ICS library
Function manual

Index

1. はじめに

1.1. はじめに

本ドキュメントは、ICS 用関数マニュアルです。

1.2. 注意事項

1. この資料に記載されたすべての情報は、本資料発行時点の物であり、予告なく変更することがあります。弊社製品のご購入およびご使用にあたりましては、必ず最新の資料を参照していただけるようお願いいたします。
2. 本資料に記載された弊社製品、技術情報の仕様に関連し発生した第三者の特許権、著作権、その他の知的財産権の侵害に関し、弊社は一切その責任を負いません。弊社は、本資料によって弊社または第三者の特許権、著作権、その他の知的財産権を許諾するものではありません。
3. 弊社製品の複製等を行わないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、インバータ製品の動作例、応用例を説明するための物です。お客様の機器の設計、実験において、回路、ソフトウェアおよびこれに関連する情報を使用する場合には、お客様の責任において行ってください。これらの仕様に起因して、お客様または、第三者に生じた損害に関し、弊社は一切その責任を負いません。
5. 輸出に際しては、「外国為替および外国貿易法」その他、輸出関連法令を順守し、かかる法令の定めるところにより必要な手続きを行ってください。本資料に記載されている弊社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的、その他軍事用途の目的で使用しないでください。また、弊社製品および技術を国内外の法令および規制により製造・使用・販売を禁止されている機器に使用することはできません。
6. 本資料に記載されている情報は、正確を期すために慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りによる損害がお客様に生じた場合においても、弊社は、一切その責任をおいません。
7. 本製品は、実験用として設計されています。特に、交通システム（自動車、電車、船舶）、交通用信号機器、防災・防犯装置、各種安全機器、医療機器、生命維持機器、航空機器、原子力制御機器などに使用なさらないようお願いいたします。
8. 本資料に記載された弊社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他、諸条件につきましては、弊社提案範囲内でご使用ください。
9. 弊社は、弊社製品の品質および信頼性の向上に努めておりますが、ある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。弊社製品は、耐放射線設計については、行っておりません。弊社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせない様、お客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全対策およびエージング処理等、機器またはシステムとしての保証をお願いいたします。特にマイコンソフトウェアは、単独での検証は困難なため、お客様が製造、実験なさる最終の機器・システムとしての安全検証をお願いいたします。
9. 本資料の全部または一部を弊社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。

ICS は、ルネサスエレクトロニクス株式会社の製品です。株式会社デスクトップラボは、ICS の使用方法、ライブラリ開発受託などの ICS 関連サポート業務を行っています。

2. ライブラリの基本仕様・動作

2.1. 通信規約・ソースコード

ICS のライブラリソースコードや詳細な通信プロトコルは、非公開となっております。ここでは、使用するに当たって重要な項目について説明します。

2.2. データ転送間隔の制限

ICS では、ユーザー側の CPU からデータを転送するため、後述の ics_watchpoint()関数を呼び出します。この関数を呼び出し方に、以下の制約があります。

最小　250us （標準 1Mbps の場合）
最大　5ms

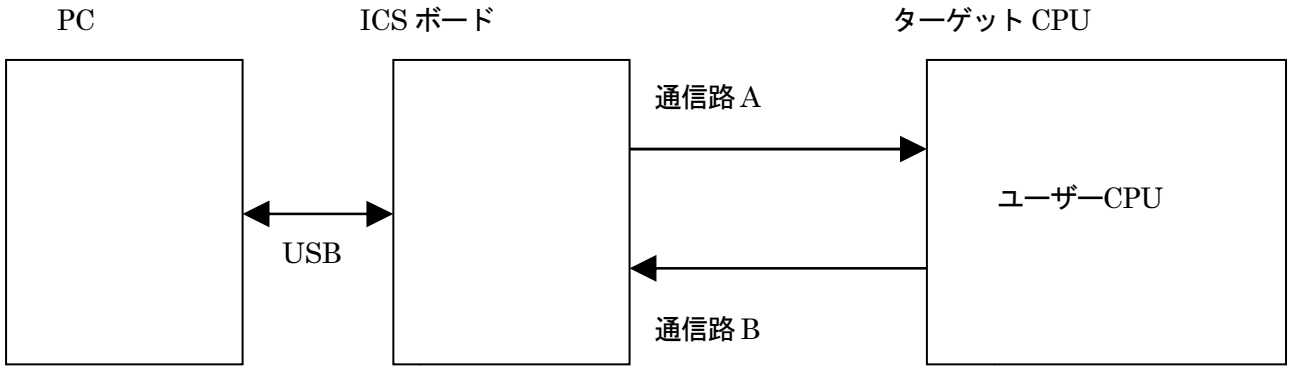

図 1　ICS システムの通信路

本 ICS には、データ転送間隔の制限があります。本制約は、図 1 の通信路 B の通信レート上限により発生します。後述のデータ転送関数　ics_watchpoint()関数を呼ぶ度に、固定長のデータがターゲットから ICS ボードに送られます。このデータ転送時間、ターゲット側の割り込みなどによる時間の遅れ、ICS ボード側のオーバーヘッドなどから、転送間隔の最短時間制限が発生します。この時間以下になると、転送がうまく行われず、ICS が正常な動作をしなくなることがあります。

ICS の転送間隔の最短時間制限は、転送速度に大きく依存します。例として通信速度が 1Mbps の場合、最短時間制約は 250us となっています。そのほかの通信速度については、各ライブラリ部分の記載を参照してください。また、ics_watchpoint()関数の最大呼び出し時間間隔の制限もあり、ライブラリにかかわらず 5ms となっています。

2.3. 16bit / 32bit ライブラリの違い

ICS には、１６ビットライブラリと３２ビットライブラリとが存在しています。このライブラリの違いは、CPU のビット長ではなく波形表示でサポートする最大ビット長で定義しています。通常、SH , RX, RZ, RH850 などの 32bit CPU では３２ビットライブラリを、 RL78 などの 16bit CPU では、１６ビットライブラリを提供しています。これらの差について説明します。

【データ転送間隔の制限】の部分で述べた時間間隔以上で、１回に転送されるデータは 16byte 分です。従って、16bit 変数ならば 8ch 分、32bit 変数ならば 4ch 分しか送ることが出来ません。32bit を 8ch 分送信するためには、2 回に分けて送ります。

１） １６ビットライブラリの動作
数値表示に関しては、８／１６／３２ビットのすべて型に対して動作します。しかしながら、波形表示に関しては型の制約があります。８ビットデータならば変数の型に応じて１６ビットに拡張し、１６ビットならばそのままで 8ｃｈ分を１回で転送します。３２ビットデータは転送することはできません。

２）３２ビットライブラリの動作
数値表示に関しては、８／１６／３２ビットのすべて型に対して動作します。ics_watchpoint()関数が呼ばれると、波形表示に関しては、一度に指定された８ｃｈ分の８ビット、１６ビット、３２ビットデータを取り込みます。さらに 4ch 分のデータを転送します。次に ics_watchpoint()関数が呼ばれた時、データを取り込まず、未転送の残り 4ch 分のデータを転送します。
つまり、３２ビットライブラリの場合には、２回の ics_watchpoint()関数により、１回の８ｃｈ分の転送が行われます。

標準ライブラリでは、16bitCPU では、16bit ライブラリをサポートし、32bit CPU では、32bit ライブラリをサポートします。カスタムライブラリーでは、16bit CPU において 32bit ライブラリをサポートしたり、32bitCPU で１６ビットライブラリを使用することも可能です。このような非標準ライブラリを必要とする場合には、別途ご相談ください。

	メリット	デメリット
１６ビットライブラリ	波形情報更新間隔が短い	32bit の波形表示ができない
３２ビットライブラリ	32bit の波形表示が可能	波形情報更新間隔が 16bit の２倍

2.4. 数値表示ウィンドウ使用時の制約

ICS では、数値表示と波形表示とを１本の通信路で共用しているため、数値表示と波形表示とを同時に行う場合、波形表示の制約が発生します。

波形表示を行っており数値表示が行われていない場合、波形データは毎回送信されるので、データはそのまま表示されます。しかしながら、数値表示と波形表示とが同時に行われている場合、数十 ms に１サンプリング分だけ波形が更新されず、表示される波形の一部が平らになる場合があります。データ測定をする場合など、このような状況が適当でない場合、一時的に、ICS のオートリフレッシュ機能を停止してください。

2.5. ファイル構成・ライブラリ

ICS ライブラリは以下のような２つのファイルの構成になっています。

ヘッダファイル
ics_<CPUNAME>.h
ics_<CPUNAME>.obj

通常関数として、
void ics_init(void* addr, char unitpin, char level);
void ics_watchpoint(void);
が提供されています。
ただし、ＣＰＵによっては、一部名称が異なる場合があります。

※注意１
CPU によって、使用する割り込みが異なります:
使用する UART ポートが異なっても、ICS 側の割り込み処理関数名は同じ名称です。ご注意ください。

※注意２
無償配布のライブラリーでは、DTC は標準アドレスモードを使用します。DTC のベクターテーブルは、RAM 上に配置する必要があります。
DTC においてショートアドレスモードを使用する場合、ビックエンディアンを使用する場合、ＲＯＭ上にＤＴＣテーブルを配置する場合など、標準の仕様と異なる場合には、無償ライブラリーは使用できません。

※注意３
標準ライブラリのコンパイラ・アセンブラ・リンカーのオプションスイッチは、プロジェクトをデフォルトで生成した状態を利用しています。お客様のプロジェクトにおいてご使用になるメモリーモデル、エンディアン、レジスターモード、などを変更していた場合、ICS ライブラリの一部、もしくは、全部の機能が動作しない場合があります。お使いになる予定のコンパイラースイッチの状態をご確認の上、ご使用になってください。

3. 使用資源・ライブラリの説明

3.1. RX62T シリーズ

3.1.1. RX62T 使用資源

CPU 名	RX62T シリーズ共通
開発環境	CubeSuite+ Ver.2.01.00
Library version	Ver.2.0 ～ Ver.2.1
通信レート	通信レート = $\frac{PCLKB}{48}[Mbps]$ 標準通信レート　PCLKB = 48MHz　の場合 1Mbps
ステイタス	SCI0, SCI1, SCI2 サポート
ライブラリ種別	32bit ライブラリ
ライブラリ	ics_RX62T.obj
ヘッダファイル	ics_RX62T.h

