Desk Top Lab

ICS++ library Ver.3.6x
Function reference manual

RENESAS CS+ CC-RX compiler

Index

1. はじめに

1.1. はじめに

本ドキュメントは、ICS シリーズ W1001, W1002, W1003, T2001A/B、T2002A/B、および、ICS++ シリーズ W1004, W2001, W2002, T2001C, T2006 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、株式会社デスクトップラボの製品です。

2. ICS ハードウェアによる機能の違い

2.1. ICS / ICS++ シリーズの種類

ICS / ICS++シリーズには、下記のように、多くの種類が配布／販売されています。
下記の説明に応じて、シリーズ名を把握して以下の関数の説明をお読みください。

2.1.1. ICS++ W2002 シリーズ

光ファイバーで接続するタイプの新しい ICS++シリーズです。
0.5Mbps～8Mbps の範囲をサポートします。加えて、12ch モードをサポートしています。

図 1 W2002 ICS++

基板上に W2002 とシールで記載されています。初期出荷分の一部のロットには、シールがない物もあります。これらの場合には、基板上のシルクで記載された番号、もしくは、PC 上のソフト DTLScope で表示される型番で判別が可能です。
シルクでの判別： P00301-D1-009 と記載がある場合には、W2002 となります。

ファームウェアのバージョンにより、波形表示チャンネル数が異なります。

Frimware version	Mode 0	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6
1.0	○	○	○	○	×	×	○
1.2	○	○	○	○	○	×	○

モードは、ics2_init()関数で設定されるモードです。

2.1.2. T2001C / T2006A 搭載 サブセット ICS

T2001C / T2006A に搭載された ICS は、W2002 シリーズに分類されます。
W2002 との主な違いはメモリー長で、T2001C / T2006A との違いは 2 点あります。

1) レコード長が 1024 点まで
2) 波形表示チャンネル数が 8ch まで

以上のように機能が制限されています。

図 2 T2001C 低電圧インバータ（T2001B の後継機種）

図 3 T2006A 低電圧インバータ（三相インバータ ３ポート版）

2.1.3. ICS++ W1004 シリーズ（販売終了品）

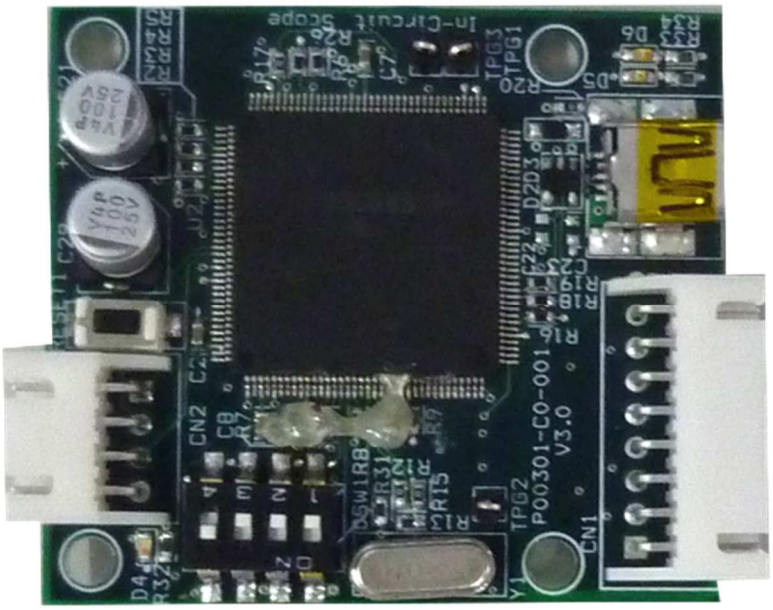
光ファイバーで接続するタイプの ICS++シリーズです。

このタイプの ICS++をお持ちの方で、Firmware が Ver.1.35 以前の方は、弊社にお送りしていただければ、無償で Ver.1.52 以降に Version UP させていただきます。Version UP することにより、ターゲット CPU の通信レートが 0.5Mbps～1.25Mbps までの範囲だったものが、0.5Mbps～1.25Mbps の範囲に加えて、1.5Mbps, 3Mbps をサポートできるようになります。

2.1.4. ICS W1001 シリーズ（販売終了品）

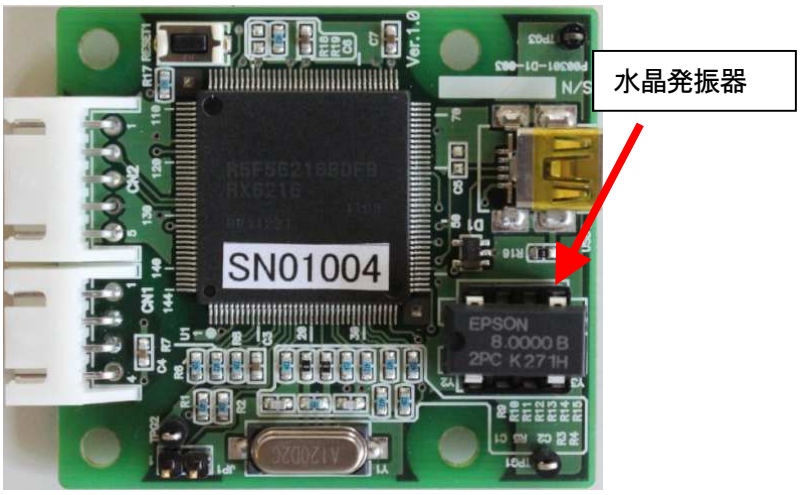
W1001

下記の写真のような、1Mbps 固定タイプの ICS です。



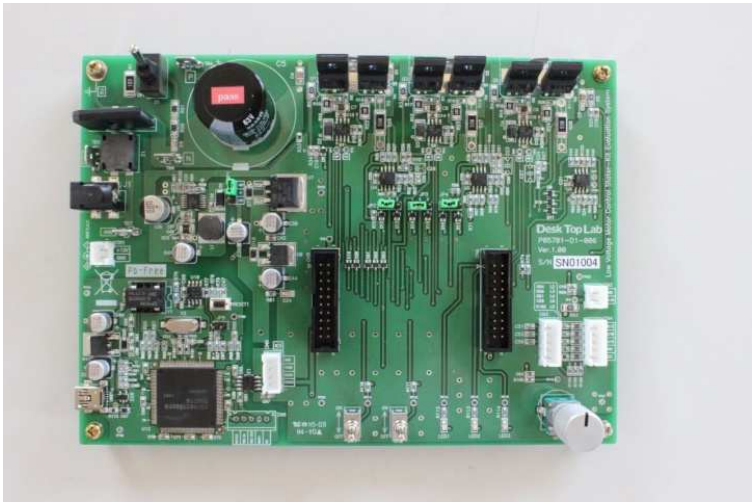
2.1.5. ICS W1003 シリーズ（販売終了品）

下記の写真のような、ボード上のソケットに実装された水晶発振器を交換して、通信レートを固定するタイプの ICS です。



2.1.6. T2001B / T2002B 搭載 サブセット ICS（販売終了品）

T2001B / T2002B に搭載された ICS は、W1003 シリーズに分類されます。
T2001B, T2002B に搭載された ICS は、W1003 のサブセットとなっています。
お試し版との位置づけのツールのため、レコード長が 1024 点までと非常に短くなっています。



