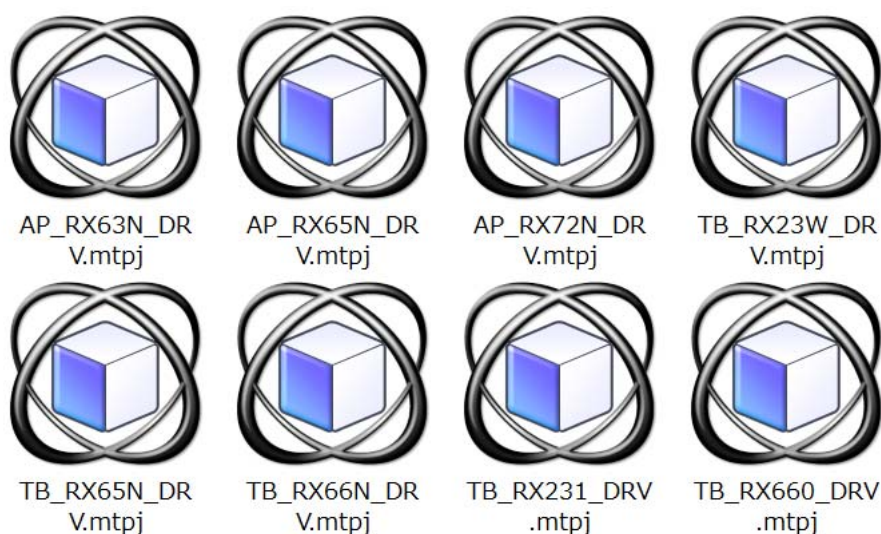


図1.1 デバイスドライバのプロジェクト・ファイル

2. I2C ドライバ

2.1 対象デバイス

スレーブアドレスが7ビットのI2Cデバイスが対象となります。

2.2 デバイス名

デバイス名は、“iica”、“iicb”、“iicc”、“iicd”を使用します。内蔵周辺機能のI2Cバスインタフェース（RIIC）のチャンネル番号とデバイス名の関係は一致しています。ターゲット毎とコア毎に利用可能なデバイス名は以下の通りです。

表2.1 ターゲット毎に利用可能なデバイス名

	iica (RIIC0)	iicb (RIIC1)	iicc (RIIC2)	iicd (RIIC3)
AP-RX63N-0A	○			
AP-RX65N-0A	○			
AP-RX72N-0A		○		
TB-RX231	○			
TB-RX23W	○			
TB-RX65N	○	○	○	
TB-RX660	○		○	
TB-RX66N	○	○	○	

表2.2 コア毎に利用可能なデバイス名

	iica (RIIC0)	iicb (RIIC1)	iicc (RIIC2)	iicd (RIIC3)
R5F563N (RX63N)	○	○	○	○
R5F5231 (RX231)	○			
R5F523W (RX23W)	○			
R5F565N (RX65N)	○	○	○	
R5F5660 (RX660)	○		○	
R5F566N (RX66N)	○	○	○	
R5F572N (RX72N)	○	○	○	

2.3 固有機能

スレーブアドレスが7ビットのI2Cデバイスとの送受信を行います。

2.4 属性データ

なし

2.5 固有データ

start : I2Cデバイスのスレーブアドレスを指定します。
下位7ビットのみが有効であり、他の上位ビットは無視されます。

buf : 送受信するデータの格納先アドレスを指定します。
送信時に size=0 であれば無視されます。

size : 送受信データのサイズを指定します。
送信（書き込み）時は 0 指定が可能です（Acknowledge Polling等にご利用します）。
受信（読み込み）時の 0 指定は E_PAR となります。
負の値は送受信（読み込み、書き込み）共に E_PAR となります。

2.6 事象通知

なし

2.7 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。I2Cドライバ固有の特殊なエラーコードは存在しません。

2.8 ヘッダファイルとサービス関数

ヘッダファイルは dev_iic.h です。

I2C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 = iicDrvEntry(void);
```

2.9 I2C ドライバが使用する資源

2.9.1 使用する資源の概要

I2Cドライバは処理を実施するに当たり、デバイス名毎に1つのタスク、1つのイベントフラグ、4つの割込みハンドラを利用します。サービス関数のAPIを呼び出すと生成または定義されます。

使用する資源	名称	優先度
タスク	iic_tsk	タスクの優先度は IIC_CFG_TASK_PRIORITY
割込みハンドラ	riic*_eei*_hdr	割込みハンドラの割込み優先度は IIC_CFG_INT_PRIORITY
	riic*_rxi*_hdr	割込みハンドラの割込み優先度は IIC_CFG_INT_PRIORITY
	riic*_txi*_hdr	割込みハンドラの割込み優先度は IIC_CFG_INT_PRIORITY
	riic*_tei*_hdr	割込みハンドラの割込み優先度は IIC_CFG_INT_PRIORITY

注：IIC_CFG_TASK_PRIORITY と IIC_CFG_INT_PRIORITY は次項を参照

* の部分にはRIICのチャンネル番号が入る（0～3）。

2.9.2 タスク（iic_tsk）

I2C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	iic_tsk
タスク起動時優先度	IIC_CFG_TASK_PRIORITY
スタックサイズ	300
DSオブジェクト名称	"iic*_t" (*はデバイス名の最後の文字)

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。

2.9.3 イベントフラグ

I2C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	"iic*_f" (*はデバイス名の最後の文字)

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。

2.9.4 割り込みハンドラ (riic*_eei*_hdr、riic*_rxi*_hdr、riic*_txi*_hdr、riic*_tei*_hdr)

RIICの割り込みハンドラです。割り込みの発生によりイベントフラグへのイベントの設定を行います。

項目 (通信エラー/通信イベント発生)	コア	値			
		iica	iicb	iicc	iicd
割り込み番号、 グループ割り込み番号	R5F563N (RX63N)	182	186	190	194
	R5F5231 (RX231)	246	—	—	—
	R5F523W (RX23W)	246	—	—	—
	R5F565N (RX65N)	111、14	111、29	111、16	—
	R5F5660 (RX660)	111、14	—	111、16	—
	R5F566N (RX66N)	111、14	111、29	111、16	—
	R5F572N (RX72N)	111、14	111、29	111、16	—
割り込み優先度	IIC_CFG_INT_PRIORITY				
割り込みハンドラ起動アドレス	riic*_eei*_hdr (*はRIICのチャネル番号)				

項目 (受信データフル)	コア	値			
		iica	iicb	iicc	iicd
割込み番号	R5F563N (RX63N)	183	187	191	195
	R5F5231 (RX231)	247	—	—	—
	R5F523W (RX23W)	247	—	—	—
	R5F565N (RX65N)	52	50	54	—
	R5F5660 (RX660)	52	—	54	—
	R5F566N (RX66N)	52	50	54	—
	R5F572N (RX72N)	52	50	54	—
割込み優先度	IIC_CFG_INT_PRIORITY				
割込みハンドラ起動アドレス	riic*_rxi*_hdr (*はRIICのチャネル番号)				

項目 (送信データエンpty)	コア	値			
		iica	iicb	iicc	iicd
割込み番号	R5F563N (RX63N)	184	188	192	196
	R5F5231 (RX231)	248	—	—	—
	R5F523W (RX23W)	248	—	—	—
	R5F565N (RX65N)	53	51	55	—
	R5F5660 (RX660)	53	—	55	—
	R5F566N (RX66N)	53	51	55	—
	R5F572N (RX72N)	53	51	55	—
割込み優先度	IIC_CFG_INT_PRIORITY				
割込みハンドラ起動アドレス	riic*_txi*_hdr (*はRIICのチャネル番号)				

項目 (送信終了)	コア	値			
		iica	iicb	iicc	iicd
割込み番号、 グループ割込み番号	R5F563N (RX63N)	185	189	193	197
	R5F5231 (RX231)	249	—	—	—
	R5F523W (RX23W)	249	—	—	—
	R5F565N (RX65N)	111、13	111、28	111、15	—
	R5F5660 (RX660)	111、13	—	111、15	—
	R5F566N (RX66N)	111、13	111、28	111、15	—
	R5F572N (RX72N)	111、13	111、28	111、15	—
割込み優先度	IIC_CFG_INT_PRIORITY				
割込みハンドラ起動アドレス	riic*_tei*_hdr (*はRIICのチャネル番号)				

2.9.5 グループ割込み (GroupBL1Handler)

RIICの送信完了 (TEI*)、通信エラー (EE1*) の割込みがグループ割込みの場合 (RX65NやRX72N等の場合)、割込み番号 (ベクタ番号) 111の割込みハンドラを定義することになります。ただし、システムで割込み番号111のグループ割込みに分類された割込みを利用する場合、RIICの割込みと共存する必要があります。

この割込みに対してはデバイスドライバ共通のグループ割込みハンドラ (GroupBL1Handler) を利用しています。ソースファイルはgroupbl1.cです。グループ割込みハンドラでは関数ポインタ経由で割込み要求に対応した割込みハンドラを呼び出します。

GroupBL1Handlerグループ割込みハンドラ

```
EXPORT void (*GroupBL1Table[32])(UINT dintno);
EXPORT void GroupBL1Handler(UINT dintno)
{
    INT i;
    UW intreqflg;
    intreqflg = ICU.GRPBL1.LONG;
    for( i=0 ; !( intreqflg & 0xFF ) ; i+=8 )
        intreqflg >>= 8;
    for( ; intreqflg ; i++, intreqflg >>= 1 )
        if( intreqflg & 1 )
            GroupBL1Table[i]( dintno );
}
```

GroupBL1Handlerグループ割込みハンドラの中でGRPBL1レジスタの各ステータスフラグをチェックし、各割込みハンドラを呼び出します。

2.10 I2C ドライバのコンフィグレーション

2.10.1 I2C ドライバのコンフィグレーション・ファイル

I2Cドライバのコンフィグレーション・ファイルは、mtkernel_3¥device¥iic¥sysdepend¥ターゲット¥iic_config.h です。

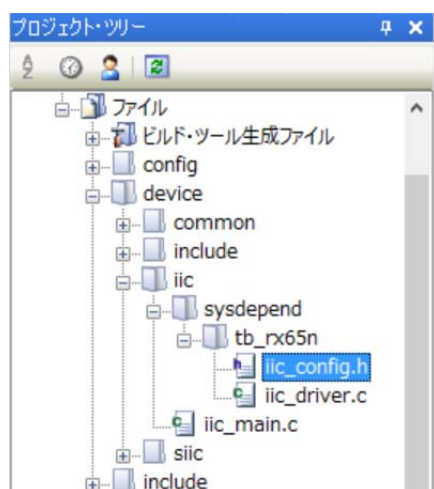


図2.1 I2Cドライバのコンフィグレーション・ファイル

2.10.2 タスクの優先度と割込みハンドラの割込み優先度

● IIC_CFG_TASK_PRIORITY

I2Cの通信処理を実行するタスク (iic_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I2C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 4 に設定されています。

● IIC_CFG_INT_PRIORITY

内蔵周辺機能のI2Cバスインタフェース (RIIC) の割込み優先度です。

システムの都合に応じて、1 ~ MAX_INT_PRI (最高外部割込み優先度) の範囲で変更できます。TIM_INT_PRI (システムタイマの割込み優先度) よりも低く設定することを推奨します。デフォルトでは 9 に設定されています。

なお、**グループ割込みを利用する場合、割込み番号が同一である他の割込みハンドラと同一の割込み優先度を指定する必要があります。**もし、異なる割込み優先度を指定している場合、iicDrvEntryのサービス関数の戻り値がエラーとなり、I2Cデバイスドライバの登録が失敗することがあります。

```
/* IIC control task priority. */  
#define IIC_CFG_TASK_PRIORITY          (4)  
  
/* IIC interrupt priority level. */  
#define IIC_CFG_INT_PRIORITY          (9)
```

2.10.3 RIIC チャンネル関連

● USE_IIC*

内蔵RIICの当該チャンネルの使用/未使用を設定します (定義=使用、未定義=未使用)。

使用する場合は目的のマクロ名を定義してください。定義すると2.9節で紹介した資源が生成、または定義されます。未使用の場合は目的のマクロ名をコメントアウトしてください。

● IIC*_HPSCL (USE_IIC*が未定義の場合は設定不要です)

SCLクロックのHigh期間を指定します。設定値の単位は μ sec です。

● IIC*_LPSCL (USE_IIC*が未定義の場合は設定不要です)

SCLクロックのLow期間を指定します。設定値の単位は μ sec です。

上記 2 つの設定値は接続するI2Cデバイスのデータシート等に記載されています。次頁にBH1750FVIの光センサとPAJ7620U2のジェスチャーセンサの例を示します。なお、設定値の最後の F は接尾子です。**必ず設定値の最後には接尾子の F を記載してください。**

BH1750FVIの光センサ

Parameter	Symbol	Limits			Units	Conditions
		Min.	Typ.	Max.		
I2C 'L' Period of the SCL Clock	tLOW	1.3	—	—	μs	
I2C 'H' Period of the SCL Clock	tHIGH	0.6	—	—	μs	

PAJ7620U2のジェスチャーセンサ

Parameter	Symbol	STANDARD MODE		FAST MODE		Unit
		Min.	Max.	Min.	Max.	
Low period of SCL clock.	tLOW	4.7		1.3		μs
High period of SCL clock.	tHIGH	4		0.6		μs

```
#define USE_IIC0          /* Use RIIC0 */
#define IIC0_HPSCl 0.6F /* RIIC0 I2C High Period of the SCL Clock(us) */
#define IIC0_LPSCl 1.3F /* RIIC0 I2C Low Period of the SCL Clock(us) */
```

2.10.4 端子関連

- USE_SCL_P**
- USE_SDA_P**

一部のターゲット、コア、RIICチャンネルにだけ存在する設定です。SCL端子やSDA端子が複数利用できる場合の選択です。使用する端子のマクロ名を定義し、使用しない端子のマクロ名はコメントアウトしてください。なお、全てをコメントアウトしたり、複数のマクロ名を定義すると正しい動作は得られないので注意してください。