CPU 使用リソース	対応 ICS	サポート変数タイプ
・内部使用リソース SCI0 　INT　SCI0 RXI 　INT　SCI0 TXI 　DTC　INT216　(TXI0) 　ICU.DTCER[216].BIT.DTCE 　SCI0　全て 　DTC　全て 　ICU.IPR[0x80].BYTE 　ICU.IER[0x1A].BIT.IEN6 　ICU.IER[0x1A].BIT.IEN7 　ICU.IER[0x1B].BIT.IEN0 　ICU.IER[0x1B].BIT.IEN1 　SYSTEM.MSTPCRA.BIT.B28 　SYSTEM.MSTPCRA.BIT.B31 　SYSTEM.MSTPCRB.BIT.B31 　PORTB.ICR.BIT.B1 = 1 　外部ピン 　PB2: TXD0 　PB1: RXD0 SCI1 　INT　SCI1 RXI 　INT　SCI1 TXI 　DTC　INT220　(TXI1) 　ICU.DTCER[220].BIT.DTCE	○対応 ICS ハード 　２種類の対応が可能です。 標準版 H/W model　1 H/W Ver.　1 S/W Ver.　1.22　以降 または、 H/W model　4 H/W Ver.　1 S/W Ver.　1.22　以降 以上は、ICS のハードを接続した状態で、ICS アプリの 　Help -> About で表示可能 ICS PC ソフト Ver. 2.5.0.0　以降	数値表示・設定 　8bit 符号なし　整数型 　8bit 符号あり　整数型 　16bit 符号なし　整数型 　16bit 符号あり　整数型 　32bit 符号なし　整数型 　32bit 符号あり　整数型 　32bit IEEE754 浮動小数点 　8bit　BOOL 型 　8bit　LOGIC 型 波形表示 　8bit 符号なし　整数型 　8bit 符号あり　整数型 　16bit 符号なし　整数型 　16bit 符号あり　整数型 　32bit 符号なし　整数型 　32bit 符号あり　整数型 　32bit IEEE754 浮動小数点

SCI1　全て DTC　全て ICU.IPR[0x81].BYTE ICU.IER[0x1B].BIT.IEN2 ICU.IER[0x1B].BIT.IEN3 ICU.IER[0x1B].BIT.IEN4 ICU.IER[0x1B].BIT.IEN5 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRA.BIT.B31 SYSTEM.MSTPCRB.BIT.B30 PORTDICR.BIT.B5= 1 外部ピン PD3 TXD1 PD5: RXD1 SCI2 (PB5, PB6) INT　SCI2RXI INT　SCI2TXI DTC　INT224　(TXI2) ICU.DTCER[224].BIT.DTCE SCI2　全て DTC　全て ICU.IPR[0x82].BYTE ICU.IER[0x1B].BIT.IEN6 ICU.IER[0x1B].BIT.IEN7 ICU.IER[0x1C].BIT.IEN0 ICU.IER[0x1C].BIT.IEN1 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRA.BIT.B31 SYSTEM.MSTPCRB.BIT.B29 PORTB.ICR.BIT.B6 = 1 IOPORT.PFFSCI.BIT.SCI2S 外部ピン PB5: TXD2 PB6: RXD2 SCI2 (P81, P80) INT　SCI2RXI INT　SCI2TXI DTC　INT224　(TXI2) ICU.DTCER[224].BIT.DTCE SCI2　全て DTC　全て		

ICU.IPR[0x82].BYTE ICU.IER[0x1B].BIT.IEN6 ICU.IER[0x1B].BIT.IEN7 ICU.IER[0x1C].BIT.IEN0 ICU.IER[0x1C].BIT.IEN1 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRA.BIT.B31 SYSTEM.MSTPCRB.BIT.B29 PORT8.ICR.BIT.B0 = 1 IOPORT.PFFSCI.BIT.SCI2S 外部ピン P81: TXD2 P80: RXD2		

3.1.2. RX62T 関数説明

Lib Ver.2.0 ～ Ver.2.1 on CubeSuite+ Ver.2.01.00
初期化関数の呼び出し void ics_init(void* addr, char port, char level);
本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。 第１パラメータ DTC で使用する DTC のベクトルテーブルの先頭アドレスを指定してください。ics_init()関数を呼び出す前にユーザーが DTC ベクトルテーブルを確保する必要があります。このアドレスの下位 12bit は０である必要があります。 第２パラメータ SCI のポート番号や、SCI の使用するピンを設定します。このパラメータは、ICS_<CPUNAME>.h 内で定義されている文字列を使用してください。 第３パラメータ ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。 最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。 SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。
転送関数の呼び出し void ics_watchpoint(void);
本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。 本関数は、ＰＣで指定された変数のデータを読み出し、ＤＴＣ用の転送バッファーにコピーします。 この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてくだ

さい。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 70[us]$$

通信速度が 1Mbps の際、上の式に数値を代入すると。

$$最小通信間隔 = 1/(1[Mbps]) \times 180 + 70[us] = 250[us]$$

※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。

使用割り込み関数

下記の割り込みベクトルを使用しているため、ユーザーソフトの割り込みベクトルに下記の関数を登録してください。
ルネサス標準のコンパイラで自動的に生成されるプロジェクトを使用する場合、intprg.c という割り込み処理を記載したファイルに追記してください。

たとえば、SCI0 を使用する場合、下記のように記述してください。

SCI0 の場合

```
// SCI0 ERI0
void Excep_SCI0_ERI0(void){ics_int_sci_eri();}
// SCI0 ERI0
void Excep_SCI0_RXI0(void){ics_int_sci_rxi();}
```

SCI1 の場合

```
// SCI1 ERI1
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
// SCI1 RXI1
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
```

SCI2 の場合

```
// SCI2 ERI2
void Excep_SCI2_ERI2(void){ ics_int_sci_eri(); }
// SCI2 RXI2
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
```

3.1.3. RX62T 関数使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）開発環境に BDTCTBL セクションを確保する。

BDTCTBL のセクションを下位 12bit が０になるような RAM 上のアドレスに割り当てます。このアドレスを開発環境に設定してします。RX62T の RAM が最小のモデルはサイズが 8kByte なので、設定できるアドレスは、0x0000 もしくは、0x1000 となります。RAM が 16kByte のモデルの場合、設定できるアドレスは、0x0000, 0x1000, 0x2000, 0x3000 の３点となります。ここでは、0x0000 からに設定することにします。
E1 などのエミュレータを使用する場合、ユーザーRAM 領域もしくは、DTC テーブルの領域と、E1 エミュレータの使用領域とが重ならないようにしてください。正常に動作しないことがあります。

２）ユーザープログラムに、DTC テーブルを定義する。

ICS_sample.c の先頭に DTC テーブルの変数である unsigned long dtc_table[256]を定義する。

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section
```

３）ics_init()を下記のように呼び出す。

初期化関数 ics_init((void*)dtc_table, ICS_SCI2_PB5_PB6, 6) を初期化部分に入れてください。
第１パラメータは、１）で確保した BDTCTBL の先頭アドレスを入力してください。
第 2 パラメータは、使用するポートを ICS_<CPUNAME>.h から選択してください。
第 3 パラメータは、ICS で使用する割り込みレベルを設定してください。通常は、キャリア割り込みより低い優先順位に設定します。

---------- List 1 main.c ----------

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section

void main(void)
{
      ics_init((void*)dtc_table, ICS_SCI2_PB5_PB6, 6);      /* Interrupt level 6            */
      while(1)
      {    nop();    }
}
```

４）ics_watchpoint()関数の組込み
このソフトでは、メインルーチンで ics_watchpoint()関数を呼び出していますが、通常は、キャリア割り込みの内部で呼び出します。

また、この関数は、250us 以上, 5ms 以下の間隔で呼び出すようにしてください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List2 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2　ics_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics_watchpoint();
        }
}
```

５）intprg.c の修正

intprg.c の中の　RXI, ERI の割り込み関数から　ics_int_sci_eri(),　ics_int_sci_rxi()関数を呼び出すようにしてください。

SCI0 の場合

```
// SCI0 ERI0
void Excep_SCI0_ERI0(void){ ics_int_sci_eri(); }
// SCI0 RXI0
void Excep_SCI0_RXI0(void){ ics_int_sci_rxi(); }
```

SCI1 の場合

```
// SCI1 ERI1
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
// SCI1 RXI1
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
```

SCI2 の場合