2.1.7. ICS++ W2001 シリーズ （販売終了品）

光ファイバーで接続するタイプの新しい ICS++シリーズです。
0.5Mbps～3.2Mbps の範囲に加えて、3.75Mbps, 5Mbps をサポートします。
一般販売はしておりません。

2.2. 各シリーズの機能の差

表 1 各 ICS / ICS++仕様

	ICS series W1001 (販売終了)	ICS series W1003 T2002B (販売終了) T2001B (販売終了)	ICS++ series W1004 (販売終了)	ICS++ series W2002 (現行品) T2001C (現行品) T2006A (現行品)
通信レート	1Mbps 固定	0.5Mbps～1.25Mbps ボード上のクロックを交換することで変更可能	・Firm Ver.1.35 以前 0.5Mbps～1.25Mbps ・Firm Ver.1.52 以降 0.5Mbps～1.25Mbps 1.5Mbps, 3.0Mbps PC ソフトから変更可能	0.5Mbps～8Mbps PC ソフトから変更可能
チェンネル数	8ch	8ch	8ch	W2002: Firm V1.0 32bit 12ch 16bit 8ch W2002: Firm V1.2 32bit 12ch 16bit 15ch T2001C, T2006A は 8ch
絶縁の方法	IC による絶縁	IC による絶縁	**光ファイバー**	W2002: **光ファイバー** T2001C, T2006A は IC 絶縁
USB 転送速度	11Mbps	11Mbps	11Mbps	480Mbps
ロールモード 波形をスクロールしながら表示するモード	なし	なし	サポート (0.2 秒サンプル以上	**サポート** (0.2 **秒サンプル以上**)
信号発生機能 任意波形を CPU 上の変数に書き込んで、実機試験やデモ運転を行う機能	なし	なし	サポート DTLScope 1.5.使用時	**サポート** DTLScope 1.5.**使用時**
対応 PC soft	InCircuitScope DTLScope	InCircuitScope DTLScope	DTLScope.exe	DTLScope.exe

2.3. 転送レートの設定方法（重要）

ライブラリを使用する際には、転送レートを決定する必要があります。通常は、可能な限り早い通信レートに設定する方が良いのですが、使用する ICS/ICS++のハードウェアや、使用する CPU の種類やクロック周波数により制約を受けます。通常は、以下の手順で最も高速な通信レートを設定してください。

2.3.1. ICS / ICS++ ハードウェアによる制約

『表 1 各 ICS / ICS++仕様』に示されるように、各ハードウェアにより、通信可能な最高転送レートが異なります。この制約の範囲内になるように、通信レートを設定してください。

2.3.2. ターゲット CPU／クロックによる制約

各 CPU や、実際に使用するクロック周波数、ライブラリのバージョンにより、設定可能な周波数が飛び飛びに存在しています。例えば、RX23T の場合、以下のようになります。

$$\text{通信レート} = \frac{PCLKB}{8\times(speed+1)}[Mbps]$$

ここで、PCLKB は、実際に使用する RX23T のクロック周波数。speed は、0 以上の整数値です。

2.4. 実際のシステムにおける通信レート設定例

例 A） RX23T PCLKB = 40MHz の場合、
通信レートは、以下表のようになります。

Speed	通信レート
0	5Mbps
1	2.5Mbps
2	1.67Mbps
3	1.25Mbps
4	1Mbps
5	0.833Mbps

W1003 の場合
0.5Mbps～1.25Mbps が選択できるので、1Mbps を選択します。
W1004 Firmware V1.35 以前 の場合
0.5Mbps～1.25Mbps が選択できるので、1.25Mbps を選択します。
W1004 Firmware V1.52 以降 の場合
0.5Mbps～1.25Mbps, 1.5Mbps, 3Mbps が選択できるので、1.25Mbps を選択します。
W2002 の場合、
0.5Mbps～8Mbps が選択できるので、5Mbps を選択します。

例 B） RX23T PCLKB = 32MHz の場合
通信レートは、以下の表のように設定されます。

Speed	通信レート
0	4Mbps
1	2Mbps
2	1.33Mbps
3	1Mbps
4	0.8Mbps
5	0.67Mbps

W1003 の場合
0.5Mbps～1.25Mbps が選択できるので、1Mbps を選択します。
W1004 Firmware V1.35 以前 の場合
0.5Mbps～1.25Mbps が選択できるので、1Mbps を選択します。
W1004 Firmware V1.52 以降 の場合

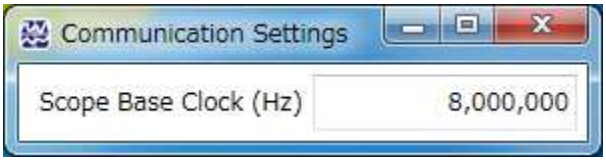
0.5Mbps～1.25Mbps, 1.5Mbps, 3Mbps が選択できるので、1Mbps を選択します。
W2001 の場合
0.5Mbps～3.2Mbps, 3.75Mbps, 5Mbps が選択できるので、2Mbps を選択します。
W2002 の場合、
0.5Mbps～8Mbps が選択できるので、4Mbps を選択します。

2.5. 通信レート決定用クロックの ICS++ハードウェアへの設定方法

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。

2.5.1. W1004, W2001, W2002, T2001C, T2006A の場合

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。
PC ソフト(DTLScope.exe)で通信レートの８倍の周波数を設定してください。
DTLScope.exe を立ち上げ、
Settings -> Communication Settings
をクリックすると、下記のようなウィンドウが表示されます。
下記に、通信レートの８倍の数値を入力してください。



2.5.2. ICS W1001 の場合（ケースがない ICS、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、通信クロック 8MHz 以外のクロックでの使用はできません。

2.5.3. ICS W1003 の場合（ケースがない ICS、水晶発振器のソケットがあるタイプ）

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。通信レートの８倍の周波数の水晶発振器モジュールに交換してください。
設定するクロック周波数の計算方法は、下記の通りです
通信レートを 1.25Mbps 以下に設定する必要があります。
選択したクロックの８倍の水晶発振子をボード上の水晶発振器と交換してください。
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。

推奨品は、EPSON SG-8002DC 3.3V タイプです。
この推奨品は、Digikey で購入可能です。周波数の指定が可能です。

3. ICS++ライブラリの基本仕様・動作

3.1. 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。

3.2. 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 ics2_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。ICS++のモデルにより性能が異なるため、制約も異なります。モデルの確認方法は、DTLScope を立ち上げた時のステイタスバーに表示される名称です。

ICS++シリーズ　W2001, W2002, T2001C, T2006A の場合
例：
　最小 210us　(通信レート 1Mbps の場合)
　最小 66us　(通信レート 5Mbps の場合)
　　最小時間 = 180/(通信レート[Mbps]) + 30 [us]　　（最大通信レートは 8Mbps）
　最大　5ms

旧製品 ICS：W1001, W1003, T2001B の場合、ICS++シリーズ　W1004 の場合
例：
　最小　250us　(@W1003, W1004 通信レート 1Mbps の場合)　最大通信レートは、1.25Mbps
　　最小時間 = 180/(通信レート[Mbps]) + 70 [us]
　最大　5ms