```
#define USE_SCL_P16 /* Use RIIC0 SCL is P16 */
//#define USE_SCL_P12 /* Use RIIC0 SCL is P12 */
#define USE_SDA_P17 /* Use RIIC0 SDA is P17 */
//#define USE_SDA_P13 /* Use RIIC0 SDA is P13 */
```

2.11 スタックサイズ

2.11.1 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
_iic_wait (116)	S	0xffff82ec1	35	12	iic_main.obj
_riic0_txi0_hdr (96)	S	0xffff833eb	6	4	iic_driver.obj
_iic_close (4)	S	0xffff82e45	3	4	iic_main.obj
_riic0_tei0_hdr (96)	S	0xffff833f1	13	4	iic_driver.obj
_iic_open (4)	S	0xffff82e42	3	4	iic_main.obj
_riic0_rxi0_hdr (96)	S	0xffff833e5	6	4	iic_driver.obj
_iic_abort (104)	S	0xffff82ee4	60	16	iic_main.obj
_riic0_eei0_hdr (96)	S	0xffff833d0	21	4	iic_driver.obj
_iic_tsk (192)		0xffff82f23	194	44	iic_main.obj
_iic_exec (112)	S	0xffff82e48	121	24	iic_main.obj
_iic_event (4)	S	0xffff82f20	3	4	iic_main.obj

図2.2 I2Cデバイスドライバのスタック見積もり結果

2.11.2 iic_tsk のスタックサイズ

iic_tskタスクが独自に使用するスタックサイズは192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300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300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する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¥iic¥sysdepend¥ターゲット¥iic_driver.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。

```
p_ctsk->stksz = 300; // Set Task StackSize
p_ctsk->itskpri = IIC_CFG_TASK_PRIORITY; // Set Task Priority
```

2.11.3 割込みハンドラのスタックサイズ

図2.2の例では、riic0_eei0_hdr、riic0_rxi0_hdr、riic0_txi0_hdr、riic0_tei0_hdrの4つが割込みハンドラです（RIICチャネル0の4つの割込み）。図2.2の例では全て96バイトですが、異なる場合は、最大のサイズがI2Cデバイスドライバのコンフィグレーションで指定した IIC_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズとなります。この数値を使って例外スタックのサイズを算出してください。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

2.11.4 グループ割込み

RX65NやRX72N等のように通信エラー/通信イベント発生割込みハンドラ（riic*_eei*_hdr）と送信終了の割込みハンドラ（riic*_tei*_hdr）がGroupBL1Handlerグループ割込みハンドラ経由で実行される場合、スタック見積もりツールの表示結果は図2.3のようになります。

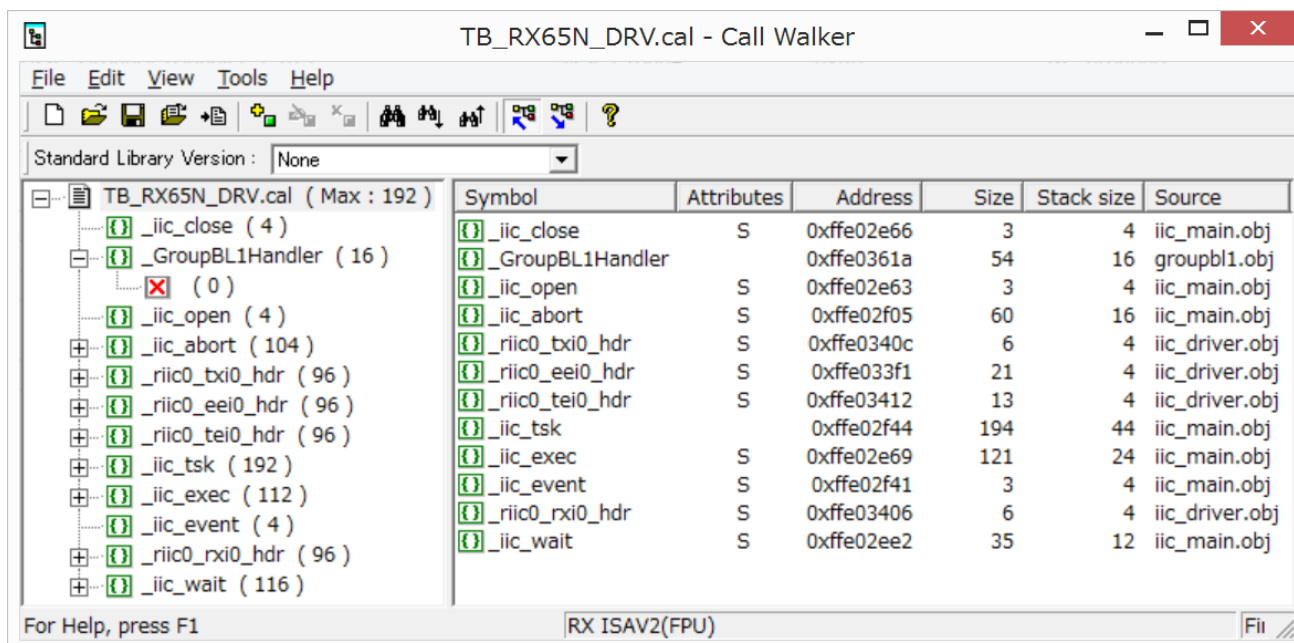


図2.3 GroupBL1Handlerグループ割込みハンドラを使用する場合のスタック見積もり結果

上記の場合、riic*_eei*_hdr割込みハンドラとriic*_tei*_hdr割込みハンドラはGroupBL1Handlerグループ割込みハンドラから関数ポインタ経由で実行されます。結果、割込みハンドラのみをドラッグ&ドロップで未解決シンボルを修正すると図2.4のようになります。

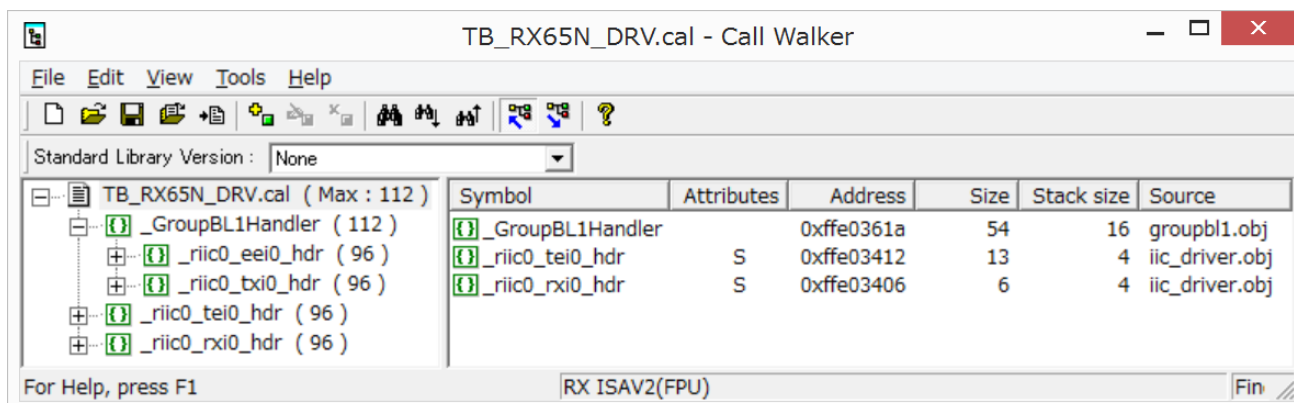


図2.4 ドラッグ&ドロップで未解決シンボルを修正した結果

結果、GroupBL1Handlerグループ割込みハンドラを利用する場合、riic*_eei*_hdr割込みハンドラとriic*_tei*_hdr割込みハンドラはGroupBL1Handlerグループ割込みハンドラが使用するサイズを加算したサイズで算出することになります。

図2.3の例であれば、IIC_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズは112バイトで計算することになります。

2.11.5 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（iic_open、iic_close、iic_exec、iic_wait、iic_abort、iic_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。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

2.12 サンプルプログラム

2.12.1 サンプルプログラムの概要

本実装では全てのターゲットで動作可能なBH1750FV1の光センサとPAJ7620U2のジェスチャーセンサを対象としたサンプルプログラムと、AP-RX63N-0A、AP-RX65N-0A、AP-RX72N-0Aの3つのターゲットでのみ動作する24LC01BTや24LC16BTのEEPROMを対象としたサンプルプログラムを準備しています。

【光センサとジェスチャーセンサ】

```
mtkernel_3¥kernel¥usermain_device_driver¥ usermain_i2c.c
```

【EEPROM】

```
mtkernel_3¥kernel¥usermain_device_driver¥ usermain_i2c_eeprom.c
```

両方に対応しているターゲットの場合、プロジェクト立ち上げ時は光センサとジェスチャーセンサのソースプログラムがアクティブになっており、EEPROMのサンプルプログラムはビルドから除外されています。必要に応じてビルドの対象を変更してお使いください。

なお、**デバイス名は使用するIICのチャンネルに合わせて変更する必要があります**（I2Cドライバや第3章で紹介する簡易I2Cドライバのデバイス名が利用可能です）。**2.10節に記載の内容に従ってI2Cドライバのコンフィグレーションを修正ください。**

```
if( ( dd = tk_opn_dev( "iica", TD_UPDATE ) ) <= E_OK )           // Open IIC or SIIC Driver
```

また、サンプルプログラムでは「CLANGSPEC」のマクロ定義を利用しています。「CLANGSPEC」のマクロ定義に関しては各ターゲットの構築仕様書を参照してください。

2.12.1 光センサとジェスチャーセンサのサンプルプログラム

usermain関数

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

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

    iicDrvEntry( );           // Entry IIC Driver
    siicDrvEntry( );         // Entry SIIC Driver
```

```

t_ctsk.tskatr = TA_HLNG | TA_DSNAME;           // Set Task Attribute
t_ctsk.stksz = 1024;                           // Set Task Stack Size
t_ctsk.itspri = 10;                            // Set Task Priority
t_ctsk.task = gesture_tsk;                     // Set Task Start Address
strcpy( t_ctsk.dsname, "gesture" );            // Set Debugger Support Name
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK )    // Create Gesture Task
    goto Err;
if( tk_sta_tsk( objid, 0 ) < E_OK )              // Start Gesture Task
    goto Err;
t_ctsk.task = light_tsk;                       // Set Task Start Address
strcpy( t_ctsk.dsname, "light" );              // Set Debugger Support Name
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK )    // Create Light Task
    goto Err;
if( tk_sta_tsk( objid, 0 ) < E_OK )              // Start Light Task
    goto Err;
tk_slp_tsk( TMO_FEVR );                        // Sleep
Err:
    while( 1 ) ;
}

```

usermain関数では、サービス関数を使ってI2Cドライバと簡易I2Cドライバを登録し、ジェスチャーセンサを操作するジャスチャータスク（gesture_tsk）と光センサを操作するライトタスク（light_tsk）を生成・起動し、自身はtk_slp_tskで待ち状態となります。

なお、I2Cドライバと簡易I2Cドライバで共通の端子を使用することはできません。不要であれば、簡易I2Cドライバのサービス関数（siicDrvEntry関数）の呼び出しは削除やコメントアウトしても構いません。

ジャスチャータスク（gesture_tsk）

```

EXPORT void gesture_tsk(INT stacd, void *exinf)
{
    ID dd;
    UB buf;
    UH data;
    SZ asize;
    INT i;
    LOCAL const UB IniReg[][2] = {               // Initialize Data
        {0xEF, 0x00}, {0x37, 0x07}, {0x38, 0x17}, {0x39, 0x06}, {0x42, 0x01},
        {0x46, 0x2D}, {0x47, 0x0F}, {0x48, 0x3C}, {0x49, 0x00}, {0x4A, 0x1E},
        {0x4C, 0x20}, {0x51, 0x10}, {0x5E, 0x10}, {0x60, 0x27}, {0x80, 0x42},
        {0x81, 0x44}, {0x82, 0x04}, {0x8B, 0x01}, {0x90, 0x06}, {0x95, 0x0A},
        {0x96, 0x0C}, {0x97, 0x05}, {0x9A, 0x14}, {0x9C, 0x3F}, {0xA5, 0x19},
        {0xCC, 0x19}, {0xCD, 0x0B}, {0xCE, 0x13}, {0xCF, 0x64}, {0xD0, 0x21},
        {0xEF, 0x01}, {0x02, 0x0F}, {0x03, 0x10}, {0x04, 0x02}, {0x25, 0x01},
        {0x27, 0x39}, {0x28, 0x7F}, {0x29, 0x08}, {0x3E, 0xFF}, {0x5E, 0x3D},
        {0x65, 0x96}, {0x67, 0x97}, {0x69, 0xCD}, {0x6A, 0x01}, {0x6D, 0x2C},
        {0x6E, 0x01}, {0x72, 0x01}, {0x73, 0x35}, {0x77, 0x01}, {0xEF, 0x00}, };
    LOCAL const VB *msg[] = { "down", "up", "right", "left", "near", "far", "R turn", "L turn", "ByeBye" };
    if( ( dd = tk_opn_dev( "iica", TD_UPDATE ) ) <= E_OK ) // Open IIC or SIIC Driver
        goto Err2;
    tk_dly_tsk( 1 );
    tk_swri_dev( dd, 0x73, "%x00", 1, &asize );           // Gesture Sensor Wakeup
    for( i=0 ; i<20 ; i++ ) {
        tk_swri_dev( dd, 0x73, "%x00", 1, &asize );       // Gesture Sensor Wakeup
        tk_srea_dev( dd, 0x73, &buf, 1, &asize );         // Status Read
        if( buf == 0x20 ) // Wakeup End ?
            break;
    }
    if( i == 20 )
        goto Err1;
    for( i=0 ; i < sizeof(IniReg)/sizeof(IniReg[0]) ; i++ ) // Gesture Sensor Initialize

```

```

        tk_swri_dev( dd, 0x73, IniReg[i], 2, &asize );           // Initialize Command
    for( i=0 ; i<2 ; i++ ) {
        tk_swri_dev( dd, 0x73, "%x43", 1, &asize );           // Indicate Register
        tk_srea_dev( dd, 0x73, &buf, 1, &asize );             // Dummy Read
    }
    while( 1 ) {
        tk_dly_tsk( 250 );                                     // Wait 250ms
        tk_swri_dev( dd, 0x73, "%x43", 1, &asize );           // Indicate Upper Register
        data = 0;                                              // Data Clear
        tk_srea_dev( dd, 0x73, &data, 2, &asize );             // Read Sensor Value
        if( ! ( data &= 0x1FF ) )                             // Find Gesture ?
            continue;
#ifdef __BIG
        data = data / 256 + ( data % 256 << 8 );             // Byte Swap
#endif
        for( i=0 ; data >>= 1 ; i++ ) ;                       // Analyze Gesture
        tm_printf("%s\n", msg[i]);                             // Output Gesture Message
    }
Err1:
    tk_cls_dev( dd, 0 );                                       // Close IIC Driver
Err2:
    tk_ext_tsk();                                              // Exit
}