```
// SCI2 ERI2
void Excep_SCI2_ERI2(void){ ics_int_sci_eri(); }
// SCI2 RXI2
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
```

3.1.4. RX62T ICS クロック

本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを PCLKB=96MHz で使用する必要があります。

ICS ボード上のクロック周波数 = (PCLKB / 6) MHz

例： PCLKB = 50MHz の場合、 ICS CLOCK = 50/6 = 8.333MHz
PCLKB = 48MHz の場合、 ICS CLOCK = 48/6 = 8.000MHz

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項 :

ICS W1002 の場合（水晶発振器のソケットがないタイプ）
クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS W1003 の場合（水晶発振器のソケットがあるタイプ）
8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
可変クロックを内蔵しているため、PC 側からの操作が可能となります。

3.2. RX111 シリーズ

3.2.1. RX111 使用資源

CPU 名	RX111 シリーズ共通
開発環境	CubeSuite+ Ver.2.01.00
Library version	Ver2.0
通信レート	$通信レート = \frac{PCLKB}{32}[Mbps]$ 標準通信レート　PCLKB = 32MHz　の場合 1Mbps
ステイタス	SCI1, SCI5, SCI12 サポート
ライブラリ種別	32bit ライブラリ
ライブラリ	ics_RX111.obj
ヘッダファイル	ics_RX111.h

CPU 使用リソース	対応 ICS	サポート変数タイプ
・内部使用リソース SCI1 　INT　SCI1 RXI 　DTC　INT220　(TXI1) 　ICU.DTCER[220].BIT.DTCE 　SCI1　全て 　DTC　全て 　ICU.IPR[218].BYTE 　ICU.IER[0x1B].BIT.IEN2 　ICU.IER[0x1B].BIT.IEN3 　ICU.IER[0x1B].BIT.IEN4 　SYSTEM.MSTPCRB.BIT.B30 　SYSTEM.MSTPCRB.BIT.B26 　SYSTEM.MSTPCRB.BIT.B4 　MPC.PWPR.BIT.B0WI 　MPC.PWPR.BIT.PFSWE 　・Ext pin, PC7:TXD1, PC6:RXD1 　　MPC.PC6PFS.BIT.PSEL 　　MPC.PC7PFS.BIT.PSEL 　　PORTC.PMR.BIT.B7 　　PORTC.PMR.BIT.B6 　・Ext pin, P26:TXD1, P30:RXD1 　　MPC.P26PFS.BIT.PSEL 　　MPC.P30PFS.BIT.PSEL 　　PORT2.PMR.BIT.B6 　　PORT3.PMR.BIT.B0 　・Ext pin P16:RXD1, P15:RXD1 　　MPC.P16PFS.BIT.PSEL	○対応 ICS ハード 　２種類の対応が可能です。 標準版 H/W model　1 H/W Ver.　1 S/W Ver.　1.22　以降 または、 H/W model　4 H/W Ver.　1 S/W Ver.　1.22　以降 以上は、ICS のハードを接続した状態で、ICS アプリの 　Help -> About で表示可能 ICS PC ソフト Ver. 2.5.0.0　以降	数値表示・設定 　8bit 符号なし　整数型 　8bit 符号あり　整数型 　16bit 符号なし　整数型 　16bit 符号あり　整数型 　32bit 符号なし　整数型 　32bit 符号あり　整数型 　8bit　BOOL 型 　8bit　LOGIC 型 波形表示 　8bit 符号なし　整数型 　8bit 符号あり　整数型 　16bit 符号なし　整数型 　16bit 符号あり　整数型 　32bit 符号なし　整数型 　32bit 符号あり　整数型

MPC.P15PFS.BIT.PSEL PORT1.PMR.BIT.B6 PORT1.PMR.BIT.B5 SCI5 INT　SCI5 RXI DTC　INT224　(TXI5) ICU.DTCER[224].BIT.DTCE SCI5　全て DTC　全て ICU.IPR[222].BYTE ICU.IER[0x1B].BIT.IEN6 ICU.IER[0x1B].BIT.IEN7 ICU.IER[0x1C].BIT.IEN0 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRA.BIT.B31 SYSTEM.MSTPCRB.BIT.B30 MPC.PWPR.BIT.B0WI MPC.PWPR.BIT.PFSWE ・Ext pin, PA4:TXD5, PA3:RXD5 MPC.PA3PFS.BIT.PSEL MPC.PA4PFS.BIT.PSEL PORTA.PMR.BIT.B4 PORTA.PMR.BIT.B3 ・Ext pin, PC3:TXD5, PC2:RXD5 MPC.PC3PFS.BIT.PSEL MPC.PC2PFS.BIT.PSEL PORTC.PMR.BIT.B3 PORTC.PMR.BIT.B2 SCI12 INT　SCI12 (RXI12) DTC　INT240　(TXI12) ICU.DTCER[240].BIT.DTCE SCI2　全て DTC　全て ICU.IPR[238].BYTE ICU.IER[0x1D].BIT.IEN6 ICU.IER[0x1D].BIT.IEN7 ICU.IER[0x1E].BIT.IEN0 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRA.BIT.B31 SYSTEM.MSTPCRB.BIT.B29 MPC.PWPR.BIT.B0WI		

MPC.PWPR.BIT.PFSWE ・Ext pin,PE1:TXD12, PE2:RXD12 MPC.PE1PFS.BIT.PSEL MPC.PE2PFS.BIT.PSEL PORTE.PMR.BIT.B1 PORTE.PMR.BIT.B2 ・Ext pin, P14:TXD12, P17:RXD12 MPC.P14PFS.BIT.PSEL MPC.P17PFS.BIT.PSEL PORT1.PMR.BIT.B4 PORT1.PMR.BIT.B7		

3.2.2. RX111 関数説明

Lib Ver.2.0 on CubeSuite+ Ver.2.01.00
初期化関数の呼び出し void ics_init(void* addr, char port, char level);
本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。 第１パラメータ DTC で使用する DTC のベクトルテーブルの先頭アドレスを指定してください。ics_init()関数を呼び出す前にユーザーが DTC ベクトルテーブルを確保する必要があります。このアドレスの下位 12bit は０である必要があります。 第２パラメータ SCI のポート番号や、SCI の使用するピンを設定します。このパラメータは、ICS_<CPUNAME>.h 内で定義されている文字列を使用してください。 第３パラメータ ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。 最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。 SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。

転送関数の呼び出し　　void　　ics_watchpoint(void);
本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。 本関数は、ＰＣで指定された変数のデータを読み出し、ＤＴＣ用の転送バッファーにコピーします。 この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてください。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。 $最小通信間隔=1/(通信速度[bps])\times 180+70[us]$ 通信速度が 1Mbps の際、上の式に数値を代入すると。 $最小通信間隔=1/(1[Mbps])\times 180+70[us]=250[us]$ ※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。
使用割り込み関数

下記の割り込みベクトルを使用しているため、ユーザーソフトの割り込みベクトルに下記の関数を登録してください。
ルネサス標準のコンパイラで自動的に生成されるプロジェクトを使用する場合、intprg.c という割り込み処理を記載したファイルに追記してください。
 	
下記のように記述してください。
 	
SCI1 の場合
// SCI1 ERI1
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
// SCI1 ERI1
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
 	
SCI5 の場合
// SCI5 ERI5
void Excep_SCI5_ERI5(void){ ics_int_sci_eri(); }
// SCI5 RXI5
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
 	
SCI12 の場合
// SCI12 ERI12
void Excep_SCI12_ERI12(void){ ics_int_sci_eri(); }
// SCI12 RXI12
void Excep_SCI12_RXI12(void){ ics_int_sci_rxi(); }

3.2.3. RX111 関数使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）開発環境に BDTCTBL セクションを確保する。

BDTCTBL のセクションを下位 12bit が０になるような RAM 上のアドレスに割り当てます。このアドレスを開発環境に設定してします。RX111 の RAM が最小のモデルはサイズが 8kByte なので、設定できるアドレスは、0x0000 もしくは、0x1000 となります。RAM が 16kByte のモデルの場合、設定できるアドレスは、0x0000, 0x1000, 0x2000, 0x3000 の３点となります。ここでは、0x0000 からに設定することにします。
E1 などのエミュレータを使用する場合、ユーザーRAM 領域もしくは、DTC テーブルの領域と、E1 エミュレータの使用領域とが重ならないようにしてください。正常に動作しないことがあります。

２）ユーザープログラムに、DTC テーブルを定義する。

ICS_sample.c の先頭に DTC テーブルの変数である unsigned long dtc_table[256]を定義する。

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section
```

３）ics_init()を下記のように呼び出す。

SCI1 を使用する場合、
初期化関数 ics_init((void*)dtc_table, ICS_SCI1_PC7_PC6) を初期化部分に入れてください。
第１パラメータは、１）で確保した BDTCTBL の先頭アドレスを入力してください。
第 2 パラメータは、使用するポートを ICS_<CPUNAME>.h から選択してください。
第 3 パラメータは、ICS で使用する割り込みレベルを設定してください。通常は、キャリア割り込みより低い優先順位に設定します。

---------- List 1 main.c ----------

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section

void main(void)
{
    ics_init((void*)dtc_table, ICS_SCI1_PC7_PC6, 6);    /* Interrupt level 6          */
    while(1)
    {   nop();   }
}
```

４）ics_watchpoint()関数の組込み

このソフトでは、メインルーチンで ics_watchpoint()関数を呼び出していますが、通常は、キャリア割り込みの内部で呼び出します。

また、この関数は、250us 以上, 5ms 以下の間隔で呼び出すようにしてください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List2 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2　ics_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics_watchpoint();
        }
}
```

５）intprg.c の修正

intprg.c の中の　RXI, ERI の割り込み関数から　ics_int_sci_eri(),　ics_int_sci_rxi()関数を呼び出すようにしてください。

SCI1 の場合

```
// SCI1 ERI1
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
// SCI1 RXI1
void Excep_SCI1_RXI1(void){ics_int_sci_rxi(); }
```

SCI5 の場合

```
// SCI5 ERI5
void Excep_SCI5_ERI5(void){ ics_int_sci_eri(); }
// SCI5 RXI5
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
```

SCI12 の場合

```
// SCI12 ERI12
void Excep_SCI12_ERI12(void){ ics_int_sci_eri(); }
// SCI12 RXI12
void Excep_SCI12_RXI12(void){ ics_int_sci_rxi(); }
```

3.2.4. RX111 ICS クロック

本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを PCLKB=32MHz で使用する必要があります。W1003 の標準品では、8MHz がついているので、RX111 を標準の 32MHz で使用する場合、ICS は、標準出荷状態で使用可能です。

RX111 を外部クロックで使用する場合、下記の式に従って、ICS 上のクロックを変更してください。
ICS ボード上のクロック周波数 = (PCLKB / 4) MHz

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項 :

ICS W1002 の場合（水晶発振器のソケットがないタイプ）
クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS W1003 の場合（水晶発振器のソケットがあるタイプ）
8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
可変クロックを内蔵しているため、PC 側からの操作が可能となります。

3.3. RL78G14 シリーズ

3.3.1. RL78G14 使用資源

CPU 名	RL78G14 シリーズ（各 CPU デバイス毎に個別のライブラリが存在）		
開発環境	CubeSuite+ Ver.2.01.00		
Library version	Ver2.00		
通信レート	通信レート $= \frac{f_{CLK}}{32}[Mbps]$ 標準通信レート　$f_{CLK} = 32MHz$　の場合　1Mbps		
ステイタス	SCI0, SCI1, SCI2 サポート		
ライブラリ種別	16bit ライブラリ		
ライブラリ	R5F104xx.rel というファイル名となります。 例：使用する CPU が R5F104LE の場合、“R5F104LE.rel”を使用します。		
ヘッダファイル	R5F104xx.h というファイル名となります。 例：使用する CPU が R5F104LE の場合、“R5F104LE.h”を使用します。		
メモリーモデル	Medium (ROM=1MB, RAM=64kB)		
DTC address mode	Standard		
Endian	Little		
サポートポート	R5F104AE (30pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P00_P01 (0x11)	
	R5F104BE (32pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P00_P01 (0x11)	
	R5F104CE (36pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P00_P01 (0x11)	
	R5F104EE (40pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P00_P01 (0x11)	
	R5F104FE (44pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P00_P01 (0x11)	
	R5F104GE (48pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P00_P01 (0x11)	
	R5F104JE (52pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P02_P03 (0x10) #define ICS_SCI2_P77_P76 (0x20)	
	R5F104LE (64pin)	#define ICS_SCI0_P51_P50 (0x00) #define ICS_SCI0_P17_P16 (0x01) #define ICS_SCI1_P02_P03 (0x10) #define ICS_SCI2_P77_P76 (0x20)	

CPU 使用リソース	対応 ICS	サポート変数タイプ
・内部リソース DTC *SCIx_Pab_Pcd x は、SCI 番号を示す数字 a, b, c, d は、ポート番号示す数字 SCIx 全資源 PFC PIOR0.1 （使用ポートによる） PMCa.b PMCc.d Pa.b = 1 PMa.b = 0 PMc.d = 1 外部ピン Pab for TXDx Pcd for RXDx CLOCK SPS0 bit4～7 INTC STPR0x STPR1x SRPR0x SRPR1x SREPR0x SREPR1x	○対応 ICS ハード 標準版 H/W model 1 H/W Ver. 1 S/W Ver. 1.22 以降 以上は、ICS のハードを接続した状態で、ICS アプリの Help -> About で表示可能 通信レート 1.00Mbps ICS PC ソフト Ver. 2.5.0.0 以降 Ver.2.7.3.0 以降を推奨	数値表示・設定 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 8bit BOOL 型 8bit LOGIC 型 波形表示 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型

3.3.2. RL78G14 シリーズ関数説明

Lib Ver.2.00　on　CubeSuite+ Ver.2.01.00
初期化関数の呼び出し　void ics_init(unsigned int addr, char pin, char level, unsigned char num);

本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。

第１パラメータ
DTC で使用する DTC のベクトルテーブルの先頭アドレスを指定してください。ics_init()関数を呼び出す前にユーザーが DTC ベクトルテーブルを確保する必要があります。 DTC ベクターテーブルのベースアドレスの先頭番地の下位１６ビットを渡します。 下位８ビットは０である必要があります。

第２パラメータ
　使用するピンを示します。ics_R5F104xx.h で定義されています。
　R5F104LE の場合、下記から選択します。
　#define　ICS_SCI0_P51_P50　(0x00)
　#define　ICS_SCI0_P17_P16　(0x01)
　#define　ICS_SCI1_P02_P03　(0x10)
　#define　ICS_SCI2_P77_P76　(0x20)
　が使用可能です。

第３パラメータ
　ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。
　最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。
　SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。