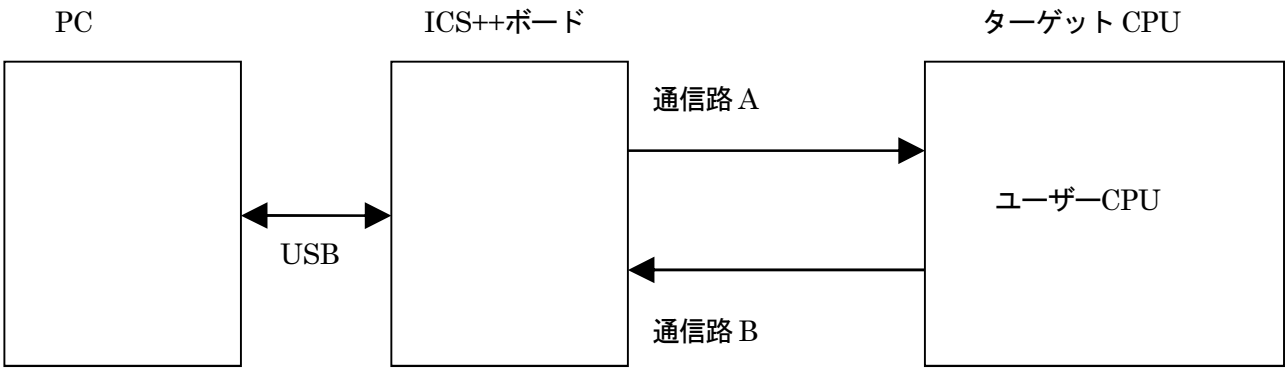


図 4　ICS++システムの通信路

本 ICS++には、データ転送間隔の制限があります。本制約は、図 4 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　ics2_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。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、ics2_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。

3.3.　転送モード 0, 1, 2, 3, 4, 5, 6 の違い

ICS++には、2019 年 7 月現在、7 種類の転送モードが存在しています。以下、mode0, mode1, mode2, mode3,mode4, mode5, mode6 と呼びます。これらのモードの違いは、波形表示でサポートする最大ビット長と、１サンプリングのデータを何回の ics2_watchpoint()関数で転送するかの差です。（将来、この転送モードは拡張される予定があります）

１） mode 0　（8 / 16 ビット 8 チャンネル　１回転送モード動作）
　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。通常、32bit　CPU ではサポートしません。全 ICS モデルで使用可能です。

２）mode 1　（8 / 16 / 32 ビット 8 チャンネル　2 回転送モード動作）
　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。ics2_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 ics2_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８チャンネルモードの場合には、２回の ics2_watchpoint()関数により、１回の８ｃｈ分の転送が行われます。全 ICS モデルで使用可能です。

３）mode 2　（8 / 16 / 32 ビット 4 チャンネル　１回転送モード動作）
　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。ics2_watchpoint()関数が呼ばれると、波形表示に関しては、毎回、指定された４ｃｈ分の８ビット、１６ビット、３２ビットデータを取り込みます。そして、その 4ch 分のデータを転送します。
　つまり、３２ビット４チャンネルモードの場合には、１回の ics2_watchpoint()関数呼び出しにより、１回の４ｃｈ分の転送が行われます。５チャンネル以上の波形表示の機能はありません。
※注意　このモードは、W1001, W1003、T2001A/B, T2002A/B ではサポートされていません。W1004, W2001, W2002, T2001C, T2006A のモデルで使用可能です。

４）mode 3　（8 / 16 / 32 ビット 12 チャンネル　３回転送モード動作）
　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。ics2_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１２ｃｈ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 ics2_watchpoint()関数が呼ばれた時、データを取り込まず、未転送のデータの内の残り 4ch 分のデータを転送します。さらに次に ics2_watchpoint()関数が呼ばれた時に、最後の 4ch 分のデータを転送します。
　つまり、３２ビット１２チャンネルモードの場合には、３回の ics2_watchpoint()関数により、１回の８ｃｈ分の転送が行われます。
※注意　このモードは、W1001, W1003、W1004、W2001, T2001A/B, T2002A/B ではサポートされていません。W2002 モデルで使用可能です。（T2001C, T2006A では、8ch 分のみ使用可能です）

４）mode 4　（ 8 / 16 ビット 15 チャンネル　２回転送モード動作）
　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。ics2_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１５ｃｈ分の８ビット、１６ビットデータを取り込みます。さらに、8ch 分のデータを転送します。次に ics2_watchpoint()関数が呼ばれた時、データを取り込まず、未転送のデータの内の残り 7ch 分のデータを転送します。

つまり、１６ビット１５チャンネルモードの場合には、２回の ics2_watchpoint()関数により、１回の１５ｃｈ分の転送が行われます。

※注意　このモードは、W2002 の Firmware Ver.1.2 以降のバージョンのみでサポートされます。

５）mode 5　（将来の予約）

４）mode 6　（16 ビットのみ 8 チャンネル　１回転送モード動作）

このモードは mode 0 とほぼ同じですが、８ビットの変数に対する波形表示をサポートしません。そのかわりに、ics2_watchpoint()関数の実行時間が短くなっています。

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。ics2_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１５ｃｈ分の１６ビットデータを取り込みます。さらに、8ch 分のデータを転送します。次に ics2_watchpoint()関数が呼ばれた時、データを取り込まず、未転送のデータの内の残り 7ch 分のデータを転送します。

※注意　このモードは、全 ICS モデルでサポートされます。

表 2 転送モード

	メリット	デメリット
8/16bit 8ch 1 time mode （モード０）	波形情報更新間隔が短い 8 チャンネルの波形表示が可能	32bit の波形表示ができない
8/16/32bit 8ch 2 times mode （モード１）	32bit の波形表示が可能 8 チャンネルの波形表示が可能	波形情報更新間隔が 16bit の２倍
8/16/32bit 4ch 1 time mode （モード２）	32bit の波形表示が可能 波形更新間隔が短い	4 チャンネルしか波形を表示できない 16bit モードと同じ間隔
8/16/32bit 12ch 3 times mode （モード３）	32bit の波形表示が可能 12 チャンネルの波形表示が可能	波形情報更新間隔がモード１の 1.5 倍
8/16bit 15ch 2 times mode （モード４）	最大 15ch の波形表示が可能。	32bit の波形表示ができない
将来の予約 （モード５）		
16bit 8ch 1 times mode （モード６）	波形情報更新間隔が短い 8 チャンネルの波形表示が可能	8bit, 32bit の波形表示ができない。

16bit CPU　例：RL78 シリーズの標準のライブラリでは、Mode 1/2/3/5 をサポートしません。
32bit CPU　例：RX シリーズの標準のライブラリでは、Mode 0/4/6 をサポートしません。s

3.4. 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。

3.5. 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
ics2_<CPUNAME>.h
ics2_<CPUNAME>.lib

通常関数として、
void ics2_init(void* addr, uint8_t unitpin, uint8_t level, uint8_t speed, uint8_t mode);
void ics2_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、DMA を使用する場合、コンパイルスイッチが標準と異なる場合など、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。
デスクトップラボにお問い合わせください。有償での対応が可能です。