```

起動されたら、I2Cドライバまたは簡易I2Cドライバをオープンし、初期化データを使ってジェスチャーセンサを初期化します。初期化後は、250msの間隔でジェスチャー検出用のレジスタを参照し、検出したジェスチャーに対応した文字列をターミナルに表示します。

ライトタスク (light_tsk)

```

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

    if( ( dd = tk_opn_dev( "iica", TD_UPDATE ) ) <= E_OK ) // Open IIC or SIIC Driver
        goto Err2;
    if( tk_swri_dev( dd, 0x23, "%x10", 1, &asize ) < E_OK ) // Set Continuously H-Resolution Mode
        goto Err1;
    tk_dly_tsk( 180 );                                       // Wait 180ms
    while( 1 ) {
        if( tk_srea_dev( dd, 0x23, buf, 2, &asize ) < E_OK )
            continue;
        data = ( buf[0] << 8 ) + buf[1];                   // Read Measurement Result
        tm_printf("%d\n", data);                             // Make Measurement Result
        tk_dly_tsk( 1000 );                                  // Output Console
        tk_dly_tsk( 1000 );                                  // Wait 1000ms
    }
Err1:
    tk_cls_dev( dd, 0 );                                       // Close IIC or SIIC Driver
Err2:
    tk_ext_tsk();                                              // Exit
}

```

起動されたら、I2Cドライバまたは簡易I2Cドライバをオープンし、センサモードを設定後、1000msの間隔でセンサ値を参照し、その結果をターミナルに表示します。

実行結果

microT-Kernel Version 3.00

13
15
18
3379
5566
6179
3686
12
13
2701
2614
4392
3128
14
5
left
3
down
up
3
right
2
left
3
L turn
3
R turn
3
ByeBye
3
far
3
near
3

2.12.2 EEPROM のサンプルプログラム

usermain関数

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

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

    iicDrvEntry();
    siicDrvEntry();
    t_ctsk.tskatr = TA_HLNG | TA_DSNAME;
    t_ctsk.stksz = 1024;
    t_ctsk.itskpri = 10;
    t_ctsk.task = eeeprom_tsk;
    strcpy( t_ctsk.dsname, "eeeprom" );
    if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK )
        goto Err;
    if( tk_sta_tsk( objid, 0 ) < E_OK )
        goto Err;
    tk_slp_tsk( TMO_FEVR );
Err:
    while( 1 ) ;
}
```

```
}
```

usermain関数では、サービス関数を使ってI2Cドライバや簡易I2Cドライバを登録し、EEPROMを操作するEEPROMタスク (eeprom_tsk) を生成・起動し、自身はtk_slp_tskで待ち状態となります。

なお、I2Cドライバと簡易I2Cドライバで共通の端子を使用することはできません。不要であれば、簡易I2Cドライバのサービス関数 (siicDrvEntry関数) の呼び出しは削除やコメントアウトしても構いません。

EEPROMタスク (eeprom_tsk)

```
#ifdef AP_RX72N
#define BSIZE 2
#else
#define BSIZE 1
#endif

UB buf[128*BSIZE];

EXPORT void eeprom_tsk(INT stacd, void *exinf)
{
    ID dd;
    SZ asize;
    INT i, j;

    if( ( dd = tk_opn_dev( "iica", TD_UPDATE ) ) <= E_OK ) // Open IIC or SIIC Driver
        goto Err;
    tk_swri_dev( dd, 0x50, "%x00", 1, &asize ); // Set Random Read Address
    tk_srea_dev( dd, 0x50, buf, 128*BSIZE, &asize ); // Read Data (Mac Address)
    tm_printf(" %02X:%02X:%02X:%02X:%02X:%02X\n", buf[0], buf[1], buf[2], buf[3], buf[4], buf[5] );
    for( i=0 ; i<8*BSIZE ; i++ ) {
        for( j=0 ; j<16 ; j++ )
            tm_printf(" %02X", buf[(i<<4)+j]);
        tm_printf("\n");
    }
    for( i=16 ; i<128*BSIZE ; i++ )
        if( buf[i] != 0xFF )
            break;
    if( i == 128*BSIZE )
        for( i=16 ; i<128*BSIZE ; i++ )
            buf[i] = i;
    else
        for( i=16 ; i<128*BSIZE ; i++ )
            buf[i] ++;
    for( i=16 ; i<64*BSIZE ; i++ ) {
        buf[i-1] = i;
        tk_swri_dev( dd, 0x50, &buf[i-1], 2, &asize ); // Byte Write
        while( tk_swri_dev( dd, 0x50, "%x00", 0, &asize ) != E_OK ) // Acknowledge Polling
            __nop();
    }
    for( i=64*BSIZE ; i<128*BSIZE ; i+=8*BSIZE ) {
        buf[i-1] = i;
        tk_swri_dev( dd, 0x50, &buf[i-1], 1+8*BSIZE, &asize ); // Page Write
        while( tk_swri_dev( dd, 0x50, "%x00", 0, &asize ) != E_OK ) // Acknowledge Polling
            ;
    }
    tm_printf("\n");
    tk_swri_dev( dd, 0x50, "%x00", 1, &asize ); // Set Random Read Address
    tk_srea_dev( dd, 0x50, buf, 128*BSIZE, &asize ); // Read Data (Mac Address)
    for( i=0 ; i<8*BSIZE ; i++ ) {
        for( j=0 ; j<16 ; j++ )
            tm_printf(" %02X", buf[(i<<4)+j]);
        tm_printf("\n");
    }
}
```

```

    }
    tk_cls_dev( dd, 0 );                                // Close IIC or SIIC Driver
Err:    tk_ext_tsk( );                                    // Exit
}

```

起動されたら、I2Cドライバまたは簡易I2Cドライバをオープンし、EEPROMのブロック0のランダムリードアドレスを 0 に設定します。その後、1 ブロックのデータを読み込みます。AP-RX63N-0AとAP-RX65N-0A搭載の24LC01BTは1 ブロックが128バイト、AP-RX72N-0A搭載の24LC16BTは1 ブロックが256バイトです。製品出荷時に最初の6 バイトにMACアドレスが書き込まれているため、それをターミナルに表示後、1 ブロック分のデータをバイト単位でターミナルに16進で表示します。

その後は、最初の16バイトを除くデータを読み込んだ値とは異なる値に更新します。そして、半分をバイト・ライト、残りの半分のページ・ライトでEEPROMに書き込みます。AP-RX63N-0AとAP-RX65N-0A搭載の24LC01BTはページ・ライトの単位が8 バイト、AP-RX72N-0A搭載の24LC16BTはページ・ライトの単位が16バイトです。また、書き込み完了を確認するアックノレッジ・ポーリングはtk_swri_drvシステムコール（またはtk_wri_devシステムコール）で第3引数のサイズに 0 を指定すれば対応できます。

書き込み後は、再度1 ブロック分のデータを読み込み、ターミナルに表示します。

実行結果

```

microT-Kernel Version 3.00

00:0C:7B:47:02:0B
00 0C 7B 47 02 0B FF FF FF FF FF FF FF FF FF
11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40
41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50
51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80

00 0C 7B 47 02 0B FF FF FF FF FF FF FF FF FF
12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31
32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41
42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51
52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61
62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71
72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81

```

3. 簡易 I2C ドライバ

3.1 対象デバイス

スレーブアドレスが7ビットのI2Cデバイスが対象となります。

3.2 デバイス名

デバイス名は、“siica”、“siicb”、...、“siicl”、“siicm”を使用します。内蔵周辺機能のシリアルコミュニケーションインタフェース（SCI）のチャネル番号とデバイス名の関係は一致しています。ターゲット毎とコア毎に利用可能なデバイス名は以下の通りです。

表3.1 ターゲット毎に利用可能なデバイス名

	siica (SCI0)	siicb (SCI1)	siicc (SCI2)	siicd (SCI3)	siice (SCI4)	siicf (SCI5)	siicg (SCI6)
AP-RX63N-0A			○				
AP-RX65N-0A	○						
AP-RX72N-0A	○	○		○			○
TB-RX231	○	○				○	○
TB-RX23W		○				○	
TB-RX65N	○	○	○	○	○	○	○
TB-RX660	○	○	○	○	○	○	○
TB-RX66N	○	○	○	○	○	○	○

表3.1 ターゲット毎に利用可能なデバイス名

	siich (SCI7)	siici (SCI8)	siicj (SCI9)	siick (SCI10)	siicl (SCI11)	siicm (SCI12)
AP-RX63N-0A						
AP-RX65N-0A						
AP-RX72N-0A					○	
TB-RX231		○	○			○
TB-RX23W						
TB-RX65N		○	○	○	○	○
TB-RX660		○	○		○	○
TB-RX66N			○		○	○

表3.2 コア毎に利用可能なデバイス名

	siica (SCI0)	siicb (SCI1)	siicc (SCI2)	siicd (SCI3)	siice (SCI4)	siicf (SCI5)	siicg (SCI6)
R5F563N (RX63N)	○	○	○	○	○	○	○

R5F5231 (RX231)	○	○				○	○
R5F523W (RX23W)		○				○	
R5F565N (RX65N)	○	○	○	○	○	○	○
R5F5660 (RX660)	○	○	○	○	○	○	○
R5F566N (RX66N)	○	○	○	○	○	○	○
R5F572N (RX72N)	○	○	○	○	○	○	○

表3.2 コア毎に利用可能なデバイス名

	siich (SCI7)	siici (SCI8)	siicj (SCI9)	siick (SCI10)	siicl (SCI11)	siicm (SCI12)
R5F563N (RX63N)	○	○	○	○	○	○
R5F5231 (RX231)		○				○
R5F523W (RX23W)	○					
R5F565N (RX65N)	○	○	○	○	○	○
R5F5660 (RX660)	○	○	○	○	○	○
R5F566N (RX66N)	○	○	○	○	○	○
R5F572N (RX72N)	○	○	○	○	○	○

3.3 固有機能

スレーブアドレスが7ビットのI2Cデバイスとの送受信を行います。

3.4 属性データ

なし

3.5 固有データ

start : I2Cデバイスのスレーブアドレスを指定します。
下位7ビットのみが有効であり、他の上位ビットは無視されます。

buf : 送受信するデータの格納先アドレスを指定します。
送信時に size=0 であれば無視されます。

size : 送受信データのサイズを指定します。
送信（書き込み）時は 0 指定が可能です（Acknowledge Polling等にご利用します）。
受信（読み込み）時の 0 指定は E_PAR となります。
負の値は送受信（読み込み、書き込み）共に E_PAR となります。

3.6 事象通知

なし

3.7 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。簡易I2Cドライバ固有の特殊なエラーコードは存在しません。

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_siic.h です。

簡易I2C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 = siicDrvEntry(void);
```

3.9 簡易 I2C ドライバが使用する資源

3.9.1 使用する資源の概要

簡易I2Cドライバは処理を実施するに当たり、デバイス名毎に1つのタスク、1つのイベントフラグ、3つの割込みハンドラを利用します。サービス関数のAPIを呼び出すと生成または定義されます。

使用する資源	名称	優先度
タスク	siic_tsk	タスクの優先度は SIIC_CFG_TASK_PRIORITY
割込みハンドラ	sci*_rx*_hdr	割込みハンドラの割込み優先度は SIIC_CFG_INT_PRIORITY
	sci*_tx*_hdr	割込みハンドラの割込み優先度は SIIC_CFG_INT_PRIORITY
	sci*_tei*_hdr	割込みハンドラの割込み優先度は SIIC_CFG_INT_PRIORITY

注：SIIC_CFG_TASK_PRIORITY と SIIC_CFG_INT_PRIORITY は次項を参照

* の部分にはRIICのチャネル番号が入る（0～3）。

3.9.2 タスク (siic_tsk)

I2C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	siic_tsk
タスク起動時優先度	SIIC_CFG_TASK_PRIORITY
スタックサイズ	300
DSオブジェクト名称	"siic*_t" (*はデバイス名の最後の文字)

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。

3.9.3 イベントフラグ

I2C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	"siic*_f" (*はデバイス名の最後の文字)

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。

3.9.4 割込みハンドラ (sci*_rx*_hdr、sci*_tx*_hdr、sci*_tei*_hdr)

SCIの割込みハンドラです。割込みの発生によりイベントフラグへのイベントの設定を行います。

項目 (受信データフル)	コア			
		siica	siicb	siicc
割込み番号	R5F563N (RX63N)	214	217	220
	R5F5231 (RX231)	214	218	—
	R5F523W (RX23W)	—	218	—
	R5F565N (RX65N)	58	60	62
	R5F5660 (RX660)	58	60	62
	R5F566N (RX66N)	58	60	62
	R5F572N (RX72N)	58	60	62
割込み優先度	SIIC_CFG_INT_PRIORITY			
割込みハンドラ起動アドレス	sci*_rx*_hdr (*はSCIのチャネル番号)			

コア	値				
	siicd	siice	siicf	siicg	siich
R5F563N (RX63N)	223	226	229	232	235
R5F5231 (RX231)	—	—	222	226	—
R5F523W (RX23W)	—	—	222	—	—
R5F565N (RX65N)	80	82	84	86	98
R5F5660 (RX660)	80	82	84	86	98
R5F566N (RX66N)	80	82	84	86	98
R5F572N (RX72N)	80	82	84	86	98
SIIC_CFG_INT_PRIORITY					
sci*_rx*_hdr (*はSCIのチャネル番号)					

コア	値				
	siici	siicj	siick	siicl	siicm
R5F563N (RX63N)	238	241	244	247	250
R5F5231 (RX231)	230	234	—	—	238
R5F523W (RX23W)	230	—	—	—	238
R5F565N (RX65N)	100	102	104	114	116
R5F5660 (RX660)	100	102	104	114	116
R5F566N (RX66N)	100	102	104	114	116
R5F572N (RX72N)	100	102	104	114	116
SIIC_CFG_INT_PRIORITY					
sci*_rxi*_hdr (*はSCIのチャネル番号)					