第４パラメータ
ＤＴＣの構造体の先頭番地です。８バイトの構造体でＤＴＣのどのコントロールデータを使用するかを示します。データとして、0x40, 0x48, 0x50... 0xF8 が使用可能です。

転送関数の呼び出し　　void　　ics_watchpoint(void);

本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。
本関数は、ＰＣで指定された変数のデータを読み出し、ＤＴＣ用の転送バッファーにコピーします。
この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてください。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 70[us]$$

通信速度が 1Mbps の際、上の式に数値を代入すると。

$$最小通信間隔 = 1/(1[Mbps]) \times 180 + 70[us] = 250[us]$$

※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。

使用割り込み関数

下記の割り込みベクトルを使用しています。
INTST0, INTSR0, INTSRE0
INTST1, INTSR1, INTSRE1
INTST2, INTSR2, INTSRE2

使用するチャンネルに応じて下のような、割り込みルーチンを作成し、
int_ics_sci_tx(),　int_ics_sci_rx(),　int_ics_sci_err()関数を呼び出してください。

```
#ifdef ICS_SCI0
__interrupt void   Excep_INTST0(void)  {int_ics_sci_tx();}
__interrupt void   Excep_INTSR0(void)  {int_ics_sci_rx();}
__interrupt void   Excep_INTSRE0(void) {int_ics_sci_err();}
#endif

#ifdef ICS_SCI1
__interrupt void   Excep_INTST1(void)  {int_ics_sci_tx();}
__interrupt void   Excep_INTSR1(void)  {int_ics_sci_rx();}
__interrupt void   Excep_INTSRE1(void) {int_ics_sci_err();}
#endif


#ifdef ICS_SCI2
__interrupt void   Excep_INTST2(void)  {int_ics_sci_tx();}
__interrupt void   Excep_INTSR2(void)  {int_ics_sci_rx();}
__interrupt void   Excep_INTSRE2(void) {int_ics_sci_err();}
#endif
```

3.3.3. RL78G14 シリーズ 使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）DTC テーブルを確保する。
　DTC テーブルを確保する方法は、いくつかありますが、ここでは、ソースコード上から確認できる方法を紹介します。
　DTC テーブルを確保するために、メインプログラムに次のような記述を追加します。
ここでは、0xFFE00 から 0xD0 バイトを確保します。お客様のメモリー使用状況に応じて変更してください。この際に、下位の 8bit は、0 にする必要があります。

```
#pragma section @@DATA   @@DTCTBL at 0xFFE00
char    dtc_tbl[0xD0];
#pragma section @@DATA   @@DATA
```

　E1 などのエミュレータを使用する場合、ユーザーRAM 領域もしくは、DTC テーブルの領域と、E1 エミュレータの使用領域とが重ならないようにしてください。正常に動作しないことがあります。

２）ics_init()を下記のように呼び出す。

SCI1 を使用する場合、初期化関数 ics_init() を初期化部分に入れてください。内部で ICS が使用する資源のみを初期化しているため、ユーザー側でピンを初期化している場合には、ユーザー側の初期化終了後に呼び出してください。ICS は DTC を使用しているため、DTC のベクトルテーブルを設定する必要があります。ics_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFE00 に設定しています。

第１パラメータは、ＤＴＣのテーブルアドレス、下位 8bit が０である必要があります。
第２パラメータは、ＤＴＣの構造体のアドレスです。８バイトの構造体でＤＴＣのどのコントロールアドレスを使用するかを示します。データとして、0x40, 0x48, 0x50, … 0xD8 が使用可能です。先に設定した 0xFE00 を DTC ベクトルアドレスに設定した場合、0xE0, 0xE8, 0xF0, 0xF8 は使用できません。
DTC を他の用途で使用しない場合には、0x40 と設定しておけば間違いありません。
第３パラメータは、ICS で使用する割り込みレベルを指定します。通常、２を指定します。
第４パラメータは、ピンを示します。ics_R5F104xx.h で定義されている文字列をお使いください。
　R5F104LE の場合、下記のピンが使用可能です。

```
#define    ICS_SCI0_P51_P50     (0x00)
#define    ICS_SCI0_P17_P16     (0x01)
#define    ICS_SCI1_P02_P03     (0x10)
#define    ICS_SCI2_P77_P76     (0x20)
```

---------- List 1　main.c ----------

```
#pragma　SFR
#pragma　DI
#pragma　EI
#pragma　NOP

#include "ICS_define.h"

#include "low_level_init.h"
#include "ics_R5F104LE.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma section @@DATA　@@DTCTBL at 0xFFE00
char　dtc_tbl[0xD0];
#pragma section @@DATA　@@DATA

    ics_init(0xFE00, ICS_SCI1_P12_P11, 2, 0x40);
```

３）ics_watchpoint()関数の組込み

250us 以上、5ms 以下の間隔で呼び出されるように、ics_watchpoint()を呼び出してください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List3 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2　ics_watchpoint()の間引き ----------

```
__interrupt void　int_TM0(void)
{
    theta_e_est = theta_e_est + 60;
    if (theta_e_est>4095)
    {
        theta_e_est = theta_e_est - 4096;
    }

    /********** pwm reference generation **************/
    refu = R_FIX_sin_int16(theta_e_est);
    refv = R_FIX_sin_int16(theta_e_est-1333);
    refw = R_FIX_sin_int16(theta_e_est-2666);

    RPECTL = 0x80U;
    ics_watchpoint();
}
```

４）割り込み処理プログラムの追加
　下記のように割り込み関数を追加してください。

SCI0 の場合
```
__interrupt void   Excep_INTST0(void)  {int_ics_sci_tx();}
__interrupt void   Excep_INTSR0(void)  {int_ics_sci_rx();}
__interrupt void   Excep_INTSRE0(void) {int_ics_sci_err();}
```

SCI1 の場合
```
__interrupt void   Excep_INTST1(void)  {int_ics_sci_tx();}
__interrupt void   Excep_INTSR1(void)  {int_ics_sci_rx();}
__interrupt void   Excep_INTSRE1(void) {int_ics_sci_err();}
```

SCI2 の場合
```
__interrupt void   Excep_INTST2(void)  {int_ics_sci_tx();}
__interrupt void   Excep_INTSR2(void)  {int_ics_sci_rx();}
__interrupt void   Excep_INTSRE2(void) {int_ics_sci_err();}
```

3.3.4.　RL78G14 向け　ICS クロック

　RL78G14 の場合、通常 CLK=32MHz で使います。ただし、CLK=32MHz 以外で使用する場合、下記のような対応が必要になります。
　本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを fclk=32MHz で使用する必要があります。W1003 の標準品では、8MHz がついているので、RL78G14 を標準の 32MHz で使用する場合、ICS は、標準出荷状態で使用可能です。

RL78G14 を外部クロックで使用する場合、下記の式に従って、ICS 上のクロックを変更してください。
　ICS ボード上のクロック周波数 ＝ (fclk/4) MHz

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項 :

ICS　W1002 の場合（水晶発振器のソケットがないタイプ）
　クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS　W1003 の場合（水晶発振器のソケットがあるタイプ）
　8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
　可変クロックを内蔵しているため、PC 側からの操作が可能となります。

3.4. RL78F14 シリーズ

3.4.1. RL78F14 使用資源

CPU 名	RL78F14 シリーズ（R5F10PMF のみ）	
開発環境	CubeSuite+ Ver.2.01.00	
Library version	Ver2.00	
通信レート	通信レート $= \frac{f_{CLK}}{32}[Mbps]$ 標準通信レート　$f_{CLK} = 32MHz$　の場合　1Mbps	
ステイタス	SCI0, SCI1, SCI2 サポート	
ライブラリ種別	16bit ライブラリ	
ライブラリ	R5F10Pxx.rel というファイル名となります。 例：使用する CPU が R5F10PMF の場合、“R5F10PMF.rel”を使用します。	
ヘッダファイル	R5F10Pxx.h というファイル名となります。 例：使用する CPU が R5F10PMF の場合、“R5F10PMF.h”を使用します。	
メモリーモデル	Medium (ROM=1MB, RAM=64kB)	
DTC address mode	Standard	
Endian	Little	
サポートポート	R5F10PMF (80pin)	#define　ICS_SCI0_P62_P61　(0x00) #define　ICS_SCI1_P74_P75　(0x10) #define　ICS_SCI1_P12_P11　(0x11)

CPU 使用リソース	対応 ICS	サポート変数タイプ
・内部リソース DTC *SCIx_Pab_Pcd 　x は、SCI 番号を示す数字 　a, b, c, d は、ポート番号示す数字 SCIx 全資源 PFC 　PIOR4　（使用ポートによる） 　Pa.b = 1 　PMa.b = 0 　PMc.d = 1 外部ピン 　Pab　for TXDx 　Pcd　for RXDx CLOCK 　SPS0　bit4～7 INTC 　　STPR0x 　　STPR1x 　　SRPR0x 　　SRPR1x	○対応 ICS ハード 標準版 H/W model　1 H/W Ver.　1 S/W Ver.　1.22　以降 以上は、ICS のハードを接続した状態で、ICS アプリの 　Help -> About で表示可能 通信レート 1.00Mbps ICS PC ソフト Ver. 2.5.0.0　以降 Ver.2.7.3.0　以降を推奨	数値表示・設定 　8bit 符号なし　整数型 　8bit 符号あり　整数型 　16bit 符号なし　整数型 　16bit 符号あり　整数型 　32bit 符号なし　整数型 　32bit 符号あり　整数型 　8bit　BOOL 型 　8bit　LOGIC 型 波形表示 　8bit 符号なし　整数型 　8bit 符号あり　整数型 　16bit 符号なし　整数型 　16bit 符号あり　整数型

3.4.2. RL78F14 シリーズ関数説明

Lib Ver.2.00 on CubeSuite+ Ver.2.01.00
初期化関数の呼び出し void ics_init(unsigned int addr, char pin, char level, unsigned char num);

本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。

第１パラメータ
DTC で使用する DTC のベクトルテーブルの先頭アドレスを指定してください。ics_init()関数を呼び出す前にユーザーが DTC ベクトルテーブルを確保する必要があります。 DTC ベクターテーブルのベースアドレスの先頭番地の下位１６ビットを渡します。 下位８ビットは、０である必要があります。

第２パラメータ
使用するピンを示します。ics_R5F10PMF.h で定義されています。
R5F10PMF の場合、下記から選択します。
#define ICS_SCI0_P62_P61 (0x00)
#define ICS_SCI1_P74_P75 (0x10)
#define ICS_SCI1_P12_P11 (0x11)が使用可能です。

第３パラメータ
ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。
最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。
SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。

第４パラメータ
ＤＴＣの構造体の先頭番地です。８バイトの構造体でＤＴＣのどのコントロールデータを使用するかを示します。データとして、0x40, 0x48, 0x50... 0xF8 が使用可能です。