※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。

4. 使用資源・ライブラリの説明

4.1. RX13T シリーズ (CC コンパイラ)

4.1.1. RX13T 使用資源

CPU 名	RX13T シリーズ
開発環境	CS+ Ver.8.03.00　CC-RX 3.01.00
Library version	Ver.3.60
通信レート	ライブラリの設定可能通信レート 0.5Mbps～4Mbps $通信レート = \frac{PCLKB}{8 \times (speed + 1)} [Mbps]$　$speed \in [0, 1, 2, \ldots 249]$ 標準通信レート　PCLKB = 32MHz,　speed=0　の場合 4Mbps ※ご使用になる ICS / ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	#define　ICS_SCI1_PD3_PD5　(0x10) #define　ICS_SCI1_PB6_PB7　(0x11) #define　ICS_SCI5_PB6_PB7　(0x50) #define　ICS_SCI5_PB2_PB1　(0x51) #define　ICS_SCI5_P23_P24　(0x52) #define　ICS_SCI12_PB0_P94　(0xC0)
ライブラリ	ics2_RX13T.lib
ヘッダファイル	ics2_RX13T.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE SCIx　全て DTC　全て ICU　対応する部分 SYSTEM.MSTPCRB　対応する部分 MPC　対応する部分 PORT　対応する部分	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

4.1.2. RX13T 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);
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。ics2_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、ICS2_<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 ICS で使用する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 ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。 $通信レート = \frac{PCLKB}{8 \times (speed + 1)} [Mbps] \quad speed \in [0, 1, 2, \ldots 249]$ ただし、例外として： ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。 各機種のサポート範囲に収まるように speed の値を設定します。 第５パラメータ 通信モードの設定 ０：設定禁止（将来の予約） １：32bit８チャンネル同時サンプリング・２回転送モード （２回の ics2_watchpoint()関数呼び出しで１回のサンプリング） ２：32bit４チャンネル同時サンプリング・１回転送モード （１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） ※注意：W1001, W1003 では使用できません。 ３：32bit１２チャンネル同時サンプリング・３回転送モード （１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） ※注意：W1001, W1003, W1004 では使用できません。 通常は、１を設定します。波形更新レートを速くしたい場合には２を設定します。ただし、波形表示のチャンネル数は、４チャンネルになります。

転送関数の呼び出し　　void　　ics2_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。

ICS+　W2001, W2002、T2001C, T2006A の場合

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 30[us]$$

ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合

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

通信速度が 1Mbps の場合、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。
※注意 : DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。

使用割り込み関数

割り込み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。

○CS+上で動作する CC-RL 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。
例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出しを追記してください。

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

○SmartConfigurator を使用してコードを生成する場合
SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、ERI, TEI, TXI は、空の関数で結構です。
ベクトル番号は、SCI のポートによって異なるため、注意してください。

```
#pragma interrupt (Excep_SCI5_ERI5(vect=222))
#pragma interrupt (Excep_SCI5_RXI5(vect=223))
#pragma interrupt (Excep_SCI5_TXI5(vect=224))
#pragma interrupt (Excep_SCI5_TEI5(vect=225))
void Excep_SCI5_ERI5(void){ /* no code */ }
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
void Excep_SCI5_TXI5(void){ /* no code */ }
void Excep_SCI5_TEI5(void){ /* no code */ }
```

4.1.3. RX13T 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

A） DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x02000
uint32_t dtc_table[256];
```

B） 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

```
#pragma section DTCTBL
uint32_t dtc_table[256];        // caution alignment 0x000
#pragma section
```

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

２）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI1_PB6_PB7, 6, 0, 1) 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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。通常は、１にしてください。

---------- List 1　main.c ----------

```
#pragma address dtc_table=0x02000
uint32_t dtc_table[256];

void main(void)
{
      ics2_init(dtc_table, ICS_SCI1_PB6_PB7, 6, 0, 1);
      while(1)
      {   nop();   }
}
```

３）ics2_watchpoint()関数の組込み

　このソフトでは、メインルーチンで ics2_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、各 ICS ユニットで可能な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指定間隔で呼び出される場合、List2 のように ics2_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　ics2_watchpoint()の間引き ----------

```
int     deci = 0;

void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_watchpoint();
        }
}
```

４）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。

SCI1 の場合

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

SCI5 の場合

```
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
```

※注意

SmartConfigurator を使用している場合、TXI, TEI, ERI の割込み関数の説明部に記載の通り、空の関数と#prgma 宣言も追加してください。

4.2. RX23T シリーズ (CC コンパイラ)

4.2.1. RX23T 使用資源

CPU 名	RX23T シリーズ
開発環境	CS+ Ver.6.00.00　CC-RX 2.07.00
Library version	Ver.3.60
通信レート	ライブラリの設定可能通信レート 0.5Mbps～5Mbps $通信レート = \frac{PCLKB}{8\times(speed+1)}[Mbps]$　　speed ∈[0, 1, 2, ..255] 標準通信レート　PCLKB = 40MHz,　speed=3　の場合 1.25Mbps PCLKB = 32MHz,　speed=3　の場合 1.00Mbps ご使用になる ICS / ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	SCI1　TXD1:PD3,　RXD1:PD5 SCI5　TXD5:PB5,　RXD5:PB6 SCI5　TXD5:PB2,　RXD5:PB1
ライブラリ	ics2_RX23T.lib
ヘッダファイル	ics2_RX23T.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE SCIx　全て DTC　全て ICU　対応する部分 SYSTEM.MSTPCRB　対応する部分 MPC　対応する部分 PORT　対応する部分	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

4.2.2. RX23T 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);
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。ics2_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、ICS2_<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 ICS で使用する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 　ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。 $$通信レート = \frac{PCLKB}{8\times(speed+1)}[Mbps]$$ ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。 各機種のサポート範囲に収まるように speed の値を設定します。 第５パラメータ 　通信モードの設定 　０：設定禁止（将来の予約） 　１：32bit８チャンネル同時サンプリング・２回転送モード 　　（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　２：32bit４チャンネル同時サンプリング・１回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003 では使用できません。 　３：32bit１２チャンネル同時サンプリング・３回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003, W1004 では使用できません。 　４：設定禁止（将来の予約） 通常は、１を設定します。波形更新レートを速くしたい場合には２を設定します。ただし、波形表示のチャンネル数は、４チャンネルになります。

転送関数の呼び出し　　void　　ics2_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。

ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合

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

通信速度が 1Mbps の場合、上の式に数値を代入すると。

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

ICS+　W2001, W2002、T2001C, T2006A の場合

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 30[us]$$

※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。
※注意 : DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。

使用割り込み関数

割り込み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。

○CS+上で動作する CC-RL 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。
例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出し、ERI 割込みに ics_int_sci_eri()を追記してください。

```
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
```

○SmartConfigurator を使用してコードを生成する場合
SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、ERI, TEI, TXI は、空の関数で結構です。
ベクトル番号は、SCI のポートによって異なるため、注意してください。

```
#pragma interrupt (Excep_SCI5_ERI5(vect=222))
#pragma interrupt (Excep_SCI5_RXI5(vect=223))
#pragma interrupt (Excep_SCI5_TXI5(vect=224))
#pragma interrupt (Excep_SCI5_TEI5(vect=225))
void Excep_SCI5_ERI5(void){ ics_int_sci_eri(); }
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
void Excep_SCI5_TXI5(void){ /* no code */ }
void Excep_SCI5_TEI5(void){ /* no code */ }
```

4.2.3. RX23T 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

C） DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x02000
uint32_t dtc_table[256];
```