項目 (送信データエンプティ)	コア			
		siica	siicb	siicc
割込み番号	R5F563N (RX63N)	215	218	221
	R5F5231 (RX231)	215	219	—
	R5F523W (RX23W)	—	219	—
	R5F565N (RX65N)	59	61	63
	R5F5660 (RX660)	59	61	63
	R5F566N (RX66N)	59	61	63
	R5F572N (RX72N)	59	61	63
割込み優先度	SIIC_CFG_INT_PRIORITY			
割込みハンドラ起動アドレス	sci*_txi*_hdr (*はSCIのチャネル番号)			

コア	値				
	siicd	siice	siicf	siicg	siich
R5F563N (RX63N)	224	227	230	233	236
R5F5231 (RX231)	—	—	223	227	—
R5F523W (RX23W)	—	—	223	—	—
R5F565N (RX65N)	81	83	85	87	99
R5F5660 (RX660)	81	83	85	87	99
R5F566N (RX66N)	81	83	85	87	99
R5F572N (RX72N)	81	83	85	87	99
SIIC_CFG_INT_PRIORITY					
sci*_txi*_hdr (*はSCIのチャネル番号)					

コア	値
----	---

	siici	siicj	siick	siicl	siicm
R5F563N (RX63N)	239	242	245	248	251
R5F5231 (RX231)	231	235	—	—	239
R5F523W (RX23W)	231	—	—	—	239
R5F565N (RX65N)	101	103	105	115	117
R5F5660 (RX660)	101	103	105	115	117
R5F566N (RX66N)	101	103	105	115	117
R5F572N (RX72N)	101	103	105	115	117
SIIC_CFG_INT_PRIORITY					
sci*_txi*_hdr (*はSCIのチャンネル番号)					

項目 (送信終了)	コア	値		
		siica	siicb	siicc
割込み番号、 グループ割込み番号	R5F563N (RX63N)	216	219	222
	R5F5231 (RX231)	216	220	—
	R5F523W (RX23W)	—	220	—
	R5F565N (RX65N)	110、0	110、2	110、4
	R5F5660 (RX660)	110、0	110、2	110、4
	R5F566N (RX66N)	110、0	110、2	110、4
	R5F572N (RX72N)	110、0	110、2	110、4
割込み優先度	SIIC_CFG_INT_PRIORITY			
割込みハンドラ起動アドレス	sci*_tei*_hdr (*はSCIのチャンネル番号)			

コア	値				
	siicd	siice	siicf	siicg	siich
R5F563N (RX63N)	225	228	231	234	237
R5F5231 (RX231)	—	—	224	228	—
R5F523W (RX23W)	—	—	224	—	—
R5F565N (RX65N)	110、6	110、8	110、10	110、12	110、14
R5F5660 (RX660)	110、6	110、8	110、10	110、12	110、14
R5F566N (RX66N)	110、6	110、8	110、10	110、12	112、22
R5F572N (RX72N)	110、6	110、8	110、10	110、12	112、22
SIIC_CFG_INT_PRIORITY					
sci*_tei*_hdr (*はSCIのチャンネル番号)					

コア	値				
	siici	siicj	siick	siicl	siicm

R5F563N (RX63N)	240	243	246	249	252
R5F5231 (RX231)	232	236	—	—	240
R5F523W (RX23W)	232	—	—	—	240
R5F565N (RX65N)	111、24	111、26	112、8	112、12	110、16
R5F5660 (RX660)	111、24	111、26	112、8	112、12	110、16
R5F566N (RX66N)	112、0	112、4	112、8	112、12	110、16
R5F572N (RX72N)	112、0	112、4	112、8	112、12	110、16
SIIC_CFG_INT_PRIORITY					
sci*_tei*_hdr (*はSCIのチャンネル番号)					

3.9.5 グループ割込み (GroupBL0Handler、GroupBL1Handler、GroupAL0Handler)

SCIの送信完了 (TEI*) 割込みがグループ割込みの場合 (RX65NやRX72N等の場合)、割込み番号 (ベクタ番号) 110、111または112の割込みハンドラを定義することになります。ただし、システムで割込み番号110、111または112のグループ割込みに分類された割込みを利用する場合、SCIの割込みと共存する必要があります。

この割込みに対してはデバイスドライバ共通のグループ割込みハンドラ (GroupBL0Handler (ベクタ番号110)、GroupBL1Handler (ベクタ番号111)、GroupAL0Handler (ベクタ番号112)) を利用しています。ソースファイルはgroupbl0.c、groupbl1.cまたはgroupal0.cです。グループ割込みハンドラでは関数ポインタ経由で割込み要求に対応した割込みハンドラを呼び出します。以下にGroupBL0Handlerグループ割込みハンドラのソースコードを示しますが、GroupBL1HandlerやGroupAL0Handlerのグループ割込みハンドラも概要は同じです。

GroupBL0Handlerグループ割込みハンドラ

```
EXPORT void (*GroupBL0Table[32])(UINT dintno);
EXPORT void GroupBL0Handler (UINT dintno)
{
    INT i;
    UW intreqflg;
    intreqflg = ICU.GRPBL0.LONG;
    for( i=0 ; !( intreqflg & 0xFF ) ; i+=8 )
        intreqflg >>= 8;
    for( ; intreqflg ; i++, intreqflg >>= 1 )
        if( intreqflg & 1 )
            GroupBL0Table[i]( dintno );
}
```

GroupBL0Handlerグループ割込みハンドラの中でGRPBL0レジスタの各ステータスフラグをチェックし、各割込みハンドラを呼び出します。

3.10 簡易 I2C ドライバのコンフィグレーション

3.10.1 簡易 I2C ドライバのコンフィグレーション・ファイル

簡易I2Cドライバのコンフィグレーション・ファイルは、mtkernel_3¥device¥siic¥sysdepend¥ターゲット

ット¥siic_config.h です。

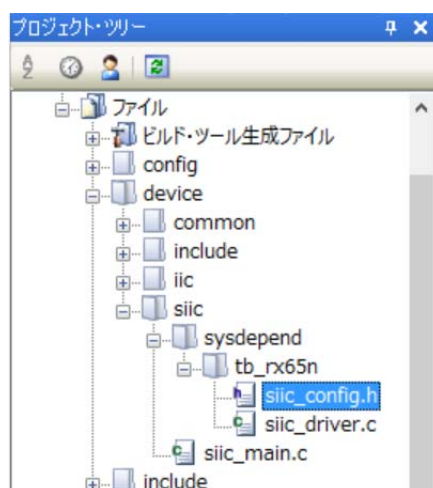


図3.1 簡易I2Cドライバのコンフィグレーション・ファイル

3.10.2 タスクの優先度と割り込みハンドラの割り込み優先度

● SIIC_CFG_TASK_PRIORITY

I2Cの通信処理を実行するタスク (siic_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I2C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 4 に設定されています。

● SIIC_CFG_INT_PRIORITY

内蔵周辺機能のシリアルコミュニケーションインタフェース (SCI) の割り込み優先度です。

システムの都合に応じて、1 ~ MAX_INT_PRI (最高外部割り込み優先度) の範囲で変更できます。TIM_INT_PRI (システムタイマの割り込み優先度) よりも低く設定することを推奨します。デフォルトでは 9 に設定されています。

なお、グループ割り込みを利用する場合、割り込み番号が同一である他の割り込みハンドラと同一の割り込み優先度を指定する必要があります。もし、異なる割り込み優先度を指定している場合、siicDrvEntryのサービス関数の戻り値がエラーとなり、簡易I2Cデバイスドライバの登録が失敗することがあります。

```
/* SIIC control task priority. */  
#define SIIC_CFG_TASK_PRIORITY (4)  
  
/* SIIC interrupt priority level. */  
#define SIIC_CFG_INT_PRIORITY (9)
```

3.10.3 SCI チャネル関連

● USE_SIIC*

内蔵SCIの当該チャネルの使用/未使用を設定します（定義=使用、未定義=未使用）。

使用する場合は目的のマクロ名を定義してください。定義すると3.9節で紹介した資源が生成、または定義されます。未使用の場合は目的のマクロ名をコメントアウトしてください。

● USE_SSCL_P**（USE_SIIC*が未定義の場合は選択不要です）

● USE_SSDA_P**（USE_SIIC*が未定義の場合は選択不要です）

SSCL端子やSSDA端子が複数利用できる場合の選択です。使用する端子のマクロ名を定義し、使用しない端子のマクロ名はコメントアウトしてください。なお、特定のチャネルに関しては選択ができない場合（固定の場合）もあります。また、全てをコメントアウトしたり、複数のマクロ名を定義すると正しい動作は得られないので注意してください。

● SIIC*_FSCL（USE_IIC*が未定義の場合は設定不要です）

SCLクロックの周波数を指定します。設定値の単位はHzです。

上記の設定値は接続するI2Cデバイスのデータシート等に記載されています。次頁にBH1750FVIの光センサとPAJ7620U2のジェスチャーセンサの例を示します。

BH1750FVIの光センサ

Parameter	Symbol	Limits			Units	Conditions
		Min.	Typ.	Max.		
I2C SCL Clock Frequency	fSCL	—	—	400	kHz	

PAJ7620U2のジェスチャーセンサ

Parameter	Symbol	STANDARD MODE		FAST MODE		Unit
		Min.	Max.	Min.	Max.	
SCL clock frequency.	fscI	10	100	10	400	kHz

#define	USE_SIIC0		/* Use SCIO */
#define	USE_SSCL0_P21		/* Use SCIO SSCL0 is P21 */
//#define	USE_SSCL0_P33		/* Use SCIO SSCL0 is P33 */
#define	USE_SSDA0_P20		/* Use SCIO SSDA0 is P20 */
//#define	USE_SSDA0_P32		/* Use SCIO SSDA0 is P32 */
#define	SIIC0_FSCL	400000	/* SIIC0 SCL Frequency(Hz) */

3.11 スタックサイズ

3.11.1 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
_siic_exec (112)	S	0xffff82e48	119	24	siic_main.obj
_siic_close (4)	S	0xffff82e45	3	4	siic_main.obj
_sci1_tx1_hdr (96)	S	0xffff8319e	6	4	siic_driver.obj
_sci1_rx1_hdr (96)	S	0xffff83191	13	4	siic_driver.obj
_sci1_te1_hdr (96)	S	0xffff831a4	13	4	siic_driver.obj
_siic_wait (116)	S	0xffff82ebf	35	12	siic_main.obj
_siic_abort (104)	S	0xffff82ee2	60	16	siic_main.obj
_siic_tsk (188)	S	0xffff82f21	192	44	siic_main.obj
_siic_event (4)	S	0xffff82f1e	3	4	siic_main.obj
_siic_open (4)	S	0xffff82e42	3	4	siic_main.obj

図3.2 簡易I2Cデバイスドライバのスタック見積もり結果

3.11.2 siic_tsk のスタックサイズ

siic_tskタスクが独自に使用するスタックサイズは188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300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300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する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¥siic¥sysdepend¥ターゲット¥siic_driver.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。

```
p_ctsk->stksz = 300; // Set Task StackSize
p_ctsk->itskpri = SIIC_CFG_TASK_PRIORITY; // Set Task Priority
```

3.11.3 割込みハンドラのスタックサイズ

図3.2の例では、sci1_rx1_hdr、sci1_tx1_hdr、sci1_te1_hdrの3つが割込みハンドラです（SCIチャネル1の3つの割込み）。図3.2の例では全て96バイトですが、異なる場合は、最大のサイズが簡易I2Cデバイスドライバのコンフィグレーションで指定した SIIC_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズとなります。この数値を使って例外スタックのサイズを算出してください。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

3.11.4 グループ割込み

RX65NやRX72N等のように送信完了の割込みハンドラ（sci*_tei*_hdr）がグループ割込みハンドラ経由で実行される場合、スタック見積もりツールの表示結果は図3.3のようになります。**送信完了の割込みハンドラとグループ割込みハンドラの対応は3.9.4項や3.9.5項を参照ください。**

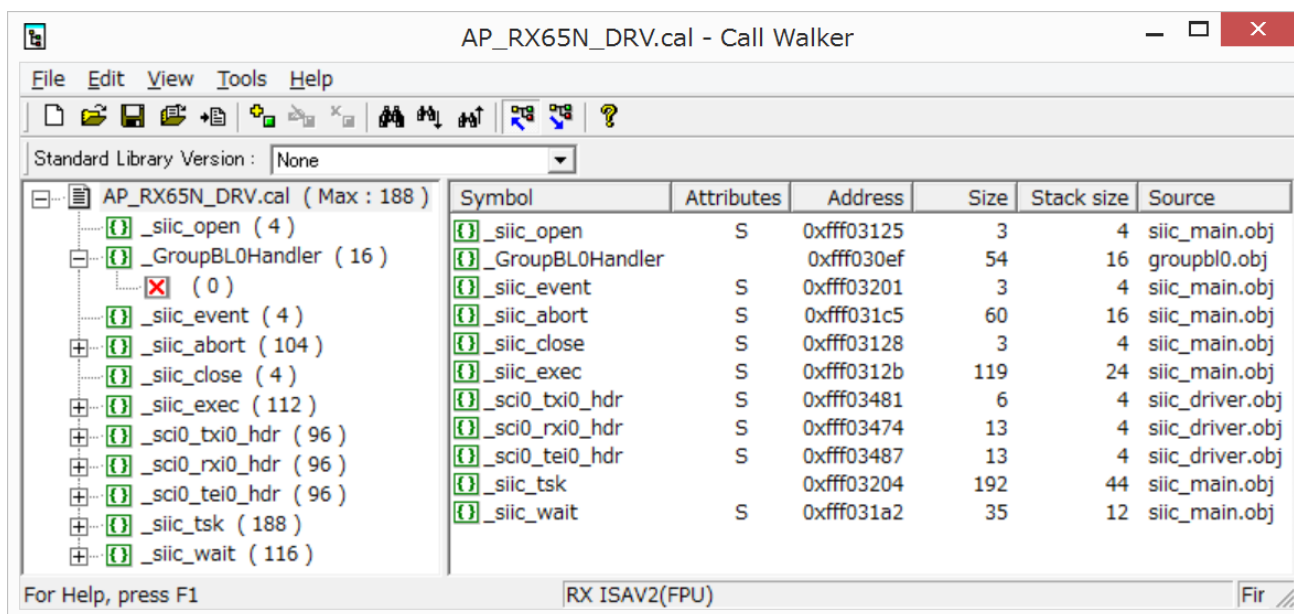


図3.3 グループ割込みハンドラを使用する場合のスタック見積もり結果

上記の場合 (RX65N)、sci0_tei0_hdr割込みハンドラはGroupBL1Handlerグループ割込みハンドラから関数ポインタ経由で実行されます。結果、割込みハンドラのみをドラッグ&ドロップで未解決シンボルを修正すると図3.4のようになります。