転送関数の呼び出し　　void　　ics_watchpoint(void);
本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。 本関数は、ＰＣで指定された変数のデータを読み出し、DTC 用の転送バッファーにコピーします。 この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてください。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。 $最小通信間隔=1/(通信速度[bps])\times 180+70[us]$ 通信速度が 1Mbps の際、上の式に数値を代入すると。 $最小通信間隔=1/(1[Mbps])\times 180+70[us]=250[us]$ ※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。
使用割り込み関数
下記の割り込みベクトルを使用しています。 INTST0, INTSR0 INTST1, INTSR1 使用するチャンネルに応じて下のような、割り込みルーチンを作成し、 int_ics_sci_tx(),　int_ics_sci_rx() 関数を呼び出してください。 #ifdef ICS_SCI0 __interrupt void　Excep_INTST0(void)　{int_ics_sci_tx();} __interrupt void　Excep_INTSR0(void)　{int_ics_sci_rx();} #endif #ifdef ICS_SCI1 __interrupt void　Excep_INTST1(void)　{int_ics_sci_tx();} __interrupt void　Excep_INTSR1(void)　{int_ics_sci_rx();} #endif

3.4.3. RL78F14 シリーズ 使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）DTC テーブルを確保する。
　DTC テーブルを確保する方法は、いくつかありますが、ここでは、ソースコード上から確認できる方法を紹介します。
　DTC テーブルを確保するために、メインプログラムに次のような記述を追加します。
ここでは、0xFFE00 から 0xD0 バイトを確保します。お客様のメモリー使用状況に応じて変更してください。この際に、下位の 8bit は、0 にする必要があります。

```
#pragma section @@DATA   @@DTCTBL at 0xFFE00
char    dtc_tbl[0xD0];
#pragma section @@DATA   @@DATA
```

　E1 などのエミュレータを使用する場合、ユーザーRAM 領域もしくは、DTC テーブルの領域と、E1 エミュレータの使用領域とが重ならないようにしてください。正常に動作しないことがあります。

２）ics_init()を下記のように呼び出す。

SCI1 を使用する場合、初期化関数 ics_init() を初期化部分に入れてください。内部で ICS が使用する資源のみを初期化しているため、ユーザー側でピンを初期化している場合には、ユーザー側の初期化終了後に呼び出してください。ICS は DTC を使用しているため、DTC のベクトルテーブルを設定する必要があります。ics_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFE00 に設定しています。

第１パラメータは、ＤＴＣのテーブルアドレス、下位 8bit が０である必要があります。
第２パラメータは、ＤＴＣの構造体のアドレスです。８バイトの構造体でＤＴＣのどのコントロールアドレスを使用するかを示します。データとして、0x40, 0x48, 0x50, … 0xD8 が使用可能です。先に設定した 0xFE00 を DTC ベクトルアドレスに設定した場合、0xE0, 0xE8, 0xF0, 0xF8 は使用できません。
DTC を他の用途で使用しない場合には、0x40 と設定しておけば間違いありません。
第３パラメータは、ICS で使用する割り込みレベルを指定します。通常、２を指定します。
第４パラメータは、ピンを示します。ics_R5F10PMF.h で定義されている文字列をお使いください。
　R5F10PMF の場合、下記のピンが使用可能です。

```
#define    ICS_SCI0_P62_P61     (0x00)
#define    ICS_SCI1_P74_P75     (0x10)
#define    ICS_SCI1_P12_P11     (0x11)
```

---------- List 1　main.c ----------

```
#pragma　SFR
#pragma　DI
#pragma　EI
#pragma　NOP

#include "ICS_define.h"

#include "low_level_init.h"
#include "ics_R5F10PMF.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma section @@DATA　@@DTCTBL at 0xFFE00
char　dtc_tbl[0xD0];
#pragma section @@DATA　@@DATA

    ics_init(0xFE00, ICS_SCI1_P12_P11, 2, 0x40);
```

３）ics_watchpoint()関数の組込み

250us 以上、5ms 以下の間隔で呼び出されるように、ics_watchpoint()を呼び出してください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List3 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2　ics_watchpoint()の間引き ----------

```
__interrupt void　int_TM0(void)
{
    theta_e_est = theta_e_est + 60;
    if (theta_e_est>4095)
    {
        theta_e_est = theta_e_est - 4096;
    }

    /********** pwm reference generation **************/
    refu = R_FIX_sin_int16(theta_e_est);
    refv = R_FIX_sin_int16(theta_e_est-1333);
    refw = R_FIX_sin_int16(theta_e_est-2666);

    RPECTL = 0x80U;
    ics_watchpoint();
}
```

４）割り込み処理プログラムの追加
　下記のように割り込み関数を追加してください。

SCI0 の場合
__interrupt void　Excep_INTST0(void)　{int_ics_sci_tx();}
__interrupt void　Excep_INTSR0(void)　{int_ics_sci_rx();}

SCI1 の場合
__interrupt void　Excep_INTST1(void)　{int_ics_sci_tx();}
__interrupt void　Excep_INTSR1(void)　{int_ics_sci_rx();}

3.4.4.　RL78F14 向け　ICS クロック

　RL78F14 の場合、通常 CLK=32MHz で使います。ただし、CLK=32MHz 以外で使用する場合、下記のような対応が必要になります。
　本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを fclk=32MHz で使用する必要があります。W1003 の標準品では、8MHz がついているので、RL78F14 を標準の 32MHz で使用する場合、ICS は、標準出荷状態で使用可能です。

RL78F14 を外部クロックで使用する場合、下記の式に従って、ICS 上のクロックを変更してください。
　ICS ボード上のクロック周波数 ＝　(fclk/4) MHz

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項：

ICS　W1002 の場合（水晶発振器のソケットがないタイプ）
　クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS　W1003 の場合（水晶発振器のソケットがあるタイプ）
　8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
　可変クロックを内蔵しているため、PC 側からの操作が可能となります。

3.5. RL78G1F シリーズ

3.5.1. RL78G1F 使用資源

CPU 名	RL78G1F シリーズ　R5F11BLEAFB (64pin)	
開発環境	CS+ Ver.3.00.00	
Library version	Ver2.01	
通信レート	通信レート $= \frac{f_{CLK}}{32}[Mbps]$ 標準通信レート　$f_{CLK} = 32MHz$　の場合 1Mbps	
ステイタス	SCI0, SCI2 サポート	
ライブラリ種別	16bit ライブラリ	
ライブラリ	IC のピンサイズによって、ライブラリが異なります。 CPU の通称名の後にパッケージ名がつきます、 64pin, ROM=64kB　R5F11BLEAFB の場合　->　“ICS_RL78G1F_Lx.lib” 48pin, ROM=64kB　R5F11BGEAFB の場合　->　“ICS_RL78G1F_Gx.lib” 36pin, ROM=64kB　R5F11BCEALA の場合　->　“ICS_RL78G1F_Cx.lib” 32pin, ROM=64kB　R5F11BBEAFP の場合　->　“ICS_RL78G1F_Bx.lib” 24pin, ROM=64kB　R5F11B7EANA の場合　->　“ICS_RL78G1F_7x.lib”	
ヘッダファイル	ライブラリと同様の命名規則となります。 64pin, ROM=64kB　R5F11BLEAFB の場合　->　“ICS_RL78G1F_Lx.h” 48pin, ROM=64kB　R5F11BGEAFB の場合　->　“ICS_RL78G1F_Gx.h” 36pin, ROM=64kB　R5F11BCEALA の場合　->　“ICS_RL78G1F_Cx.h” 32pin, ROM=64kB　R5F11BBEAFP の場合　->　“ICS_RL78G1F_Bx.h” 24pin, ROM=64kB　R5F11B7EANA の場合　->　“ICS_RL78G1F_7x.h”	
メモリーモデル	Medium (ROM=1MB, RAM=64kB)	
DTC address mode	Standard	
Endian	Little	
サポートポート	R5F11BLx (64pin)	#define　ICS_SCI0_P51_P50　(0x00) #define　ICS_SCI0_P17_P16　(0x01) #define　ICS_SCI2_P77_P76　(0x20)
	R5F11BGx (48pin)	Not supported now
	R5F11BCx (36pin)	Not supported now
	R5F11BBx (32pin)	Not supported now
	R5F11B7x (24pin)	Not supported now

CPU 使用リソース	対応 ICS	サポート変数タイプ
・内部リソース 1) ICS_SCI0_P51_P50 DTC SCI0 全資源 PFC PIOR0.1 = 0 P5.0 = 1 PM5.1 = 0 PM5.0 = 1 外部ピン P51 for TXD0 P50 for RXD0 CLOCK SPS0 bit4～7 INTC STPR00 STPR10 SRPR00 SRPR10 2) ICS_SCI0_P17_P16 DTC SCI0 全資源 PFC PIOR0.1 = 1 P1.7 = 1 PM1.7 = 0 PM1.6 = 1 外部ピン P17 for TXD0 P16 for RXD0 CLOCK SPS0 bit4～7 INTC STPR00 STPR10 SRPR00 SRPR10 3) ICS_SCI2_P77_P76 DTC SCI2 全資源 PFC PIOR0.1 = 1 P7.7 = 1	○対応 ICS ハード 標準版 H/W model 1 H/W Ver. 1 S/W Ver. 1.22 以降 以上は、ICS のハードを接続した状態で、ICS アプリの Help -> About で表示可能 通信レート 1.00Mbps ICS PC ソフト Ver. 2.5.0.0 以降 Ver.2.7.3.0 以降を推奨	数値表示・設定 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 8bit BOOL 型 8bit LOGIC 型 波形表示 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型

PM7.7 = 0 PM7.6 = 1 外部ピン P77　for TXD2 P76　for RXD2 CLOCK SPS0　bit4～7 INTC STPR02 STPR12 SRPR02 SRPR12		

3.5.2. RL78G1F シリーズ関数説明

Lib Ver.2.01　on　CS+ Ver.3.xx.xx
初期化関数の呼び出し　void ics_init(unsigned int addr, char pin, char level, unsigned char num);
本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。 第１パラメータ DTC で使用する DTC のベクトルテーブルの先頭アドレスを指定してください。ics_init()関数を呼び出す前にユーザーが DTC ベクトルテーブルを確保する必要があります。 DTC ベクターテーブルのベースアドレスの先頭番地の下位１６ビットを渡します。 下位８ビットは、０である必要があります。 第２パラメータ 使用するピンを示します。ヘッダファイルに定義されています。 R5F11Bxxxx の場合、下記から選択します。 #define　ICS_SCI0_P51_P50　(0x00) #define　ICS_SCI0_P17_P16　(0x01) #define　ICS_SCI2_P77_P76　(0x20)が使用可能です。 第３パラメータ ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。 最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。 SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。 第４パラメータ ＤＴＣの構造体の先頭番地です。８バイトの構造体でＤＴＣのどのコントロールデータを使用するかを示します。データとして、0x40, 0x48, 0x50... 0xF8 が使用可能です。

転送関数の呼び出し　　void　　ics_watchpoint(void);
本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。 本関数は、ＰＣで指定された変数のデータを読み出し、DTC 用の転送バッファーにコピーします。 この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてください。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。 $最小通信間隔=1/(通信速度[bps])\times 180+70[us]$ 通信速度が 1Mbps の際、上の式に数値を代入すると。 $最小通信間隔=1/(1[Mbps])\times 180+70[us]=250[us]$ ※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。
使用割り込み関数
下記の割り込みベクトルを使用しています。 INTST0, INTSR0 INTST2, INTSR2 使用するチャンネルに応じて下のような、割り込みルーチンを作成し、 int_ics_sci_tx(),　int_ics_sci_rx() 関数を呼び出してください。 #ifdef ICS_SCI0 __interrupt void　Excep_INTST0(void)　{int_ics_sci_tx();} __interrupt void　Excep_INTSR0(void)　{int_ics_sci_rx();} #endif #ifdef ICS_SCI2 __interrupt void　Excep_INTST2(void)　{int_ics_sci_tx();} __interrupt void　Excep_INTSR2(void)　{int_ics_sci_rx();} #endif