D） 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

```
#pragma section DTCTBL
uint32_t dtc_table[256];        // caution alignment 0x000
#pragma section
```

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

２）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI5_PB5_PB6, 6, 0, 1）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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。通常は、１にしてください。

---------- List 1　main.c ----------

```
#pragma address dtc_table=0x02000
uint32_t dtc_table[256];

void main(void)
{
      ics2_init(dtc_table, ICS_SCI5_PB5_PB6, 6, 0, 1); // CN3
      while(1)
      {   nop();   }
}
```

３）ics2_watchpoint()関数の組込み

　このソフトでは、メインルーチンで ics2_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、各 ICS ユニットで可能な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指定間隔で呼び出される場合、List2 のように ics2_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　ics2_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_watchpoint();
        }
}
```

４）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。

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

SCI1 の場合

```
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
```

SCI5 の場合

```
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
void Excep_SCI5_ERI5(void){ ics_int_sci_eri(); }
```

※注意

SmartConfigurator を使用している場合、TXI, TEI, ERI の割込み関数の説明部に記載の通り、空の関数と #prgma 宣言も追加してください。

4.3. RX24T シリーズ （CC コンパイラ ）

4.3.1. RX24T 使用資源

CPU 名	RX24T シリーズ
開発環境	CS+ Ver.6.00.00　CC-RX 2.07.00
Library version	Ver.3.60
通信レート	ライブラリの設定可能通信レート 0.5Mbps～5Mbps $通信レート = \frac{PCLKB}{8\times(speed+1)}[Mbps]$　　speed ∈[0, 1, 2, ..255] 標準通信レート　PCLKB = 40MHz,　speed=4　の場合　1.00Mbps PCLKB = 32MHz,　speed=3　の場合　1.00Mbps ご使用になる ICS / ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	SCI1　TXD1:PD3,　RXD1:PD5 SCI5　TXD5:PB5,　RXD5:PB6 SCI6　TXD6:PB2,　RXD6:PB1 SCI6　TXD6:PB0,　RXD6:PA5 SCI6　TXD6:P81,　RXD6:P80
ライブラリ	ics2_RX24T.lib
ヘッダファイル	ics2_RX24T.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE SCIx　全て DTC　全て ICU:対応する部分 SYSTEM.MSTPCRB. 対応する部分 MPC　対応する部分 PORTB　対応する部分	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

4.3.2. RX24T 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);
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。ics2_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、ICS2_<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 ICS で使用する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 　ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。 $$通信レート = \frac{PCLKB}{8\times(speed+1)}[Mbps]$$ ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。 各機種のサポート範囲に収まるように speed の値を設定します。 第５パラメータ 　通信モードの設定 　０：設定禁止（将来の予約） 　１：32bit８チャンネル同時サンプリング・２回転送モード 　　（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　２：32bit４チャンネル同時サンプリング・１回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003 では使用できません。 　３：32bit１２チャンネル同時サンプリング・３回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003, W1004 では使用できません。 　４：設定禁止（将来の予約） 通常は、１を設定します。波形更新レートを速くしたい場合には２を設定します。ただし、波形表示のチャンネル数は、４チャンネルになります。

転送関数の呼び出し　　void　　ics2_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。

ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合

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

通信速度が 1Mbps の場合、上の式に数値を代入すると。

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

ICS+　W2001, W2002、T2001C, T2006A の場合

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 30[us]$$

※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。
※注意 : DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。

使用割り込み関数

割り込み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。

○CS+上で動作する CC-RL 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。
例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出し、ERI 割込みに ics_int_sci_eri()を追記してください。

```
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
```

○SmartConfigurator を使用してコードを生成する場合
SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、ERI, TEI, TXI は、空の関数で結構です。
ベクトル番号は、SCI のポートによって異なるため、注意してください。

```
#pragma interrupt (Excep_SCI5_ERI5(vect=222))
#pragma interrupt (Excep_SCI5_RXI5(vect=223))
#pragma interrupt (Excep_SCI5_TXI5(vect=224))
#pragma interrupt (Excep_SCI5_TEI5(vect=225))
void Excep_SCI5_ERI5(void){ ics_int_sci_eri(); }
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
void Excep_SCI5_TXI5(void){ /* no code */ }
void Excep_SCI5_TEI5(void){ /* no code */ }
```

4.3.3. RX24T 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

A） DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x02000
uint32_t dtc_table[256];
```

B） 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

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

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

２）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI5_PB5_PB6, 6, 0 1) 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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。通常は、１にしてください。

```
------------ List 1   main.c ------------------------------------------------
#pragma address dtc_table=0x02000
uint32_t dtc_table[256];

void main(void)
{
      ics2_init((void*)dtc_table, ICS_SCI5_PB5_PB6, 6, 0, 1);     /* Interrupt level 6            */
      while(1)
      {   nop();   }
}
```

３）ics2_watchpoint()関数の組込み

このソフトでは、メインルーチンで ics2_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、各 ICS ユニットで可能な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指定間隔で呼び出される場合、List2 のように ics2_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   ics2_watchpoint()の間引き  -------------------------
int     deci = 0;


void    int_TM0(void)     /* 100us間隔  */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_watchpoint();
        }
}
```

４）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。
intprg.c の中の　ERI の割り込み関数から ics_int_sci_eri()関数を呼び出すようにしてください。

SCI1 の場合

```
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
```

※注意

SmartConfigurator を使用している場合、TXI, TEI, ERI の割込み関数の説明部に記載の通り、空の関数と#prgma 宣言も追加してください。

4.4. RX62T シリーズ(CC コンパイラ)

4.4.1. RX62T 使用資源

CPU 名	RX62T シリーズ
開発環境	CS+ Ver.6.00.00　CC-RX 2.07.00
Library version	Ver.3.60
通信レート	ライブラリの設定可能通信レート 0.5Mbps～3.125Mbps $通信レート = \frac{PCLKB}{16 \times (speed + 1)} [Mbps]$　speed ∈[0,1, 2,,,255] 標準通信レート　PCLKB = 48MHz,　speed=2　の場合 1Mbps ご使用になる ICS/ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	SCI0　TXD0:PB2,　RXD0:PB1 SCI1　TXD1:PD3,　RXD1:PD5 SCI2　TXD2:PB5,　RXD2:PB6 SCI2　TXD2:P81,　RXD2:P80
ライブラリ	ics2_RX62T.lib
ヘッダファイル	ics2_RX62T.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE SCIx　全て DTC　全て ICU:対応する部分 SYSTEM.MSTPCRB. 対応する部分 MPC　対応する部分 PORTx　対応する部分	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

4.4.2. RX62T 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);

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。ics2_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、ics2_<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 ICS で使用する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

ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。

$$\text{通信レート} = \frac{PCLKB}{16 \times (speed + 1)} [Mbps]$$

ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。
各機種のサポート範囲に収まるように speed の値を設定します。

第５パラメータ

通信モードの設定

第５パラメータ
通信モードの設定

０：設定禁止（将来の予約）
１：32bit８チャンネル同時サンプリング・２回転送モード
（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング）
２：32bit４チャンネル同時サンプリング・１回転送モード
（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）
※注意：W1001, W1003 では使用できません。
３：32bit１２チャンネル同時サンプリング・３回転送モード
（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）
※注意：W1001, W1003, W1004 では使用できません。
４：設定禁止（将来の予約）

転送関数の呼び出し　　void　　ics2_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、下記の条件に適合するように呼び出すようにしてください。

ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合

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

通信速度が 1Mbps の場合、上の式に数値を代入すると。

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

ICS+　W2001, W2002、T2001C, T2006A の場合

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 30[us]$$

※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。
※注意 : DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。

使用割り込み関数

割り込み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。

○CS+上で動作する CC-RL 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。
例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出し、ERI 割込みに ics_int_sci_eri()を追記してください。