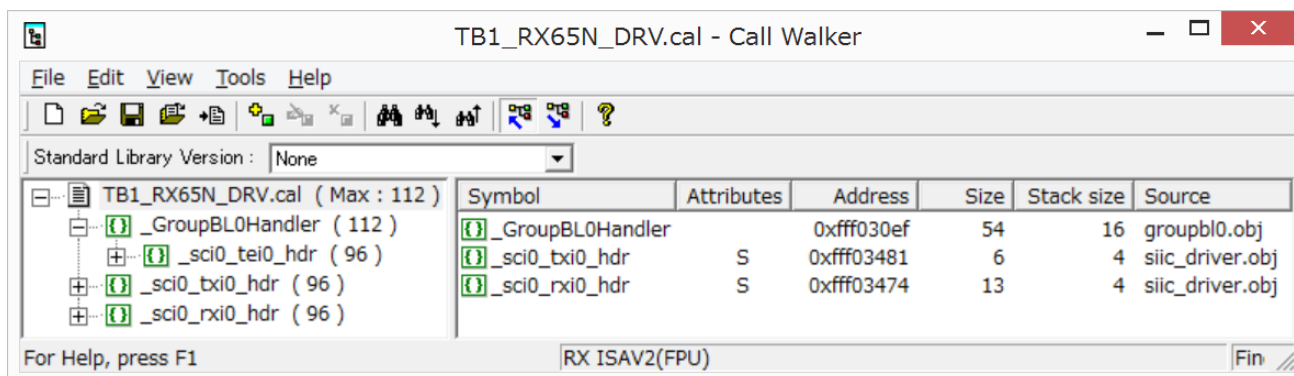


図3.4 ドラッグ&ドロップで未解決シンボルを修正した結果

結果、GroupBL0Handlerグループ割込みハンドラを利用する場合、sci0_tei0_hdr割込みハンドラはGroupBL0Handlerグループ割込みハンドラが使用するサイズを加算したサイズで算出することになります。

図3.3の例であれば、SIIC_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズは112バイトで計算することになります。

3.11.5 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 (siic_open、siic_close、

siic_exec、siic_wait、siic_abort、siic_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。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

3.12 サンプルプログラム

第2章のI2Cドライバのサンプルプログラムと同じです。デバイス名を簡易I2Cドライバのドライブ名に変更してご利用ください。

なお、I2Cドライバと簡易I2Cドライバで共通の端子を使用することはできません。不要であれば、I2Cドライバのサービス関数(iicDrvEntry関数)の呼び出しは削除やコメントアウトしても構いません。

4. シリアルドライバ

4.1 対象デバイス

調歩同期式のシリアルデバイスが対象となります。

4.2 デバイス名

デバイス名は、“scia”、“scib”、...、“scil”、“scim” を使用します。内蔵周辺機能のシリアルコミュニケーションインタフェース（SCI）のチャンネル番号とデバイス名の関係は一致しています。ターゲット毎とコア毎に利用可能なデバイス名は以下の通りです。

表4.1 ターゲット毎に利用可能なデバイス名

	scia (SCI0)	scib (SCI1)	scic (SCI2)	scid (SCI3)	scie (SCI4)	scif (SCI5)	scig (SCI6)
AP-RX65N-0A	○						
AP-RX72N-0A	○	○		○			○
TB-RX231	○	○				○	○
TB-RX23W		○				○	
TB-RX65N	○	○	○	○	○	○	○
TB-RX660	○	○	○	○	○	○	○
TB-RX66N	○	○	○	○	○	○	○

表4.1 ターゲット毎に利用可能なデバイス名

	scih (SCI7)	scii (SCI8)	scij (SCI9)	scik (SCI10)	scil (SCI11)	scim (SCI12)
AP-RX65N-0A						
AP-RX72N-0A					○	
TB-RX231		○	○			○
TB-RX23W						
TB-RX65N		○	○	○	○	○
TB-RX660		○	○		○	○
TB-RX66N			○		○	○

表4.2 コア毎に利用可能なデバイス名

	scia (SCI0)	scib (SCI1)	scic (SCI2)	scid (SCI3)	scie (SCI4)	scif (SCI5)	scig (SCI6)
R5F5231 (RX231)	○	○				○	○
R5F523W (RX23W)		○				○	
R5F565N (RX65N)	○	○	○	○	○	○	○

R5F5660 (RX660)	○	○	○	○	○	○	○
R5F566N (RX66N)	○	○	○	○	○	○	○
R5F572N (RX72N)	○	○	○	○	○	○	○

表4.2 コア毎に利用可能なデバイス名

	scih (SCI7)	scii (SCI8)	scij (SCI9)	scik (SCI10)	scil (SCI11)	scim (SCI12)
R5F5231 (RX231)		○				○
R5F523W (RX23W)	○					
R5F565N (RX65N)	○	○	○	○	○	○
R5F5660 (RX660)	○	○	○	○	○	○
R5F566N (RX66N)	○	○	○	○	○	○
R5F572N (RX72N)	○	○	○	○	○	○

4.3 固有機能

調歩同期式のシリアルデバイスとの送受信を行います。

4.4 属性データ

以下の属性データをサポートします。

```
typedef enum {
    DN_RSMODE = -100,      /* 通信モード */
} RSDataNo;
```

DN_RSMODE: 通信モード (RW: 読み込み/書き込み可)

data: RsMode

```
typedef struct {
    UW    parity   : 2;    /* 0:なし, 1:奇数, 2:偶数, 3:- */
    UW    datalen  : 2;    /* 0:5bit, 1:6bit, 2:7bit, 3:8bit */
    UW    stopbits : 2;    /* 0:1bit, 1:1.5bit, 2:2bit, 3:- */
    UW          : 2;    /* 予約 */
    UW    baud     : 24;   /* ボーレート */
} RsMode;
```

parity: 0:なし 1:奇数 2:偶数 3:-

datalen: 0:5bit 1:6bit 1:7bit 3:8bit

stopbits: 0:1bit 1:1.5bit 2:2bit 3:-

baud: 通信速度 (bps)

通信モードを設定する/取り出す。

サポートしていない設定 (青字) をした場合はエラーとなります。

設定 (書き込み) により通信環境は以下のように初期化されます。

- ・ 受信バッファクリア
- ・ 送信バッファクリア

4.5 固有データ

start : 0 に固定。

buf : 送受信するデータの格納先アドレスを指定します。

size : 送受信データのサイズを指定します。

送信（書き込み）時は常に 0 を返します（何も送信しません）。

受信（読み込み）時は受信済み（受信バッファに溜まっている）バイト数を有効データ長として返します。

4.6 事象通知

なし

4.7 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。シリアルドライバ固有の特殊なエラーコードは存在しません。

4.8 ヘッダファイルとサービス関数

ヘッダファイルは dev_sci.h です。

シリア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 = sciDrvEntry(void);
```

4.9 シリアルドライバが使用する資源

4.9.1 使用する資源の概要

シリアルドライバは処理を実施するに当たり、デバイス名毎に1つのタスク、1つのイベントフラグ、4つの割込みハンドラを利用します。サービス関数のAPIを呼び出すと生成または定義されます。

使用する資源	名称	優先度
タスク	sci_tsk	タスクの優先度は SCI_CFG_TASK_PRIORITY
割込みハンドラ	sci*_rx*_hdr	割込みハンドラの割込み優先度は SCI_CFG_INT_PRIORITY
	sci*_tx*_hdr	割込みハンドラの割込み優先度は SCI_CFG_INT_PRIORITY
	sci*_tei*_hdr	割込みハンドラの割込み優先度は SCI_CFG_INT_PRIORITY
	sci*_eri*_hdr	割込みハンドラの割込み優先度は SCI_CFG_INT_PRIORITY

注：SCI_CFG_TASK_PRIORITY と SCI_CFG_INT_PRIORITY は次項を参照

* の部分にはSCIのチャンネル番号が入る（0～3）。

4.9.2 タスク（sci_tsk）

シリアル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	sci_tsk
タスク起動時優先度	SCI_CFG_TASK_PRIORITY
スタックサイズ	300
DSオブジェクト名称	"sci*_t" (*はデバイス名の最後の文字)

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。

4.9.3 イベントフラグ

シリアル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	"sci*_f" (*はデバイス名の最後の文字)

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。

4.9.4 割り込みハンドラ (sci*_rx*_hdr、sci*_tx*_hdr、sci*_tei*_hdr、sci*_eri*_hdr)

SCIの割り込みハンドラです。割り込みの発生によりイベントフラグへのイベントの設定を行います。

項目 (受信データフル)	コア			
		scia	scib	scic
割り込み番号	R5F5231 (RX231)	214	218	—
	R5F523W (RX23W)	—	218	—
	R5F565N (RX65N)	58	60	62
	R5F5660 (RX660)	58	60	62
	R5F566N (RX66N)	58	60	62
	R5F572N (RX72N)	58	60	62
割り込み優先度	SCI_CFG_INT_PRIORITY			
割り込みハンドラ起動アドレス	sci*_rx*_hdr (*はSCIのチャネル番号)			

コア	値				
	scid	scie	scif	scig	scih
R5F5231 (RX231)	—	—	222	226	—
R5F523W (RX23W)	—	—	222	—	—
R5F565N (RX65N)	80	82	84	86	98
R5F5660 (RX660)	80	82	84	86	98
R5F566N (RX66N)	80	82	84	86	98
R5F572N (RX72N)	80	82	84	86	98
SCI_CFG_INT_PRIORITY					
sci*_rxi*_hdr (*はSCIのチャンネル番号)					

コア	値				
	scii	scij	scik	scil	scim
R5F5231 (RX231)	230	234	—	—	238
R5F523W (RX23W)	230	—	—	—	238
R5F565N (RX65N)	100	102	104	114	116
R5F5660 (RX660)	100	102	104	114	116
R5F566N (RX66N)	100	102	104	114	116
R5F572N (RX72N)	100	102	104	114	116
SCI_CFG_INT_PRIORITY					
sci*_rxi*_hdr (*はSCIのチャンネル番号)					

項目 (送信データエンプティ)	コア			
		scia	scib	scic
割込み番号	R5F5231 (RX231)	215	219	—
	R5F523W (RX23W)	—	219	—
	R5F565N (RX65N)	59	61	63
	R5F5660 (RX660)	59	61	63
	R5F566N (RX66N)	59	61	63
	R5F572N (RX72N)	59	61	63
割込み優先度	SCI_CFG_INT_PRIORITY			
割込みハンドラ起動アドレス	sci*_txi*_hdr (*はSCIのチャンネル番号)			

コア	値				
	scid	scie	scif	scig	scih
R5F5231 (RX231)	—	—	223	227	—
R5F523W (RX23W)	—	—	223	—	—

R5F565N (RX65N)	81	83	85	87	99
R5F5660 (RX660)	81	83	85	87	99
R5F566N (RX66N)	81	83	85	87	99
R5F572N (RX72N)	81	83	85	87	99
SCI_CFG_INT_PRIORITY					
sci*_txi*_hdr (*はSCIのチャネル番号)					

コア	値				
	scii	scij	scik	scil	scim
R5F5231 (RX231)	231	235	—	—	239
R5F523W (RX23W)	231	—	—	—	239
R5F565N (RX65N)	101	103	105	115	117
R5F5660 (RX660)	101	103	105	115	117
R5F566N (RX66N)	101	103	105	115	117
R5F572N (RX72N)	101	103	105	115	117
SIIC_CFG_INT_PRIORITY					
sci*_txi*_hdr (*はSCIのチャネル番号)					

項目 (送信終了)	コア			
		scia	scib	scic
割込み番号、 グループ割込み番号	R5F5231 (RX231)	216	220	—
	R5F523W (RX23W)	—	220	—
	R5F565N (RX65N)	110、0	110、2	110、4
	R5F5660 (RX660)	110、0	110、2	110、4
	R5F566N (RX66N)	110、0	110、2	110、4
	R5F572N (RX72N)	110、0	110、2	110、4
割込み優先度	SCI_CFG_INT_PRIORITY			
割込みハンドラ起動アドレス	sci*_tei*_hdr (*はSCIのチャネル番号)			

コア	値				
	scid	scie	scif	scig	scih
R5F5231 (RX231)	—	—	224	228	—
R5F523W (RX23W)	—	—	224	—	—
R5F565N (RX65N)	110、6	110、8	110、10	110、12	110、14
R5F5660 (RX660)	110、6	110、8	110、10	110、12	110、14
R5F566N (RX66N)	110、6	110、8	110、10	110、12	112、22
R5F572N (RX72N)	110、6	110、8	110、10	110、12	112、22

SCI_CFG_INT_PRIORITY
sci*_tei*_hdr (*はSCIのチャンネル番号)

コア	値				
	scii	scij	scik	scil	scim
R5F5231 (RX231)	232	236	—	—	240
R5F523W (RX23W)	232	—	—	—	240
R5F565N (RX65N)	111、24	111、26	112、8	112、12	110、16
R5F5660 (RX660)	111、24	111、26	112、8	112、12	110、16
R5F566N (RX66N)	112、0	112、4	112、8	112、12	110、16
R5F572N (RX72N)	112、0	112、4	112、8	112、12	110、16
SCI_CFG_INT_PRIORITY					
sci*_tei*_hdr (*はSCIのチャンネル番号)					

項目 (送信終了)	コア			
		scia	scib	scic
割込み番号、 グループ割込み番号	R5F5231 (RX231)	217	221	—
	R5F523W (RX23W)	—	221	—
	R5F565N (RX65N)	110、1	110、3	110、5
	R5F5660 (RX660)	110、1	110、3	110、5
	R5F566N (RX66N)	110、1	110、3	110、5
	R5F572N (RX72N)	110、1	110、3	110、5
割込み優先度	SCI_CFG_INT_PRIORITY			
割込みハンドラ起動アドレス	sci*_eri*_hdr (*はSCIのチャンネル番号)			

コア	値				
	scid	scie	scif	scig	scih
R5F5231 (RX231)	—	—	225	229	—
R5F523W (RX23W)	—	—	225	—	—
R5F565N (RX65N)	110、7	110、9	110、11	110、13	110、15
R5F5660 (RX660)	110、7	110、9	110、11	110、13	110、15
R5F566N (RX66N)	110、7	110、9	110、11	110、13	112、23
R5F572N (RX72N)	110、7	110、9	110、11	110、13	112、23
SCI_CFG_INT_PRIORITY					
sci*_eri*_hdr (*はSCIのチャンネル番号)					

コア	値
----	---

	scii	scij	scik	scil	scim
R5F5231 (RX231)	233	237	—	—	241
R5F523W (RX23W)	233	—	—	—	241
R5F565N (RX65N)	111、25	111、27	112、9	112、13	110、17
R5F5660 (RX660)	111、25	111、27	112、9	112、13	110、17
R5F566N (RX66N)	112、1	112、5	112、9	112、13	110、17
R5F572N (RX72N)	112、1	112、5	112、9	112、13	110、17
SCI_CFG_INT_PRIORITY					
sci*_eri*_hdr (*はSCIのチャンネル番号)					

4.9.5 グループ割込み (GroupBL0Handler、GroupBL1Handler、GroupAL0Handler)

SCIの送信完了 (TEI*) 割込みや受信エラー (ERI*) 割込みがグループ割込みの場合 (RX65NやRX72N等の場合)、割込み番号 (ベクタ番号) 110、111または112の割込みハンドラを定義することになります。ただし、システムで割込み番号110、111または112のグループ割込みに分類された割込みを利用する場合、SCIの割込みと共存する必要があります。

この割込みに対してはデバイスドライバ共通のグループ割込みハンドラ (GroupBL0Handler (ベクタ番号110)、GroupBL1Handler (ベクタ番号111)、GroupAL0Handler (ベクタ番号112)) を利用しています。ソースファイルはgroupbl0.c、groupbl1.cまたはgroupal0.cです。グループ割込みハンドラでは関数ポインタ経由で割込み要求に対応した割込みハンドラを呼び出します。以下にGroupBL0Handlerグループ割込みハンドラのソースコードを示しますが、GroupBL1HandlerやGroupAL0Handlerのグループ割込みハンドラも概要は同じです。

GroupBL0Handlerグループ割込みハンドラ