3.5.3. RL78G1F シリーズ 使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）DTC テーブルを確保する。
　DTC テーブルを確保する方法は、いくつかありますが、ここでは、ソースコード上から確認できる方法を紹介します。
　DTC テーブルを確保するために、メインプログラムに次のような記述を追加します。
ここでは、0xFFE00 から 0xD0 バイトを確保します。お客様のメモリー使用状況に応じて変更してください。この際に、下位の 8bit は、0 にする必要があります。

```
#pragma section @@DATA   @@DTCTBL at 0xFFE00
char    dtc_tbl[0xD0];
#pragma section @@DATA   @@DATA
```

　E1 などのエミュレータを使用する場合、ユーザーRAM 領域もしくは、DTC テーブルの領域と、E1 エミュレータの使用領域とが重ならないようにしてください。正常に動作しないことがあります。

２）ics_init()を下記のように呼び出す。

SCI1 を使用する場合、初期化関数 ics_init() を初期化部分に入れてください。内部で ICS が使用する資源のみを初期化しているため、ユーザー側でピンを初期化している場合には、ユーザー側の初期化終了後に呼び出してください。ICS は DTC を使用しているため、DTC のベクトルテーブルを設定する必要があります。ics_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFE00 に設定しています。

第１パラメータは、ＤＴＣのテーブルアドレス、下位 8bit が０である必要があります。
第２パラメータは、ＤＴＣの構造体のアドレスです。８バイトの構造体でＤＴＣのどのコントロールアドレスを使用するかを示します。データとして、0x40, 0x48, 0x50, … 0xD8 が使用可能です。先に設定した 0xFE00 を DTC ベクトルアドレスに設定した場合、0xE0, 0xE8, 0xF0, 0xF8 は使用できません。
DTC を他の用途で使用しない場合には、0x40 と設定しておけば間違いありません。
第３パラメータは、ICS で使用する割り込みレベルを指定します。通常、２を指定します。
第４パラメータは、ピンを示します。一例として、R5F11BLEAFB を使いの場合、ics_RL78G1F_L.h で定義されている文字列をお使いください。
　R5F11BLEAFB の場合、下記のピンが使用可能です。

```
#define    ICS_SCI0_P51_P50     (0x00)
#define    ICS_SCI0_P17_P16     (0x01)
#define    ICS_SCI2_P77_P76     (0x20)
```

---------- List 1　main.c ----------

```
#pragma　SFR
#pragma　DI
#pragma　EI
#pragma　NOP

#include "ICS_define.h"

#include "low_level_init.h"
#include "ics_RL78G1F_Lx.h"

/********** KEEP DTC TABLE AREA **********/
#pragma section @@DATA　@@DTCTBL at 0xFFE00
char　dtc_tbl[0xD0];
#pragma section @@DATA　@@DATA

    ics_init(0xFE00, ICS_SCI0_P17_P16, 2, 0x40);
```

３）ics_watchpoint()関数の組込み

250us 以上、5ms 以下の間隔で呼び出されるように、ics_watchpoint()を呼び出してください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List3 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2　ics_watchpoint()の間引き ----------

```
__interrupt void　int_TM0(void)
{
    theta_e_est = theta_e_est + 60;
    if (theta_e_est>4095)
    {
        theta_e_est = theta_e_est - 4096;
    }

    /********** pwm reference generation **************/
    refu = R_FIX_sin_int16(theta_e_est);
    refv = R_FIX_sin_int16(theta_e_est-1333);
    refw = R_FIX_sin_int16(theta_e_est-2666);

    RPECTL = 0x80U;
    ics_watchpoint();
}
```

４）割り込み処理プログラムの追加
　下記のように割り込み関数を追加してください。

SCI0 の場合
__interrupt void　Excep_INTST0(void)　{int_ics_sci_tx();}
__interrupt void　Excep_INTSR0(void)　{int_ics_sci_rx();}

SCI2 の場合
__interrupt void　Excep_INTST2(void)　{int_ics_sci_tx();}
__interrupt void　Excep_INTSR2(void)　{int_ics_sci_rx();}

3.5.4.　RL78G1F 向け　ICS クロック

　RL78G1F の場合、通常 CLK=32MHz で使います。ただし、CLK=32MHz 以外で使用する場合、下記のような対応が必要になります。
　本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを fclk=32MHz で使用する必要があります。W1003 の標準品では、8MHz がついているので、RL78F14 を標準の 32MHz で使用する場合、ICS は、標準出荷状態で使用可能です。
　W1004 光ファイバー版の場合、PC 側からソフトウェアで周波数を設定できます。

RL78G1F を外部クロックで使用する場合、下記の式に従って、ICS 上のクロックを変更してください。
　ICS ボード上のクロック周波数 ＝ (fclk/4) MHz

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項：

ICS　W1002 の場合（水晶発振器のソケットがないタイプ）
　クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS　W1003 の場合（水晶発振器のソケットがあるタイプ）
　8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
　可変クロックを内蔵しているため、PC 側からの操作が可能となります。

3.6. RX64M シリーズ

3.6.1. RX64M 使用資源

CPU 名	RX64M シリーズ共通
開発環境	CubeSuite+ Ver.2.0x.xx
Library version	Ver.2.0
通信レート	通信レート = $\frac{PCLKB}{48}[Mbps]$ 標準通信レート　PCLKB = 60MHz　の場合 1.25Mbps
ステイタス	SCI0, SCI1, SCI2 サポート
ライブラリ種別	32bit ライブラリ
ライブラリ	ics_RX64M.obj
ヘッダファイル	ics_RX64M.h

CPU 使用リソース	対応 ICS	サポート変数タイプ
・内部使用リソース SCI0 (P32, P33) INT SCI0 RXI INT SCI0 TXI DTC INT59 (TXI0) ICU.DTCER[59].BIT.DTCE SCI0 全て DTC 全て ICU.IPR[58].BYTE ICU.IPR[59].BYTE ICU.IER[0x07].BIT.IEN2 ICU.IER[0x07].BIT.IEN3 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRB.BIT.B31 MPC.P32PFS.BYTE MPC.P33PFS.BYTE PORT3.PMR.BIT.B2 = 1 PORT3.PMR.BIT.B3 = 1 外部ピン P32: TXD0 P33: RXD0 SCI1 (P16, P15) INT SCI1 RXI INT SCI1 TXI DTC INT61 (TXI1) ICU.DTCER[61].BIT.DTCE SCI1 全て DTC 全て	○対応 ICS ハード ２種類の対応が可能です。 標準版 H/W model 1 H/W Ver. 1 S/W Ver. 1.22 以降 または、 H/W model 4 H/W Ver. 1 S/W Ver. 1.22 以降 以上は、ICS のハードを接続した状態で、ICS アプリの Help -> About で表示可能 ICS PC ソフト Ver. 2.5.0.0 以降	数値表示・設定 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 32bit IEEE754 浮動小数点 8bit BOOL 型 8bit LOGIC 型 波形表示 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 32bit IEEE754 浮動小数点

ICU.IPR[60].BYTE ICU.IPR[61].BYTE ICU.IER[0x07].BIT.IEN4 ICU.IER[0x07].BIT.IEN5 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRB.BIT.B30 MPC.P16PFS.BYTE MPC.P15PFS.BYTE PORT1.PMR.BIT.B6 = 1 PORT1.PMR.BIT.B5 = 1 外部ピン P16: TXD1 P15: RXD1 SCI2 (P13, P12) INT SCI2RXI INT SCI2TXI DTC INT63 (TXI2) ICU.DTCER[63].BIT.DTCE SCI2 全て DTC 全て ICU.IPR[62].BYTE ICU.IPR[63].BYTE ICU.IER[0x07].BIT.IEN6 ICU.IER[0x07].BIT.IEN7 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRB.BIT.B29 MPC.P13PFS.BYTE MPC.P12PFS.BYTE PORT1.PMR.BIT.B3 = 1 PORT1.PMR.BIT.B2 = 1 外部ピン P13: TXD2 P12: RXD2		

3.6.2. RX64M 関数説明

Lib Ver.2.0 on CubeSuite+ Ver.2.0x.xx
初期化関数の呼び出し void ics_init(void* addr, char port, char level);
本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。 第１パラメータ DTC で使用する DTC のベクトルテーブルの先頭アドレスを指定してください。ics_init()関数を呼び出す前にユーザーが DTC ベクトルテーブルを確保する必要があります。このアドレスの下位 12bit は０である必要があります。 第２パラメータ SCI のポート番号や、SCI の使用するピンを設定します。このパラメータは、ICS_<CPUNAME>.h 内で定義されている文字列を使用してください。 第３パラメータ ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。 最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。 SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。
転送関数の呼び出し void ics_watchpoint(void);
本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。 本関数は、ＰＣで指定された変数のデータを読み出し、ＤＴＣ用の転送バッファーにコピーします。 この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてください。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。 $最小通信間隔=1/(通信速度[bps])\times 180+70[us]$ 通信速度が 1Mbps の際、上の式に数値を代入すると。 $最小通信間隔=1/(1[Mbps])\times 180+70[us]=250[us]$ ※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。

使用割り込み関数

下記の割り込みベクトルを使用しているため、ユーザーソフトの割り込みベクトルに下記の関数を登録してください。
　ルネサス標準のコンパイラで自動的に生成されるプロジェクトを使用する場合、intprg.c という割り込み処理を記載したファイルに追記してください。

　たとえば、SCI0 を使用する場合、下記のように記述してください。
SCI0 の場合

```
// SCI0 ERI0
void Excep_SCI0_RXI0(void){ ics_int_sci_rxi(); }
```

SCI1 の場合

```
// SCI1 RXI1
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
```

SCI2 の場合

```
// SCI2 RXI2
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
```

3.6.3. RX64M 関数使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）開発環境に BDTCTBL セクションを確保する。

BDTCTBL のセクションを下位 12bit が０になるような RAM 上のアドレスに割り当てます。このアドレスを開発環境に設定してします。ここでは、0x3000 からに設定することにします。
E1 などのエミュレータを使用する場合、ユーザーRAM 領域もしくは、DTC テーブルの領域と、E1 エミュレータの使用領域とが重ならないようにしてください。正常に動作しないことがあります。

２）ユーザープログラムに、DTC テーブルを定義する。

ICS_sample.c の先頭に DTC テーブルの変数である unsigned long dtc_table[256]を定義する。

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section
```

３）ics_init()を下記のように呼び出す。

初期化関数 ics_init((void*)dtc_table, ICS_SCI0_P32_P33, 6) を初期化部分に入れてください。
第１パラメータは、１）で確保した BDTCTBL の先頭アドレスを入力してください。
第 2 パラメータは、使用するポートを ICS_<CPUNAME>.h から選択してください。
第 3 パラメータは、ICS で使用する割り込みレベルを設定してください。通常は、キャリア割り込みより低い優先順位に設定します。

---------- List 1 main.c ----------

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section

void main(void)
{
      ics_init((void*)dtc_table, ICS_SCI0_P32_P33, 6);      /* Interrupt level 6              */
      while(1)
      {    nop();    }
}
```