```
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_ERI1(void){ ics_int_sci_eri(); }
```

○SmartConfigurator を使用してコードを生成する場合
SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、ERI, TEI, TXI は、空の関数で結構です。
ベクトル番号は、SCI のポートによって異なるため、注意してください。

```
#pragma interrupt (Excep_SCI2_ERI2(vect=222))
#pragma interrupt (Excep_SCI2_RXI2(vect=223))
#pragma interrupt (Excep_SCI2_TXI2(vect=224))
#pragma interrupt (Excep_SCI2_TEI2(vect=225))
void Excep_SCI2_ERI2(void){ ics_int_sci_eri(); }
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
void Excep_SCI2_TXI2(void){ /* no code */ }
void Excep_SCI2_TEI2(void){ /* no code */ }
```

4.4.3. 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

A） DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x03000
uint32_t dtc_table[256];
```

B） 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

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

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

２）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI0_PB2_PB1, 6, 2, 1）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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。１にしてください。

---------- List 1　main.c ----------

```
#pragma address dtc_table=0x03000
uint32_t dtc_table[256];

void main(void)
{
      Ics2_init((void*)dtc_table, ICS_SCI0_PB2_PB1, 6, 2, 1);      /* Interrupt level 6            */
      while(1)
      {    nop();    }
}
```

３）ics2_watchpoint()関数の組込み

　このソフトでは、メインルーチンで ics2_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 ics2_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。（呼出し間隔の 250us は、W1004 以前を使用し、通信レート 1Mbps の場合です）

---------- List 2　ics2_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)      /* 100us間隔  */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_watchpoint();
        }
}
```

４）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。

intprg.c の中の　ERI の割り込み関数から ics_int_sci_eri()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_ERI0(void){ ics_int_sci_eri(); }
```

※注意

SmartConfigurator を使用している場合、TXI, TEI, ERI の割込み関数の説明部に記載の通り、空の関数と #prgma 宣言も追加してください。

4.5. RX63T シリーズ(CC コンパイラ)

4.5.1. RX63T 使用資源

CPU 名	RX63T シリーズ
開発環境	CS+ Ver.6.00.00　CC-RX 2.07.00
Library version	Ver.3.60
通信レート	ライブラリの設定可能通信レート 0.5Mbps～3.125Mbps $通信レート = \frac{PCLKB}{16 \times (speed + 1)} [Mbps]$　speed ∈[0, 1, 2, ..255] 標準通信レート　PCLKB = 48MHz,　speed=2　の場合 1Mbps ご使用になる ICS/ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	SCI0　TXD0:PB2,　RXD0:PB1 SCI0　TXD0:P30,　RXD0:P24 SCI0　TXD0:PA4,　RXD0:PA5 SCI0　TXD0:P23,　RXD0:P22 SCI1　TXD1:PD3,　RXD1:PD5 SCI1　TXD1:P94,　RXD1:P93 SCI1　TXD1:PF3,　RXD1:PF2 SCI1　TXD1:P95,　RXD1:P96 SCI2　TXD2:P02,　RXD2:P03 SCI2　TXD2:PG0,　RXD2:PG1 SCI2　TXD2:PA1,　RXD2:PA2 SCI3　TXD3:P35,　RXD3:P34 SCI3　TXD3:PG3,　RXD3:PG4 SCI12　TXD12:PB5,　RXD12:PB6　// 64, 48pin 版は未対応 SCI12　TXD12:P81,　RXD12:P80
ライブラリ	ics2_RX63T.lib
ヘッダファイル	ics2_RX63T.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE SCIx　全て DTC　全て ICU:対応する部分 SYSTEM.MSTPCRB. 対応する部分 MPC　対応する部分 PORTx　対応する部分	数値表示・設定 8bit 符号なし／/符号あり　整数型 16bit 符号なし／符号あり　整数型 32bit 符号なし／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／符号あり　整数型 16bit 符号なし／符号あり　整数型 32bit 符号なし／符号あり　整数型 32bit IEEE754 浮動小数点

4.5.2. RX63T 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);
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。ics2_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、ics2_<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 ICS で使用する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 　ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。 $$通信レート = \frac{PCLKB}{16 \times (speed + 1)} [Mbps]$$ ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。 各機種のサポート範囲に収まるように speed の値を設定します。 第５パラメータ 　通信モードの設定 　０：設定禁止（将来の予約） 　１：32bit８チャンネル同時サンプリング・２回転送モード 　　（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　２：32bit４チャンネル同時サンプリング・１回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003 では使用できません。 　３：32bit１２チャンネル同時サンプリング・３回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003, W1004 では使用できません。 　４：設定禁止（将来の予約） 通常は、１を設定します。波形更新レートを速くしたい場合には２を設定します。ただし、波形表示のチャンネル数は、４チャンネルになります。

転送関数の呼び出し　　void　　ics2_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、下記の条件に適合するように呼び出すようにしてください。

ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合

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

通信速度が 1Mbps の場合、上の式に数値を代入すると。

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

ICS+　W2001, W2002、T2001C, T2006A の場合

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 30[us]$$

※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。
※注意 : DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。

使用割り込み関数

割り込み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。

○CS+上で動作する CC-RL 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。
例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出しを追記してください。

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

○SmartConfigurator を使用してコードを生成する場合
SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、TEI, TXI は、空の関数で結構です。
ベクトル番号は、SCI のポートによって異なるため、注意してください。

```
#pragma interrupt (Excep_SCI2_RXI2(vect=220))
#pragma interrupt (Excep_SCI2_TXI2(vect=221))
#pragma interrupt (Excep_SCI2_TEI2(vect=222))
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
void Excep_SCI2_TXI2(void){ /* no code */ }
void Excep_SCI2_TEI2(void){ /* no code */ }
```

4.5.3. 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

A） DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x03000
uint32_t dtc_table[256];
```

B） 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

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

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

３）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI0_PB2_PB1, 6, 0, 1) 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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。通常は、１にしてください。

---------- List 1　main.c ----------

```
#pragma address dtc_table=0x03000
uint32_t dtc_table[256];

void main(void)
{
      ics2_init((void*)dtc_table, ICS_SCI0_PB2_PB1, 6, 0, 1);      /* Interrupt level 6              */
      while(1)
      {    nop();    }
}
```

４）ics2_watchpoint()関数の組込み

　このソフトでは、メインルーチンで ics2_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、通信レート、ICS のモデルに応じた時間以上, 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上記規定時間以下の間隔で呼び出される場合、List2 のように ics2_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　ics2_watchpoint()の間引き ----------