```
EXPORT void (*GroupBL0Table[32])(UINT dintno);
EXPORT void GroupBL0Handler (UINT dintno)
{
    INT i;
    UW intreqflg;
    intreqflg = ICU.GRPBL0.LONG;
    for( i=0 ; !( intreqflg & 0xFF ) ; i+=8 )
        intreqflg >>= 8;
    for( ; intreqflg ; i++, intreqflg >>= 1 )
        if( intreqflg & 1 )
            GroupBL0Table[i]( dintno );
}
```

GroupBL0Handlerグループ割込みハンドラの中でGRPBL0レジスタの各ステータスフラグをチェックし、各割込みハンドラを呼び出します。

4.10 シリアルドライバのコンフィグレーション

4.10.1 シリアルドライバのコンフィグレーション・ファイル

シリアルドライバのコンフィグレーション・ファイルは、mtkernel_3¥device¥sci¥sysdepend¥ターゲット

ット¥sci_config.h です。

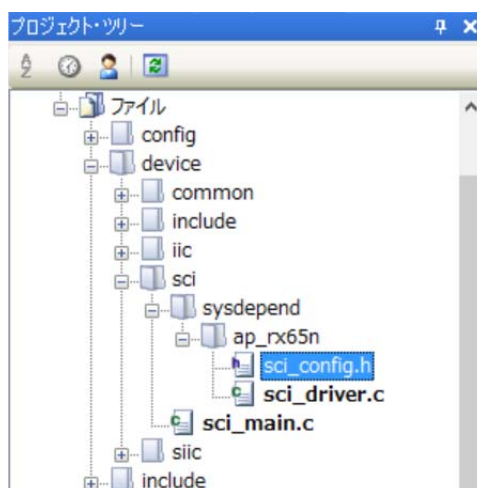


図4.1 シリアルドライバのコンフィグレーション・ファイル

4.10.2 タスクの優先度と割り込みハンドラの割り込み優先度

● SCI_CFG_TASK_PRIORITY

シリアル通信処理を実行するタスク (sci_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 4 に設定されています。

● SCI_CFG_INT_PRIORITY

内蔵周辺機能のシリアルコミュニケーションインタフェース (SCI) の割り込み優先度です。

システムの都合に応じて、1 ~ MAX_INT_PRI (最高外部割り込み優先度) の範囲で変更できます。TIM_INT_PRI (システムタイマの割り込み優先度) よりも低く設定することを推奨します。デフォルトでは 9 に設定されています。

なお、**グループ割り込みを利用する場合、割り込み番号が同一である他の割り込みハンドラと同一の割り込み優先度を指定する必要があります。**もし、異なる割り込み優先度を指定している場合、sciDrvEntryのサービス関数の戻り値がエラーとなり、シリアルデバイスドライバの登録が失敗することがあります。

● SCI_CFG_RBUF_SIZE

受信バッファのサイズです。読み込みのシステムコール (tk_era_dev、tk_srea_dev) が発行されていない場合は、この設定値で指定した受信バッファに受信データが格納されます。受信バッファを超える受信データは破棄されます。デフォルトは32 (単位はバイト) が指定されています。

```
/* SCI control task priority. */  
#define SCI_CFG_TASK_PRIORITY (4)
```

```

/* SCI interrupt priority level. */
#define SCI_CFG_INT_PRIORITY          (9)

/* SCI receive buffer size. */
#define SCI_CFG_RBUF_SIZE            (32)

```

4.10.3 SCI チャンネル関連

● USE_SCI*

内蔵SCIの当該チャンネルの使用/未使用を設定します（定義=使用、未定義=未使用）。

使用する場合は目的のマクロ名を定義してください。定義すると4.9節で紹介した資源が生成、または定義されます。未使用の場合は目的のマクロ名をコメントアウトしてください。

● USE_RXD_P**（USE_SCI*が未定義の場合は選択不要です）

● USE_TXD_P**（USE_SCI*が未定義の場合は選択不要です）

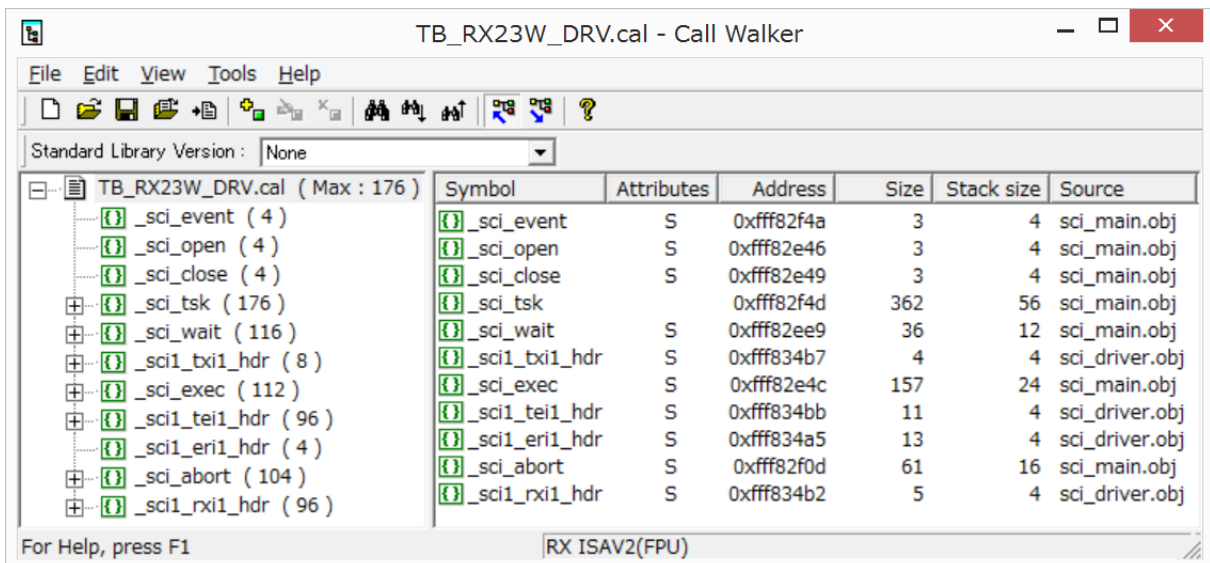
RXD端子やTXD端子が複数利用できる場合の選択です。使用する端子のマクロ名を定義し、使用しない端子のマクロ名はコメントアウトしてください。なお、特定のチャンネルに関しては選択ができない場合（固定の場合）もあります。また、全てをコメントアウトしたり、複数のマクロ名を定義すると正しい動作は得られないので注意してください。

4.11 スタックサイズ

4.11.1 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
_sci_event (4)	_sci_event S	0xfff82f4a	3	4	sci_main.obj
_sci_open (4)	_sci_open S	0xfff82e46	3	4	sci_main.obj
_sci_close (4)	_sci_close S	0xfff82e49	3	4	sci_main.obj
_sci_tsk (176)	_sci_tsk	0xfff82f4d	362	56	sci_main.obj
_sci_wait (116)	_sci_wait S	0xfff82ee9	36	12	sci_main.obj
_sci1_txi1_hdr (8)	_sci1_txi1_hdr S	0xfff834b7	4	4	sci_driver.obj
_sci_exec (112)	_sci_exec S	0xfff82e4c	157	24	sci_main.obj
_sci1_tei1_hdr (96)	_sci1_tei1_hdr S	0xfff834bb	11	4	sci_driver.obj
_sci1_eri1_hdr (4)	_sci1_eri1_hdr S	0xfff834a5	13	4	sci_driver.obj
_sci_abort (104)	_sci_abort S	0xfff82f0d	61	16	sci_main.obj
_sci1_rxi1_hdr (96)	_sci1_rxi1_hdr S	0xfff834b2	5	4	sci_driver.obj

図4.2 シリアルデバイスドライバのスタック見積もり結果

4.11.2 sci_tsk のスタックサイズ

sci_tskタスクが独自に使用するスタックサイズは176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300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300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する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¥sci¥sysdepend¥ターゲット¥sci_driver.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。

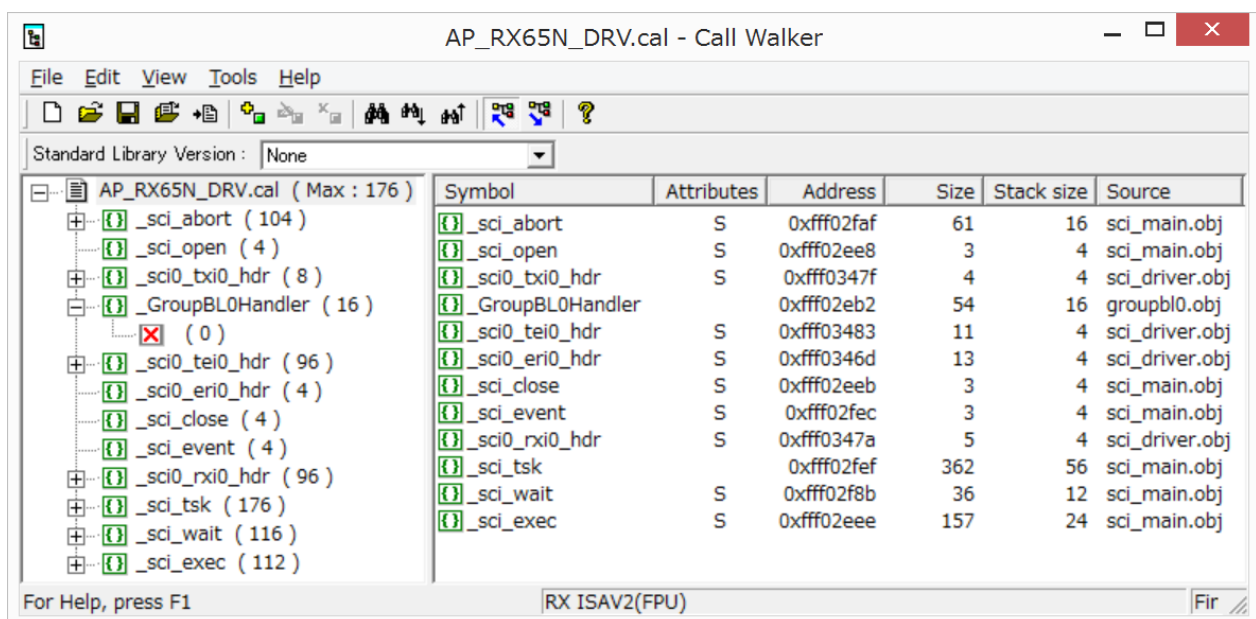
```
p_ctsk->stksz = 300;                // Set Task StackSize
p_ctsk->itskpri = SCI_CFG_TASK_PRIORITY; // Set Task Priority
```

4.11.3 割込みハンドラのスタックサイズ

図4.2の例では、sci1_rx1_hdr、sci1_tx1_hdr、sci1_te1_hdr、sci1_er1_hdrの4つが割込みハンドラです（SCIチャネル1の4つの割込み）。図4.2の例では全て96バイトですが、異なる場合は、最大のサイズがシリアルデバイスドライバのコンフィグレーションで指定した SCI_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズとなります。この数値を使って例外スタックのサイズを算出してください。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

4.11.4 グループ割込み

RX65NやRX72N等のように送信完了の割込みハンドラ（sci*_tei*_hdr、sci*_eri*_hdr）がグループ割込みハンドラ経由で実行される場合、スタック見積もりツールの表示結果は図4.3のようになります。**送信完了の割込みハンドラ、受信エラー割込みハンドラとグループ割込みハンドラの対応は4.9.4項や4.9.5項を参照ください。**



Symbol	Attributes	Address	Size	Stack size	Source
_sci_abort (104)	S	0xffff02faf	61	16	sci_main.obj
_sci_open (4)	S	0xffff02ee8	3	4	sci_main.obj
_sci0_tx0_hdr (8)	S	0xffff0347f	4	4	sci_driver.obj
_GroupBL0Handler (16)	S	0xffff02eb2	54	16	groupbl0.obj
(0)	S	0xffff03483	11	4	sci_driver.obj
_sci0_te0_hdr (96)	S	0xffff0346d	13	4	sci_driver.obj
_sci0_eri0_hdr (4)	S	0xffff02eeb	3	4	sci_main.obj
_sci_close (4)	S	0xffff02fec	3	4	sci_main.obj
_sci_event (4)	S	0xffff0347a	5	4	sci_driver.obj
_sci0_rx0_hdr (96)	S	0xffff02fef	362	56	sci_main.obj
_sci_tsk (176)	S	0xffff02f8b	36	12	sci_main.obj
_sci_wait (116)	S	0xffff02eee	157	24	sci_main.obj
_sci_exec (112)	S				

図4.3 グループ割込みハンドラを使用する場合のスタック見積もり結果

上記の場合（RX65N）、sci0_tei0_hdrとsci0_eri0_hdrの割込みハンドラはGroupBL1Handlerグループ割込みハンドラから関数ポインタ経由で実行されます。結果、割込みハンドラのみをドラッグ&ドロップで未解決シンボルを修正すると図4.4のようになります。

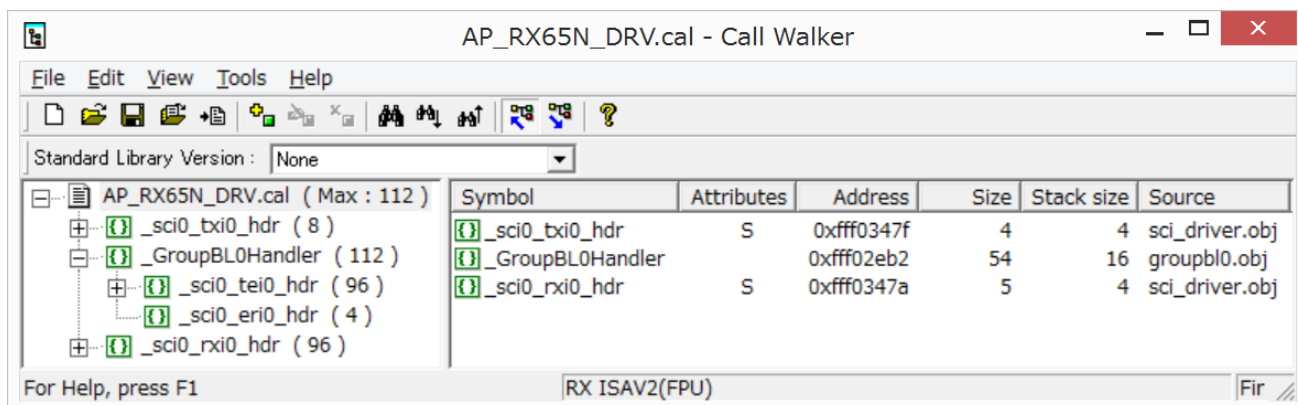


図4.4 ドラッグ&ドロップで未解決シンボルを修正した結果

結果、GroupBL0Handlerグループ割込みハンドラを利用する場合、sci0_tei0_hdrとsci0_eri0_hdrの割込みハンドラはGroupBL0Handlerグループ割込みハンドラが使用するサイズを加算したサイズで算出することになります。

図4.3の例であれば、SCI_CFG_INT_PRIORITY 割込み優先度で実行される割込みハンドラのスタックサイズは112バイトで計算することになります。

4.11.5 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（sci_open、sci_close、sci_exec、sci_wait、sci_abort、sci_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。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

4.12 サンプルプログラム

本実装では単純なシリアル送受信（エコーバック）を行うサンプルプログラムを準備しています。

なお、サンプルプログラムでは「CLANGSPEC」のマクロ定義を利用しています。「CLANGSPEC」のマクロ定義に関しては各ターゲットの構築仕様書を参照してください。

usermain関数