４）ics_watchpoint()関数の組込み

このソフトでは、メインルーチンで ics_watchpoint()関数を呼び出していますが、通常は、キャリア割り込みの内部で呼び出します。

また、この関数は、250us 以上, 5ms 以下の間隔で呼び出すようにしてください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List2 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2　ics_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics_watchpoint();
        }
}
```

５）intprg.c の修正

intprg.c の中の　RXI の割り込み関数から ics_int_sci_rxi()関数を呼び出すようにしてください。

SCI0 の場合

```
// SCI0 RXI0
void Excep_SCI0_RXI0(void){ ics_int_sci_rxi(); }
```

SCI1 の場合

```
// SCI1 RXI1
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
```

SCI2 の場合

```
// SCI2 RXI2
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
```

3.6.4. RX64M ICS クロック

本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを PCLKB=48MHz で使用する必要があります。

ICS ボード上のクロック周波数 = (PCLKB / 6) MHz

例： PCLKB = 60MHz の場合、 ICS CLOCK = 60/6 = 10.000MHz
PCLKB = 48MHz の場合、 ICS CLOCK = 48/6 = 8.000MHz

デスクトップラボでは、 標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項：

ICS W1002 の場合（水晶発振器のソケットがないタイプ）
クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS W1003 の場合（水晶発振器のソケットがあるタイプ）
8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
可変クロックを内蔵しているため、PC 側からの操作が可能となります。

3.7. V850E2M/FJ4 シリーズ

3.7.1. V850E2M/FJ4 使用資源

CPU 名	V850E2M/Fx4 シリーズ
開発環境	CubeSuite+ Ver.2.02.00
Library version	Ver.2.0
通信レート	$通信レート = \frac{PCLK}{80}[Mbps]$ 標準通信レート： PCLK = 80MHz の場合 1Mbps
ステイタス	UARTE4, UARTE5, UARTE10, UARTE11 サポート
ライブラリ種別	32bit ライブラリ
ライブラリ	ics_V850FJ4.obj
ヘッダファイル	ics_V850FJ4.h

CPU 使用リソース	対応 ICS	サポート変数タイプ
・内部使用リソース UARTE4 INT INTLMA4IR INT INTLMA4IS DMA3 INTLMA4IT ICU.DTCER[215].BIT.DTCE UARTE4 全て DMA3 全て /* Set URTE4RX pin */ FCLA27CTL2 = 0x80U; PFC1 |= 0x0200U; PFCE1 |= 0x0200U; PMC1 |= 0x0200U; PM1 |= 0x0200U; /* Set URTE4TX pin */ PFC1 |= 0x0100U; PFCE1 |= 0x0100U; PMC1 |= 0x0100U; PM1 &= (~0x0100U); UARTE5 INT INTLMA5IR INT INTLMA5IS DMA3 INTLMA5IT UARTE5 全て DMA3 全て /* Set URTE5RX pin */ FCLA27CTL3 = 0x80U; PFC25 &= (~0x4000U); PFCE25 |= 0x4000U;	○対応 ICS, ICS++ hardware W1001 H/W model 1 H/W Ver. 1 S/W Ver. 1.22 以降 W1003 H/W model 4 H/W Ver. 1 S/W Ver. 1.22 以降 以上は、ICS のハードを接続した状態で、ICS アプリの Help ->About で表示可能 ICS PC ソフト Ver. 2.5.0.0 以降	数値表示・設定 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 32bit IEEE754 浮動小数点 8bit BOOL 型 8bit LOGIC 型 波形表示 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 32bit IEEE754 浮動小数点

PMC25 |= 0x4000U; PM25 |= 0x4000U; /* Set URTE5TX pin */ PFC25 &= ~(0x8000U); PFCE25 |= 0x8000U; PMC25 |= 0x8000U; PM25 &= ~(0x8000U); UARTE10 INT INTLMA10IR INT INTLMA10IS DMA3 INTLMA10IT UARTE10 全て DMA3 全て /* Set URTE10RX pin */ FCLA7CTL0 = 0x80U; PFC4 |= 0x0010U; PFCE4 &= (~0x0010U); PMC4 |= 0x0010U; PM4 |= 0x0010U; /* Set URTE10TX pin */ PFC4 |= 0x0008U; PFCE4 &= (~0x0008U); PMC4 |= 0x0008U; PM4 &= (~0x0008U); UARTE11 INT INTLMA11IR INT INTLMA11IS DMA3 INTLMA11IT UARTE11 全て DMA3 全て /* Set URTE11RX pin */ FCLA7CTL1 = 0x80U; PFC0 &= (~0x0080U); PFCE0 &= (~0x0080U); PMC0 |= 0x0080U; PM0 |= 0x0080U; /* Set URTE11TX pin */ PFC0 &= (~0x0040U); PFCE0 &= (~0x0040U); PMC0 |= 0x0040U; PM0 &= (~0x0040U);		

3.7.2. V850E2M/Fx4 関数説明

Lib Ver.2.0 on CubeSuite+ Ver.2.02.00
初期化関数の呼び出し void ics_init(unsigned char port, unsigned char level);
本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。 第１パラメータ SCI のポート番号や、SCI の使用するピンを設定します。このパラメータは、ICS_<CPUNAME>.h 内で定義されている文字列を使用してください。 第２パラメータ ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。 最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。 SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。
転送関数の呼び出し void ics_watchpoint(void);
本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。 本関数は、ＰＣで指定された変数のデータを読み出し、ＤＴＣ用の転送バッファーにコピーします。 この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてください。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。 $最小通信間隔 = 1/(通信速度[bps]) \times 180 + 70[us]$ 通信速度が 1Mbps の際、上の式に数値を代入すると。 $最小通信間隔 = 1/(1[Mbps]) \times 180 + 70[us] = 250[us]$ ※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。
使用割り込み関数
下記の割り込みベクトルを使用しているため、ユーザーソフトの割り込みベクトルに下記の関数を登録してください。たとえば、UARTE4 を使用する場合、下記のように記述してください。 UARTE4 の場合 // UARTE4 RXI4, ERI4 #pragma interrupt INTLMA4IR R_UARTE4_Interrupt_Receive multi void R_UARTE4_Interrupt_Receive(void) { ics_int_sci_rxi(); } #pragma interrupt INTLMA4IS R_UARTE4_Interrupt_Error void R_UARTE4_Interrupt_Error(void) { ics_int_sci_eri(); }

3.7.3. V850E2M/Fx4 関数使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）ics_init()を下記のように呼び出す。

初期化関数 ics_init(ICS_UARTE4_P19_P110, 6) を初期化部分に入れてください。
第 1 パラメータは、使用するポートを ICS_<CPUNAME>.h から選択してください。
第 2 パラメータは、ICS で使用する割り込みレベルを設定してください。通常は、キャリア割り込みより低い優先順位に設定します。

---------- List 1 main.c ----------

```
void main(void)
{
    ics_init(ICS_UARTE4_P19_P110,   6);     /* Interrupt level 6            */
    while(1)
    {   nop();   }
}
```

2）ics_watchpoint()関数の組込み
このソフトでは、メインルーチンで ics_watchpoint()関数を呼び出していますが、通常は、キャリア割り込みの内部で呼び出します。
また、この関数は、250us 以上, 5ms 以下の間隔で呼び出すようにしてください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List2 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2 ics_watchpoint()の間引き ----------

```
int     deci = 0;

void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics_watchpoint();
        }
}
```

５）割り込みベクトルの修正
　割り込みベクトルが記述されたファイルを下記のように記述してください。

UARTE4 の場合
```
#pragma interrupt INTLMA4IR R_UARTE4_Interrupt_Receive multi
void R_UARTE4_Interrupt_Receive(void) { ics_int_sci_rxi(); }

#pragma interrupt INTLMA4IS R_UARTE4_Interrupt_Error
void R_UARTE4_Interrupt_Error(void) { ics_int_sci_eri(); }
```

UARTE5 の場合
```
#pragma interrupt INTLMA5IR R_UARTE5_Interrupt_Receive multi
void R_UARTE5_Interrupt_Receive(void) { ics_int_sci_rxi(); }

#pragma interrupt INTLMA5IS R_UARTE5_Interrupt_Error
void R_UARTE5_Interrupt_Error(void) { ics_int_sci_eri(); }
```

UARTE10 の場合
```
#pragma interrupt INTLMA10IR R_UARTE10_Interrupt_Receive multi
void R_UARTE10_Interrupt_Receive(void) { ics_int_sci_rxi(); }

#pragma interrupt INTLMA10IS R_UARTE10_Interrupt_Error
void R_UARTE10_Interrupt_Error(void) { ics_int_sci_eri(); }
```

UARTE11 の場合
```
#pragma interrupt INTLMA11IR R_UARTE11_Interrupt_Receive multi
void R_UARTE11_Interrupt_Receive(void) { ics_int_sci_rxi(); }