```
int     deci = 0;

void    int_TM0(void)      /* 100us間隔  */

{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_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(); }
```

※注意

SmartConfigurator を使用している場合、TXI, TEI の割込み関数の説明部に記載の通り、空の関数と #prgma 宣言も追加してください。

4.6. RX66T シリーズ (CC コンパイラ)

4.6.1. RX66T 使用資源

CPU 名	RX66T シリーズ
開発環境	CS+ Ver.8.00.00　CC-RX 3.00.00　(Compile by RX V3 core 対応版)
Library version	Ver.3.60
通信レート	ライブラリの設定可能通信レート 0.5Mbps～7.5Mbps SCI1, SCI5, SCI6, SCI8, SCI9, SCI12 PCLKB= 40MHz @CPU CLK=160MHz　(PCLKB max 60MHz) $通信レート = \frac{PCLKB}{8 \times (speed + 1)} [Mbps]$　$speed \in [0, 1, 2, \ldots 255]$ 標準通信レート　PCLKB = 40MHz,　speed=4　の場合　1.00Mbps PCLKB = 40MHz,　speed=0　の場合　5.00Mbps ※SCI11 は、本ライブラリではサポートしておりません。 ※ご使用になる ICS / ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	#define　ICS_SCI1_PD3_PD5　(0x10) #define　ICS_SCI5_PD7_PE0　(0x50) #define　ICS_SCI5_PB5_PB6　(0x51) #define　ICS_SCI6_PB0_PB1　(0x60) #define　ICS_SCI8_PD0_PD1　(0x80) #define　ICS_SCI8_PC1_PC0　(0x81) #define　ICS_SCI8_PA4_PA5　(0x82) #define　ICS_SCI8_P23_P22　(0x83) #define　ICS_SCI9_P01_P00　(0x90) #define　ICS_SCI9_PA3_PA2　(0x91) #define　ICS_SCI9_PG1_PG0　(0x92) #define　ICS_SCI12_PB5_PB6　(0xC0) #define　ICS_SCI12_P23_P22　(0xC1) #define　ICS_SCI12_P01_P00　(0xC2)
ライブラリ	ics2_RX66T.lib
ヘッダファイル	ics2_RX66T.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE SCIx　全て DTC　全て ICU　対応する部分 SYSTEM.MSTPCRB　対応する部分 MPC　対応する部分 PORT　対応する部分	数値表示・設定 8bit 符号なし／符号あり　整数型 16bit 符号なし／符号あり　整数型 32bit 符号なし／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／符号あり　整数型 16bit 符号なし／符号あり　整数型 32bit 符号なし／符号あり　整数型 32bit IEEE754 浮動小数点

4.6.2. RX66T 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);
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。ics2_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、ICS2_<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 ICS で使用する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 　ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。 SCI1, SCI5, SCI6, SCI8, SCI9, SCI12 の場合 　PCLKB= 40MHz @CPU CLK=160MHz　（PCLKB max 60MHz） $通信レート = \frac{PCLKB}{8 \times (speed + 1)} [Mbps]$　$speed \in [0, 1, 2, \ldots 255]$ ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。 各機種のサポート範囲に収まるように speed の値を設定します。 第５パラメータ 　通信モードの設定 　０：設定禁止（将来の予約） 　１：32bit８チャンネル同時サンプリング・２回転送モード 　　（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　２：32bit４チャンネル同時サンプリング・１回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003 では使用できません。 　３：32bit１２チャンネル同時サンプリング・３回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003, W1004 では使用できません。 通常は、１を設定します。波形更新レートを速くしたい場合には２を設定します。ただし、波形表示のチャンネル数は、４チャンネルになります。

転送関数の呼び出し　　void　　ics2_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。 ICS+　W2001, W2002、T2001C, T2006A の場合 $最小通信間隔 = 1/(通信速度[bps]) \times 180 + 30[us]$ Previous products ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合 $最小通信間隔 = 1/(通信速度[bps]) \times 180 + 70[us]$ 通信速度が 1Mbps の場合、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。 ※注意：DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。
使用割り込み関数
割り込み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。 ○CS+上で動作する CC-RL 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。 例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出しを追記してください。 void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); } ○SmartConfigurator を使用してコードを生成する場合 SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、TXI は、空の関数で結構です。 ベクトル番号は、SCI のポートによって異なるため、注意してください。 #pragma interrupt (Excep_SCI1_RXI1(vect=60)) #pragma interrupt (Excep_SCI1_TXI1(vect=61)) void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); } void Excep_SCI1_TXI1(void){ /* no code */ }

4.6.3. RX66T 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

A） DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x0F000
uint32_t dtc_table[256];
```

B） 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

```
#pragma section DTCTBL
uint32_t dtc_table[256];        // caution alignment 0x000
#pragma section
```

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

２）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI6_PB0_PB1, 6, 0, 1）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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。通常は、１にしてください。

---------- List 1　main.c ----------

```
#pragma address dtc_table=0x0F000
uint32_t dtc_table[256];

void main(void)
{
      ics2_init(dtc_table, ICS_SCI6_PB0_PB1, 6, 0, 1); // CN3
      while(1)
      {   nop();   }
}
```

３）ics2_watchpoint()関数の組込み

　このソフトでは、メインルーチンで ics2_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、各 ICS ユニットで可能な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指定間隔で呼び出される場合、List2 のように ics2_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　ics2_watchpoint()の間引き ----------

```
int     deci = 0;

void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_watchpoint();
        }
}
```

４）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。

SCI1 の場合

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

SCI5 の場合

```
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
```

※注意

SmartConfigurator を使用している場合、TXI の割込み関数の説明部に記載の通り、空の関数と#prgma 宣言も追加してください。

4.7. RX71M シリーズ(CC コンパイラ)

4.7.1. RX71M 使用資源

CPU 名	RX71M シリーズ
開発環境	CS+ Ver.6.00.00　CC-RX 2.07.00
Library version	Ver.3.60
通信レート	ライブラリの設定可能通信レート 0.5Mbps～7.5Mbps $通信レート = \frac{PCLKB}{8\times(speed+1)}[Mbps]$　speed ∈[0, 1, 2, ..255] 標準通信レート　PCLKB = 60MHz,　speed=0　の場合 7.5Mbps ご使用になる ICS/ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	SCI0　TXD0:P32,　RXD0:P33 SCI0　TXD0:P20,　RXD0:P21 SCI1　TXD1:P26,　RXD1:P30 SCI1　TXD1:P16,　RXD1:P15 SCI2　TXD2:P13,　RXD2:P12 SCI2　TXD2:P50,　RXD2:P52 SCI3　TXD3:P23,　RXD3:P25 SCI3　TXD3:P16,　RXD3:P17 SCI4　TXD4:PB1,　RXD4:PB0 SCI5　TXD5:PC3,　RXD5:PC2 SCI5　TXD5:PA4,　RXD5:PA3 SCI6　TXD6:P00,　RXD6:P01 SCI6　TXD6:P32,　RXD6:P33 SCI6　TXD6:PB1,　RXD6:PB0 SCI7　TXD7:P90,　RXD7:P92
ライブラリ	ics2_RX71M.lib
ヘッダファイル	ics2_RX71M.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE SCIx　全て DTC　全て ICU:対応する部分 SYSTEM.MSTPCRB. 対応する部分 MPC　対応する部分 PORTx　対応する部分	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