```
#include <string.h>
#include <tk/tkernel.h>
#include "dev_sci.h"

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

```

sciDrvEntry(); // Entry SCI Driver
t_ctsk.tskatr = TA_HLNG | TA_DSNAME; // Set Task Attribute
t_ctsk.stksz = 1024; // Set Task Stack Size
t_ctsk.itspri = 10; // Set Task Priority
t_ctsk.task = echo_tsk; // Set Task Start Address
strcpy( t_ctsk.dsname, "echo" ); // Set Debugger Support Name
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) // Create Echo Back Task
    goto Err;
if( tk_sta_tsk( objid, 0 ) < E_OK ) // Start Echo Back Task
    goto Err;
tk_slp_tsk( TMO_FEVR ); // Sleep
Err:
    while( 1 ) ;
}

```

usermain関数では、サービス関数を使ってシリアルドライバを登録し、シリアルのエコーバックを行うタスク（echo_tsk）を生成・起動し、自身はtk_slp_tskで待ち状態となります。

エコータスク（echo_tsk）

```

#define LEN 5
EXPORT void echo_tsk(INT stacd, void *exinf)
{
    RsMode rsm = { 0, 3, 0, 115200 }; // 8bit, Non Parity, 1Stop bit
    ID dd, reqid;
    VB buf[2][LEN];
    SZ asize;
    ER ioer;
    if( ( dd = tk_opn_dev( "scia", TD_UPDATE ) ) <= E_OK ) // Open SCI Driver
        goto Err2;
    if( tk_swri_dev( dd, DN_RSMODE, &rsm, sizeof(rsm), &asize ) < E_OK ) // Set Communication Mode
        goto Err1;
    while( 1 ) {
        reqid = tk_rea_dev( dd, 0, buf[0], LEN, TMO_FEVR ); // Receive Request
        while( reqid != tk_wai_dev( dd, 0, &asize, &ioer, TMO_FEVR ) ) // Wait Request End
            ;
        strncpy( buf[1], buf[0], LEN ); // Copy Receive Data
        tk_wri_dev( dd, 0, buf[1], LEN, TMO_FEVR ); // Echo Back (Send Request)
    }
Err1:
    tk_cls_dev( dd, 0 ); // Close IIC or SIIC Driver
Err2:
    tk_ext_tsk(); // Exit
}

```

起動されたら、シリアルドライバをオープンし、通信モードを設定（書き込み）します。設定後は、LENマクロで指定した文字数を受信し、受信後はエコーバックします。

5. データフラッシュドライバ

5.1 対象デバイス

データフラッシュが対象となります。

5.2 デバイス名

デバイス名は“DFLASH”です。

<dev_dfl.h>をインクルードすることでDFL_DEVNMマクロ名で参照できます。

5.3 固有機能

データフラッシュとのリード・ライト及びブランクチェック、イレースを行います。

5.4 属性データ

なし

5.5 固有データ

start : 読み込み（リード）、書き込み（ライト）を行う領域の先頭アドレスを指定します。
 データフラッシュの先頭を 0 番地とし、指定するアドレスは 4 の倍数番地でなければなりません。4 の倍数番地でない場合は E_PAR となります。

buf : 読み込み、書き込みするデータの格納先アドレスを指定します。格納先アドレスは 4 の倍数番地でなくても構いません。

size : 読み込み、書き込みするデータのサイズを指定します。
 本パラメータの単位はデータフラッシュの書込みサイズに合わせて 4 バイトです。
 1 を指定すれば 4 バイト、2 を指定すれば 8 バイトとなります。
 startを起点として、データフラッシュの容量を超えない範囲で指定可能です。

5.6 事象通知

データフラッシュのブランクチェックとブロックイレースはtk_evt_devシステムコールを介して、要求します。以下、tk_evt_devシステムコールのパラメータに関する情報を記載します。

devid : デバイスIDです。tk_ref_devシステムコールで取得したデバイスIDを指定します。

例 : devid = tk_ref_dev(dd, NULL); // ddはデバイスディスクリプタ

evttyp : 以下のドライバ要求イベントタイプをサポートします。

```
typedef enum {  
    TDV_BLANKCHECK4    = 0,           // 4バイト領域のブランクチェック  
    TDV_BLANKCHECK64   = 1,           // 64バイト領域のブランクチェック  
    TDV_BLOCKERASE64   = 2,           // 64バイト領域のブロックイレース  
    TDV_BLOCKERASE128  = 3,           // 128バイト領域のブロックイレース  
    TDV_BLOCKERASE256  = 4,           // 256バイト領域のブロックイレース  
} DataFlashEventType;
```


evtinf : データフラッシュの先頭を 0 番地とし、ブランクチェックやブロックイレースを行う領域の先頭アドレスを指定します。

指定する先頭アドレスはアライメントの制約を受けます。例えば、64バイト領域のブランクチェックやブロックイレースを行うのであれば、指定する先頭アドレスは64バイトの境界でなければなりません。本制約に違反すると E_PAR となります。

5.7 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。データフラッシュドライバ固有の特殊なエラーコードは存在しません。

5.8 ヘッダファイルとサービス関数

ヘッダファイルは dev_dfl.h です。

シリア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 = dfldrvEntry(void);
```

5.9 データフラッシュドライバが使用する資源

5.9.1 データフラッシュドライバが使用する資源の概要

データフラッシュドライバは処理を実施するに当たり、タスクの起床要求と割込みハンドラを利用します。タスクの起床要求はデバイスドライバのシステムコールを発行時、割込みハンドラはサービス関数を呼び出すと定義されます。

使用する資源	名称	優先度
割込みハンドラ	fcu_frdyi_hdr	割込みハンドラの割込み優先度は DFL_CFG_INT_PRIORITY

注：DFL_CFG_INT_PRIORITY は次節を参照

5.9.2 割込みハンドラ (fcu_frdyi_hdr)

フラッシュコントロールユニット (FCU) の割込みハンドラです。割込みの発生により起床待ちのタスクを起床します。

項目	ターゲット	値
割込み番号	TB-RX65N	23 (FCUのFRDYI)
割込み優先度	—	DFL_CFG_INT_PRIORITY
割込みハンドラ起動アドレス	—	fcu_frdyi_hdr : FRDI (フラッシュレディ割込み)

シリアルドライバは処理を実施するに当たり、デバイス名毎に1つのタスク、1つのイベントフラグ、4つの割込みハンドラを利用します。サービス関数のAPIを呼び出すと生成または定義されます。

5.10 データフラッシュドライバのコンフィグレーション

5.10.1 データフラッシュドライバのコンフィグレーション・ファイル

データ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¥df¥sysdepend¥ターゲット¥df¥df_l_config.h です。

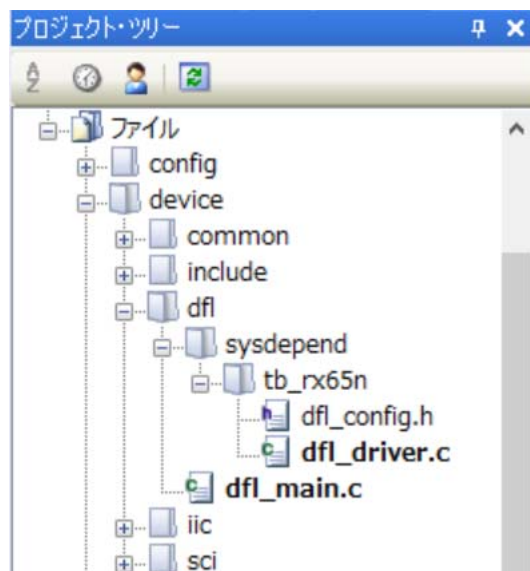


図5.1 データフラッシュドライバのコンフィグレーション・ファイル

5.10.2 割込みハンドラの割込み優先度

● DFL_CFG_INT_PRIORITY

内蔵周辺機能のフラッシュコントロールユニット（FCU）の割込み優先度です。

システムの都合に応じて、1 ～ MAX_INT_PRI（最高外部割込み優先度）の範囲で変更できます。

TIM_INT_PRI（システムタイマの割込み優先度）よりも低く設定することを推奨します。デフォルトでは 1 に設定されています。

```
/* data flash interrupt priority level. */  
#define DFL_CFG_INT_PRIORITY (1)
```

5.11 スタックサイズ

5.11.1 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。

TB_RX65N_DRV.cal - Call Walker						
File Edit View Tools Help						
Standard Library Version : None						
Symbol	Attributes	Address	Size	Stack size	Source	
_dfl_open (52)	S	0xffe02f08	3	4	dfl_main.obj	
_dfl_abort (4)	S	0xffe02f4d	3	4	dfl_main.obj	
_dfl_close (52)	S	0xffe02f0b	3	4	dfl_main.obj	
_fcu_frdyi_hdr (76)	S	0xffe02fe9	9	4	dfl_driver.obj	
_dfl_event (108)	S	0xffe02f50	85	12	dfl_main.obj	
_dfl_wait (4)	S	0xffe02f4a	3	4	dfl_main.obj	
_dfl_exec (128)	S	0xffe02f0e	60	16	dfl_main.obj	

図5.2 データフラッシュデバイスドライバのスタック見積もり結果

5.11.2 割り込みハンドラのスタックサイズ

図5.2の例では、fcu_frdyi_hdrが割り込みハンドラです。図5.2の例では76バイトですが、この値がデータフラッシュドライバのコンフィグレーションで指定した DFL_CFG_INT_PRIORITY 割り込み優先度で実行される割り込みハンドラのスタックサイズとなります。この数値を使って例外スタックのサイズを算出してください。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

5.11.3 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（dfl_open、dfl_close、dfl_exec、dfl_wait、dfl_abort、dfl_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。詳しくはターゲット毎の構築仕様書の第5章を参照してください。

5.12 サンプルプログラム

本実装ではデータフラッシュの最初の2ブロックに対して書き込み・読み込み及びブランクチェックとブロックイレースを行うサンプルプログラムを準備しています。

なお、サンプルプログラムでは「CLANGSPEC」のマクロ定義を利用しています。「CLANGSPEC」のマクロ定義に関しては各ターゲットの構築仕様書を参照してください。

usermain関数

```
#include <string.h>
#include <tk/tkernel.h>
#include "dev_sci.h"

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

    dflDrvEntry( ); // Entry Data Flash Driver
    t_ctsk.tskatr = TA_HLNG | TA_DSNAME; // Set Task Attribute
}
```

```

t_ctsk.stksz = 1024; // Set Task Stack Size
t_ctsk.itskpri = 10; // Set Task Priority
t_ctsk.task = flash_tsk; // Set Task Start Address
strcpy( t_ctsk.dsname, "D flash" ); // Set Debugger Support Name
if( (objid = tk_cre_tsk( &t_ctsk )) <= E_OK ) // Create Data Flash Task
    goto Err;
if( tk_sta_tsk( objid, 0 ) < E_OK ) // Start Data Flash Task
    goto Err;
tk_slp_tsk( TMO_FEVR ); // Sleep
Err:
while( 1 ) ;
}

```

usermain関数では、サービス関数を使ってデータフラッシュドライバを登録し、データフラッシュへの書き込みや読み込み、ブランクチェックやブロックイレースを行うタスク（flash_tsk）を生成・起動し、自身はtk_slp_tskで待ち状態となります。

フラッシュタスク（flash_tsk）