#pragma interrupt INTLMA11IS R_UARTE11_Interrupt_Error
void R_UARTE11_Interrupt_Error(void) { ics_int_sci_eri(); }
```

3.7.4. V850E2M/Fx4 ICS クロック

本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを PCLK=80MHz で使用する必要があります。

ICS ボード上のクロック周波数 = (PCLK / 10) MHz

例： PCLK = 80MHz の場合、 ICS CLOCK = 80/10 = 8.000MHz

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項：

ICS W1001 の場合（水晶発振器のソケットがないタイプ）
クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS W1003 の場合（水晶発振器のソケットがあるタイプ）
8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
可変クロックを内蔵しているため、PC 側からの操作が可能となります。

3.8. RX63T シリーズ

3.8.1. RX63T 使用資源

CPU 名	RX63T シリーズ共通
開発環境	CS+ Ver.3.00.00
Library version	Ver.2.0, Ver.2.1
通信レート	通信レート = $\frac{PCLK}{48}[Mbps]$ 標準通信レート PCLKB = 48MHz の場合 1.00Mbps
ステイタス	SCI0, SCI2, SCI3 サポート
ライブラリ種別	32bit ライブラリ
ライブラリ	ics_RX63T.obj
ヘッダファイル	ics_RX63T.h

CPU 使用リソース	対応 ICS / ICS++	サポート変数タイプ
・内部使用リソース SCI0 (PB2, PB1) INT SCI0 RXI INT SCI0 TXI DTC INT215 (TXI0) ICU.DTCER[215].BIT.DTCE SCI0 全て DTC 全て ICU.IPR[214].BYTE ICU.IER[0x1A].BIT.IEN6 ICU.IER[0x1A].BIT.IEN7 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRB.BIT.B31 MPC.PB2PFS.BYTE MPC.PB1PFS.BYTE PORTB.PMR.BIT.B2 = 1 PORTB.PMR.BIT.B1 = 1 外部ピン PB2: TXD0 PB1: RXD0 SCI2 (P02, P03) INT SCI2RXI INT SCI2TXI DTC INT221 (TXI2) ICU.DTCER[221].BIT.DTCE SCI2 全て DTC 全て ICU.IPR[220].BYTE ICU.IER[0x1B].BIT.IEN4 ICU.IER[0x1B].BIT.IEN5 SYSTEM.MSTPCRA.BIT.B28	○対応 ICS ハード ２種類の対応が可能です。 標準版 H/W model 1 H/W Ver. 1 S/W Ver. 1.22 以降 または、 H/W model 4 H/W Ver. 1 S/W Ver. 1.22 以降 以上は、ICS のハードを接続した状態で、ICS アプリの Help -> About で表示可能 ICS PC ソフト Ver. 2.5.0.0 以降 ICS++ PC ソフト	数値表示・設定 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 32bit IEEE754 浮動小数点 8bit BOOL 型 8bit LOGIC 型 波形表示 8bit 符号なし 整数型 8bit 符号あり 整数型 16bit 符号なし 整数型 16bit 符号あり 整数型 32bit 符号なし 整数型 32bit 符号あり 整数型 32bit IEEE754 浮動小数点

SYSTEM.MSTPCRB.BIT.B29 MPC.P02PFS.BYTE MPC.P03PFS.BYTE PORT0.PMR.BIT.B2 = 1 PORT0.PMR.BIT.B3 = 1 外部ピン P02: TXD2 P03: RXD2 Ver.2.01 からサポート SCI2 (PG0, PG1) INT SCI2RXI INT SCI2TXI DTC INT221 (TXI2) ICU.DTCER[221].BIT.DTCE SCI2 全て DTC 全て ICU.IPR[220].BYTE ICU.IER[0x1B].BIT.IEN4 ICU.IER[0x1B].BIT.IEN5 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRB.BIT.B29 MPC.PG1PFS.BYTE MPC.PG0PFS.BYTE PORTG.PMR.BIT.B1 = 1 PORTG.PMR.BIT.B0 = 1 外部ピン PG0: TXD2 PG1: RXD2 SCI3 (P35, P34) INT SCI3 RXI INT SCI3 TXI DTC INT224 (TXI3) ICU.DTCER[224].BIT.DTCE SCI3 全て DTC 全て ICU.IPR[223].BYTE ICU.IER[0x1B].BIT.IEN7 ICU.IER[0x1C].BIT.IEN0 SYSTEM.MSTPCRA.BIT.B28 SYSTEM.MSTPCRB.BIT.B28 MPC.P35PFS.BYTE MPC.P34PFS.BYTE PORT3.PMR.BIT.B5 = 1 PORT3.PMR.BIT.B4 = 1 外部ピン		

P35: TXD3 P34: RXD3		

3.8.2. RX63T 関数説明

Lib Ver.2.00 / Ver.2.01 on CS+ Ver.3.00.00
初期化関数の呼び出し void ics_init(void* addr, char port, char level);
本関数内部で、ピン定義を含む ICS 関連の初期化を行います。本関数の初期化後に、前項で記載された ICS で使用する資源ピンの定義や、スタンバイコントロールレジスタなどの設定を破壊しないように注意してください。 第１パラメータ DTC で使用する DTC のベクトルテーブルの先頭アドレスを指定してください。ics_init()関数を呼び出す前にユーザーが DTC ベクトルテーブルを確保する必要があります。このアドレスの下位 12bit は０である必要があります。 第２パラメータ SCI のポート番号や、SCI の使用するピンを設定します。このパラメータは、ICS_<CPUNAME>.h 内で定義されている文字列を使用してください。 第３パラメータ ICS で使用する SCI の割り込みレベルを指定します。下記の条件を満たすように設定してください。 最小間隔で 2ms の割り込みが発生する可能性があるため、システムとして、この割り込み間隔を許容できる割り込みレベルを設定してください。 SCI の受信割り込みが一番処理時間の長い割り込みです。10us 程度ですが、割り込み禁止時間を許容できない割り込みソースがある場合には、この割り込み設定レベルよりも高い割り込みレベルを設定してください。
転送関数の呼び出し void ics_watchpoint(void);
本転送関数は、データの転送セットアップ用の関数です。通常は、キャリア割り込み内部で置きます。ただし、サンプルソフトでは、記述方法をわかりやすくするために、メインルーチン内に記述しています。 本関数は、ＰＣで指定された変数のデータを読み出し、ＤＴＣ用の転送バッファーにコピーします。 この関数は、通信速度が 1Mbps の際最小 250us 以上、最大 5ms の間隔を保って呼び出すようにしてください。通信速度が 1Mbps 以外の場合には、次の式で定義される時間を置いて呼び出してください。 $最小通信間隔 = 1/(通信速度[bps]) \times 180 + 70[us]$ 通信速度が 1Mbps の際、上の式に数値を代入すると。 $最小通信間隔 = 1/(1[Mbps]) \times 180 + 70[us] = 250[us]$ ※注意：ユーザソフトウェアでの割り込み間隔は、他の割り込みの関係で、割り込みの発生が遅延する場合があります。割り込みタイミングがずれることも考慮して、呼び出すようにしてください。

使用割り込み関数

下記の割り込みベクトルを使用しているため、ユーザーソフトの割り込みベクトルに下記の関数を登録してください。
　ルネサス標準のコンパイラで自動的に生成されるプロジェクトを使用する場合、intprg.c という割り込み処理を記載したファイルに追記してください。

　たとえば、SCI0 を使用する場合、下記のように記述してください。
SCI0 の場合
```
void Excep_SCI0_RXI0(void){ ics_int_sci_rxi(); }
void Excep_SCI0_TXI0(void){ ics_int_sci_txi(); }
```

SCI1 の場合
```
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_TXI1(void){ ics_int_sci_txi(); }
```

SCI2 の場合
```
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
void Excep_SCI2_TXI2(void){ ics_int_sci_txi(); }
```

3.8.3. RX63T 関数使用方法

ICS を使用するためのユーザープログラムの設定方法を、付属のサンプルソフトを例に説明します。

１）開発環境に BDTCTBL セクションを確保する。

BDTCTBL のセクションを下位 12bit が０になるような RAM 上のアドレスに割り当てます。このアドレスを開発環境に設定してします。ここでは、0x3000 からに設定することにします。
E1 などのエミュレータを使用する場合、ユーザーRAM 領域もしくは、DTC テーブルの領域と、E1 エミュレータの使用領域とが重ならないようにしてください。正常に動作しないことがあります。

２）ユーザープログラムに、DTC テーブルを定義する。

ICS_sample.c の先頭に DTC テーブルの変数である unsigned long dtc_table[256]を定義する。

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section
```

３）ics_init()を下記のように呼び出す。

初期化関数 ics_init((void*)dtc_table, ICS_SCI0_PB2_PB1, 6) を初期化部分に入れてください。
第１パラメータは、１）で確保した BDTCTBL の先頭アドレスを入力してください。
第 2 パラメータは、使用するポートを ICS_<CPUNAME>.h から選択してください。
第 3 パラメータは、ICS で使用する割り込みレベルを設定してください。通常は、キャリア割り込みより低い優先順位に設定します。

---------- List 1 main.c ----------

```
#pragma section DTCTBL
unsigned long dtc_table[256];        // caution alignment 0x000
#pragma section

void main(void)
{
    ics_init((void*)dtc_table, ICS_SCI0_PB2_PB1, 6);    /* Interrupt level 6          */
    while(1)
    {   nop();   }
}
```

４）ics_watchpoint()関数の組込み

このソフトでは、メインルーチンで ics_watchpoint()関数を呼び出していますが、通常は、キャリア割り込みの内部で呼び出します。

また、この関数は、250us 以上, 5ms 以下の間隔で呼び出すようにしてください。割り込み処理関数が 250us 以下の間隔で呼び出される場合、List2 のように ics_watchpoint()の呼び出しを間引くようにソフトウェアを組んでください。

---------- List 2　ics_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics_watchpoint();
        }
}
```

５）intprg.c の修正

intprg.c の中の　RXI の割り込み関数から ics_int_sci_rxi()関数を呼び出すようにしてください。

SCI0 の場合

```
void Excep_SCI0_RXI0(void){ ics_int_sci_rxi(); }
void Excep_SCI0_TXI0(void){ ics_int_sci_txi(); }
```

SCI1 の場合

```
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_TXI1(void){ ics_int_sci_txi(); }
```

SCI2 の場合

```
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
void Excep_SCI2_TXI2(void){ ics_int_sci_txi(); }
```

3.8.4. RX63T ICS クロック

本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ＩＣＳボード上のクロックを下記のように選択してください。クロックを可変できないモデルの場合には、本ライブラリを PCLKB=48MHz で使用する必要があります。

ICS ボード上のクロック周波数 = (PCLK / 6) MHz

例： PCLK = 50MHz の場合、 ICS CLOCK = 50/6 = 8.333MHz
PCLK = 48MHz の場合、 ICS CLOCK = 48/6 = 8.000MHz

デスクトップラボでは、 標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

デスクトップラボでは、 W1003 用のクロック標準品として 8.000MHz, 8.333MHz, 10.000MHz の在庫を用意しております。

注意事項：

ICS W1002 の場合（水晶発振器のソケットがないタイプ）
クロックが固定されているため、8MHz 以外のクロックでの使用はできません。

ICS W1003 の場合（水晶発振器のソケットがあるタイプ）
8MHz 以外の ICS 用クロックを使用する場合、ボード上のソケットに実装されている水晶発振器を交換することにより、クロックを変更することが可能です。

ICS++ W1004 の場合（光ファイバー版）
可変クロックを内蔵しているため、PC 側からの操作が可能となります。

4. 改訂履歴

バージョン	変更日	変更内容
Ver.1.00	2013-08-12	・RX62T Ver2.0 ライブラリ　初版作成
Ver.1.01	2013-10-10	・ics_watchpoint()関数の最大呼び出し間隔の記述を追加
Ver.1.02	2013-11-06	・記述フォーマットを変更
Ver.1.03	2014-01-06	・RX111 のライブラリの記述を追加 ・RX62T, RX111 に対応ライブラリのバージョンを追加
Ver.1.04	2014-02-10	・RL78G14 対応ライブラリの記述を追加 ・RL78F14 対応ライブラリの記述を追加
Ver.1.05	2014-02-11	・誤記修正
Ver.1.06	2014-02-25	・RX63U 対応ライブラリ追加
Ver.1.07	2014-03-12	・RX64M 対応ライブラリ追加
Ver.1.08	2014-06-18	・RX63U の記述削除、V850E2M/Fx4 記述追加 ・新 ICS シリーズ W1004 の記述追加
Ver.1.09	2014-10-03	・RX63T シリーズの記述を追加
Ver.1.10		・欠番
Ver.1.11		・欠番
Ver.1.12	2014-12-26	・RX63T シリーズライブラリ・サポートポート追加
Ver.1.13	2015-10-14	・RL78G1F 追加

ICS　Library　function　manual

発行年月日　2015 年 10 月 14 日　　Ver.1.13JP

発行　　デスクトップラボ株式会社
〒192-0362 東京都八王子市松木３５－７事務所１０１