```

VB buf[64];
EXPORT void flash_tsk(INT stacd, void *exinf)
{
ID dd, devid;
SZ asize;
INT i;

if( ( dd = tk_opn_dev( DFL_DEVMN, TD_UPDATE | TD_EXCL ) ) <= E_OK )
    goto Err;
devid = tk_ref_dev( DFL_DEVMN, NULL ); // Get Device ID

for( i=0 ; i<64 ; i++ ) // Make Write Data
    buf[i] = i + 32;
tm_putstring("Flash Programming at 0x100000-0x10003F\n\r");
for( i=0 ; i<16 ; i++ ) // Output Write Data
    tm_printf(" %.2x = %c %.2x = %c %.2x = %c %.2x = %c\n\r",
        buf[i],buf[i], buf[i+16],buf[i+16], buf[i+32],buf[i+32], buf[i+48],buf[i+48] );

if( tk_evt_dev( devid, TDV_BLANKCHECK64, (void *)0 ) < E_OK ) // Blank Check
    tk_evt_dev( devid, TDV_BLOCKERASE64, (void *)0 ); // 64 Byte Erase
tk_swri_dev( dd, 0, buf, sizeof(buf)/sizeof(UW), &asize ); // 64 Byte Program

tm_putstring("Flash Reading at 0x100000-0x10003F\n\r");
tk_srea_dev( dd, 0, buf, sizeof(buf)/sizeof(UW), &asize ); // 64 Byte Read
for( i=0 ; i<16 ; i++ ) // Output Write Data
    tm_printf(" %.2x = %c %.2x = %c %.2x = %c %.2x = %c\n\r",
        buf[i],buf[i], buf[i+16],buf[i+16], buf[i+32],buf[i+32], buf[i+48],buf[i+48] );

for( i=0 ; i<64 ; i++ ) // Make Write Data
    buf[i] = i + 64;
tm_putstring("Flash Programming at 0x100040-0x10007B\n\r");
for( i=0 ; i<16 ; i++ ) // Output Write Data
    tm_printf(" %.2x = %c %.2x = %c %.2x = %c %.2x = %c\n\r",
        buf[i],buf[i], buf[i+16],buf[i+16], buf[i+32],buf[i+32], buf[i+48],buf[i+48] );

if( tk_evt_dev( devid, TDV_BLANKCHECK64, (void *)64 ) < E_OK ) // Blank Check
    tk_evt_dev( devid, TDV_BLOCKERASE64, (void *)64 ); // 64 Byte Erase
for( i=0 ; i<15 ; i++ ) // 4 * 15 Byte Program
    tk_swri_dev( dd, 64+sizeof(UW)*i, &buf[1+sizeof(UW)*i], 1, &asize );

tm_putstring("Flash Reading at 0x100040-0x10007B\n\r");
tk_srea_dev( dd, 64, &buf[1], sizeof(buf)/sizeof(UW)-1, &asize ); // 60 Byte Read
for( i=0 ; i<16 ; i++ ) // Output Write Data

```

```

        tm_printf(" %#.2x = %c  %#.2x = %c  %#.2x = %c  %#.2x = %c\n\r",
        buf[i],buf[i], buf[i+16],buf[i+16], buf[i+32],buf[i+32], buf[i+48],buf[i+48] );

        if( tk_evt_dev( devid, TDV_BLANKCHECK4, (void *)124 ) == E_OK )           // Blank Check
            tm_putstring("0xi0007C-0xi0007F is Blank !\n\r");
        tk_cls_dev( dd, 0 );
Err:
        tk_ext_tsk();                                                           // Exit
}

```

第1段落： tk_opn_devでデータフラッシュドライバをオープンします。引数は基本的に本サンプルと同じものを指定してください。返却値のデバイスディスクリプタは変数等に保存します。その後、tk_ref_devでデバイスIDを取得し、これも変数等に保存します。

第2段落： データフラッシュに書き込むデータをバッファに作成し、ターミナルに表示します。

第3段落： tk_evt_devでデータフラッシュの最初の64バイト（1ブロック）がブランクであるかを調べます。ブランクでない場合、tk_evt_devでデータフラッシュの最初の64バイトをイレースします。その後、tk_swri_devでデータフラッシュの最初の64バイトに書き込みを行います。

第4段落： tk_srea_devでデータフラッシュの最初の64バイトを読み込み、ターミナルに表示します。

第5段落： データフラッシュに書き込むデータをバッファに作成し、ターミナルに表示します。

第6段落： tk_evt_devでデータフラッシュの64番地からの64バイトがブランクであるかを調べます。ブランクでない場合、tk_evt_devでデータフラッシュの64番地からの64バイトをイレースします。その後、tk_swri_devでデータフラッシュの64番地からの60バイトに対して、4バイトずつ書き込みを行います。なお、書き込むデータは4バイトアライメントではない番地から4バイト単位で書き込んでいます。これは書き込むデータにはアライメントの制約がないことを意味しています。

第7段落： tk_srea_devでデータフラッシュの64番地からの60バイトを読み込み、ターミナルに表示します。

第8段落： データフラッシュの2ブロック目の最後の4バイト（124番地からの4バイト）、すなわち書き込みを行っていない4バイトがブランクであることを確認します。

実行結果

```

Flash Programming at 0x100000-0x10003F
0x20 = 0x30 = 0 0x40 = @ 0x50 = P
0x21 = ! 0x31 = 1 0x41 = A 0x51 = Q
0x22 = ~ 0x32 = 2 0x42 = B 0x52 = R
0x23 = # 0x33 = 3 0x43 = C 0x53 = S
0x24 = $ 0x34 = 4 0x44 = D 0x54 = T
0x25 = % 0x35 = 5 0x45 = E 0x55 = U
0x26 = & 0x36 = 6 0x46 = F 0x56 = V
0x27 = ' 0x37 = 7 0x47 = G 0x57 = W
0x28 = ( 0x38 = 8 0x48 = H 0x58 = X
0x29 = ) 0x39 = 9 0x49 = I 0x59 = Y
0x2a = * 0x3a = : 0x4a = J 0x5a = Z
0x2b = + 0x3b = ; 0x4b = K 0x5b = [
0x2c = , 0x3c = < 0x4c = L 0x5c = ¥
0x2d = - 0x3d = = 0x4d = M 0x5d = ]
0x2e = . 0x3e = > 0x4e = N 0x5e = ^
0x2f = / 0x3f = ? 0x4f = O 0x5f = _
Flash Reading at 0x100000-0x10003F

```

```

0x20 = 0x30 = 0 0x40 = @ 0x50 = P
0x21 = ! 0x31 = 1 0x41 = A 0x51 = Q
0x22 = " 0x32 = 2 0x42 = B 0x52 = R
0x23 = # 0x33 = 3 0x43 = C 0x53 = S
0x24 = $ 0x34 = 4 0x44 = D 0x54 = T
0x25 = % 0x35 = 5 0x45 = E 0x55 = U
0x26 = & 0x36 = 6 0x46 = F 0x56 = V
0x27 = ' 0x37 = 7 0x47 = G 0x57 = W
0x28 = ( 0x38 = 8 0x48 = H 0x58 = X
0x29 = ) 0x39 = 9 0x49 = I 0x59 = Y
0x2a = * 0x3a = : 0x4a = J 0x5a = Z
0x2b = + 0x3b = ; 0x4b = K 0x5b = [
0x2c = , 0x3c = < 0x4c = L 0x5c = ¥
0x2d = - 0x3d = = 0x4d = M 0x5d = ]
0x2e = . 0x3e = > 0x4e = N 0x5e = ^
0x2f = / 0x3f = ? 0x4f = O 0x5f = _

```

Flash Programming at 0x100040-0x10007B

```

0x40 = @ 0x50 = P 0x60 = ` 0x70 = p
0x41 = A 0x51 = Q 0x61 = a 0x71 = q
0x42 = B 0x52 = R 0x62 = b 0x72 = r
0x43 = C 0x53 = S 0x63 = c 0x73 = s
0x44 = D 0x54 = T 0x64 = d 0x74 = t
0x45 = E 0x55 = U 0x65 = e 0x75 = u
0x46 = F 0x56 = V 0x66 = f 0x76 = v
0x47 = G 0x57 = W 0x67 = g 0x77 = w
0x48 = H 0x58 = X 0x68 = h 0x78 = x
0x49 = I 0x59 = Y 0x69 = i 0x79 = y
0x4a = J 0x5a = Z 0x6a = j 0x7a = z
0x4b = K 0x5b = [ 0x6b = k 0x7b = {
0x4c = L 0x5c = ¥ 0x6c = l 0x7c = |
0x4d = M 0x5d = ] 0x6d = m 0x7d = }
0x4e = N 0x5e = ^ 0x6e = n 0x7e = ~
0x4f = O 0x5f = _ 0x6f = o 0x7f =

```

Flash Reading at 0x100040-0x10007B

```

0x40 = @ 0x50 = P 0x60 = ` 0x70 = p
0x41 = A 0x51 = Q 0x61 = a 0x71 = q
0x42 = B 0x52 = R 0x62 = b 0x72 = r
0x43 = C 0x53 = S 0x63 = c 0x73 = s
0x44 = D 0x54 = T 0x64 = d 0x74 = t
0x45 = E 0x55 = U 0x65 = e 0x75 = u
0x46 = F 0x56 = V 0x66 = f 0x76 = v
0x47 = G 0x57 = W 0x67 = g 0x77 = w
0x48 = H 0x58 = X 0x68 = h 0x78 = x
0x49 = I 0x59 = Y 0x69 = i 0x79 = y
0x4a = J 0x5a = Z 0x6a = j 0x7a = z
0x4b = K 0x5b = [ 0x6b = k 0x7b = {
0x4c = L 0x5c = ¥ 0x6c = l 0x7c = |
0x4d = M 0x5d = ] 0x6d = m 0x7d = }
0x4e = N 0x5e = ^ 0x6e = n 0x7e = ~
0x4f = O 0x5f = _ 0x6f = o 0x7f =

```

0x10007C-0x10007F is Blank !

6. 本実装に関する考察

6.1 排他制御

デバイス処理の実施に当たり、デバイスからの応答待ちとなるデバイスドライバは待ち状態となるためにタスクとして作成しなければなりません（データフラッシュドライバに関しては6.2節を参照）。

μ T-Kernel仕様において、タスクとして処理を行うデバイスドライバは汎用デバイスドライバとして作成することが推奨されており、デバイスの処理を行いながら複数のタスクからの処理要求を受け付けなければならないため、タスク内部で排他制御が必要となります。この排他制御に関しても、 μ T-Kernel仕様では高速ロック機能を使用することを推奨しています。ただし、高速ロック機能を使用すると、内部でセマフォ資源が使われてしまいます。

本実装では、可能な限り μ T-Kernel仕様が持つ資源を無駄に使用しないことを設計方針の最優先項目に掲げています。 μ T-Kernel仕様が推奨しているデバイスドライバのインタフェースライブラリを使用しないことも、この設計方針が関係しています（インタフェースライブラリを使用するとイベントフラグ、セマフォ、高速ロック機能等が使われてしまいます）。そこで**本実装では以下に示す独自の排他制御機構を使っています**。

```
/*
 *      drv_lock.c
 *
 *      Driver Lock
 */

#include <tk/tkernel.h>

EXPORT void drv_lock(INT mode, W *lock)
{
W work;
    if( mode == TRUE ) {
        work = TRUE;
        while( __xchg( lock, &work ), work == TRUE )
            tk_dly_tsk( 1 );
    }
    else {
        work = FALSE;
        __xchg( lock, &work );
    }
}
```

特徴は2つあり、1つは資源の競合チェックにCC-RX固有の組み込み関数（**青字の部分**）を使っていること、もう1つは資源が競合した際の資源待ちにtk_dly_tskシステムコール（**緑字の部分**）を使っていることです。デバイスドライバはマイコン固有の処理であるため、CC-RX固有の機能を利用したことに関しては大きな問題となりません。もし、他の処理系を使用したならば、組み込み関数の部分をアセンブリ言語で記述すれば実装できるからです。

これに対して、資源が競合した際の資源待ちにtk_dly_tskシステムコールを使用したことには大きな問題があります。それは資源を獲得したタスクが資源を解放しても、待ち状態となったタスクはタイム

アウト（今回の実装では1ms）が発生しないと待ち解除とならないことです。これは**リアルタイム性が欠如していることを意味します**。リアルタイム性が欠如することを理解したうえで、本実装とした理由は以下の通りです。

現状サポートしているデバイスドライバは、複数のタスクから利用する可能性が低いこと、非同期の読み込み・書き込み要求を発行する可能性が低いことにあります。つまり、もともと**資源の競合が発生する可能性が低い**のです。競合が発生しないのであれば、どのような待ち条件でタスクが待ち状態となっても問題ありません。このような理由により、本実装ではtk_dly_tskを使用した1msのタイムアウト機能を使っています。**もし、本実装を製品に移植する場合は、本件に関する検討をお願い致します**。drv_lock関数を高速ロック機能に切り換える等で対策が可能です。詳しくはお問い合わせください。

6.2 データフラッシュドライバ

データフラッシュへのライト、ブランクチェック、ブロックイレースの処理に関しては待ち時間が必要となります。 μ T-Kernel仕様において、待ち時間を有するデバイスドライバはタスクで処理を行う汎用デバイスドライバとして作成することを推奨しています。しかしながら、**本実装においてデータフラッシュドライバは汎用デバイスドライバではなく、単純デバイスドライバとして実装されています**。

その理由は、汎用デバイスドライバとしてインプリメントを行うと数多くのカーネルオブジェクト資源が使用されてしまうからです。データフラッシュの利用用途を考えた場合、特定のタスクからのみの利用となることが想定されます。一方、単純デバイスドライバであれば排他制御や非同期構造を考えなくて良いため、実装も簡単なものにできます。

また、データフラッシュドライバでは、タスクの起床要求を利用し、割込みハンドラとのインタフェースを行っています。このため、タスクの起床要求がキューイングされた状態でデータフラッシュドライバのシステムコールを発行すると E_OBJ のエラーとなります。これもデータフラッシュドライバ特有の問題点となります。**もし、本実装を製品に移植する場合は、本件に関する検討をお願い致します**。これらの問題点は汎用デバイスドライバとして実装を行えばすべて解決されます。詳しくはお問い合わせください。

7. 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をお願い致します。

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