4.7.2. RX71M 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);
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。ics2_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、ics2_<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 ICS で使用する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 　ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。 $$通信レート = \frac{PCLKB}{8 \times (speed + 1)} [Mbps]$$ ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。 各機種のサポート範囲に収まるように speed の値を設定します。 第５パラメータ 　通信モードの設定 　０：設定禁止（将来の予約） 　１：32bit８チャンネル同時サンプリング・２回転送モード 　　（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　２：32bit４チャンネル同時サンプリング・１回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003 では使用できません。 　３：32bit１２チャンネル同時サンプリング・３回転送モード 　　（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） 　　※注意：W1001, W1003, W1004 では使用できません。 　４：設定禁止（将来の予約） 通常は、１を設定します。波形更新レートを速くしたい場合には２を設定します。ただし、波形表示のチャンネル数は、４チャンネルになります。

転送関数の呼び出し　　void　　ics2_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、下記の条件に適合するように呼び出すようにしてください。

ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合

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

通信速度が 1Mbps の場合、上の式に数値を代入すると。

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

ICS+　W2001, W2002、T2001C, T2006A の場合

$$最小通信間隔 = 1/(通信速度[bps]) \times 180 + 30[us]$$

※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。
※注意 : DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。

使用割り込み関数

割り込み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。

○CS+上で動作する CC-RL 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。
例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出しを追記してください。

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

○SmartConfigurator を使用してコードを生成する場合
SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、TXI は、空の関数で結構です。
ベクトル番号は、SCI のポートによって異なるため、注意してください。

```
#pragma interrupt (Excep_SCI2_RXI2(vect=62))
#pragma interrupt (Excep_SCI2_TXI2(vect=63))
void Excep_SCI2_RXI2(void){ ics_int_sci_rxi(); }
void Excep_SCI2_TXI2(void){ /* no code */ }
```

4.7.3. RX71M 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

C) DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x0F000
uint32_t dtc_table[256];
```

D) 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

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

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

３）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI0_P32_P33, 6, 0, 1) 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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。通常は、１にしてください。

---------- List 1　main.c ----------

```
#pragma address dtc_table=0x0F000
uint32_t dtc_table[256];

void main(void)
{
      ics2_init((void*)dtc_table, ICS_SCI0_P32_P33, 6, 0, 1);     /* Interrupt level 6            */
      while(1)
      {   nop();   }
}
```

４）ics2_watchpoint()関数の組込み

　このソフトでは、メインルーチンで ics2_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、通信レート、ICS のモデルに応じた時間以上, 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上記規定時間以下の間隔で呼び出される場合、List2 のように ics2_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　ics2_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_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(); }
```

SCI1 の場合

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

SCI2 の場合

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

SCI3 の場合

```
   void Excep_SCI3_RXI3(void){ ics_int_sci_rxi(); }
```

SCI4 の場合

```
   void Excep_SCI4_RXI4(void){ ics_int_sci_rxi(); }
```

SCI5 の場合

```
    void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
```
SCI6 の場合
```
    void Excep_SCI6_RXI6(void){ ics_int_sci_rxi(); }
```
SCI7 の場合
```
    void Excep_SCI7_RXI7(void){ ics_int_sci_rxi(); }
```

※注意
SmartConfigurator を使用している場合、TXI の割込み関数の説明部に記載の通り、空の関数と#prgma 宣言も追加してください。

4.8. RX72T シリーズ (CC コンパイラ)

4.8.1. RX72T 使用資源

CPU 名	RX72T シリーズ
開発環境	CS+ Ver.8.01.00　CC-RX 3.01.00
Library version	Ver.3.60 (RX V3 core)
通信レート	ライブラリの設定可能通信レート 0.5Mbps～7.5Mbps PCLK は、 SCI1, SCI5, SCI6, SCI8, SCI9, SCI12 の時 PCLKB (PCLKB max 60MHz) SCI11　の時 PCLKA（PCLKA max 120MHz） を使用します。 $通信レート = \frac{PCLK}{8 \times (speed + 1)} [Mbps]$　$speed \in [0, 1, 2, \ldots 249]$ 例外設定： 1 [Mpbs]　@ speed = 250 かつ PCLK = 50MHz 1 [Mbps]　@ speed = 251 かつ PCLK = 60MHz 標準通信レート　PCLK = 50MHz,　speed=0　の場合 6.25Mbps ※ご使用になる ICS / ICS++の通信レート範囲に合致するように、設定してください。
サポートポート	#define　ICS_SCI1_PD3_PD5　(0x10) #define　ICS_SCI5_PD7_PE0　(0x50) #define　ICS_SCI5_PB5_PB6　(0x51) #define　ICS_SCI6_PB0_PB1　(0x60) #define　ICS_SCI8_PD0_PD1　(0x80) #define　ICS_SCI8_PC1_PC0　(0x81) #define　ICS_SCI8_PA4_PA5　(0x82) #define　ICS_SCI8_P23_P22　(0x83) #define　ICS_SCI9_P01_P00　(0x90) #define　ICS_SCI9_PA3_PA2　(0x91) #define　ICS_SCI9_PG1_PG0　(0x92) #define　ICS_SCI11_PD3_PD5　(0xB1) #define　ICS_SCI11_PB5_PB6　(0xB2) #define　ICS_SCI12_PB5_PB6　(0xC0) #define　ICS_SCI12_P23_P22　(0xC1) #define　ICS_SCI12_P01_P00　(0xC2)
ライブラリ	ics2_RX72T.lib
ヘッダファイル	ics2_RX72T.h

CPU 使用リソース	サポート変数タイプ
・内部使用リソース INT　SCIx RXI INT　SCIx TXI DTC　TXIx ICU.DTCER[xx].BIT.DTCE	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

SCIx　全て DTC　全て ICU　対応する部分 SYSTEM.MSTPCRB　対応する部分 MPC　　対応する部分 PORT　対応する部分	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

4.8.2. RX72T 関数説明

初期化関数の呼び出し　void ics2_init(void* addr, char　port,　char level,　char speed,　char mode);
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。ics2_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、ICS2_<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 ICS で使用する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 ICS++システムで利用する通信レートを定義します。通信レートの計算方法は以下の通りです。 SCI1, SCI5, SCI6, SCI8, SCI9, SCI12 の場合 PCLK として PCLKB を使用します。 SCI11 の場合、PCLK として PCLKA を使用します。 $通信レート = \frac{PCLK}{8 \times (speed + 1)} [Mbps] \quad speed \in [0, 1, 2, \ldots 249]$ ただし、例外として： 1 [Mpbs]　@ speed = 250 かつ PCLK = 50MHz 1 [Mbps]　@ speed = 251 かつ PCLK = 60MHz ICS ユニットの種類や Firmware のバージョンにより、使用可能な通信レートの範囲が異なります。 各機種のサポート範囲に収まるように speed の値を設定します。 第５パラメータ 通信モードの設定 ０：設定禁止（将来の予約） １：32bit８チャンネル同時サンプリング・２回転送モード （２回の ics2_watchpoint()関数呼び出しで１回のサンプリング） ２：32bit４チャンネル同時サンプリング・１回転送モード （１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） ※注意：W1001, W1003 では使用できません。 ３：32bit１２チャンネル同時サンプリング・３回転送モード （１回の ics2_watchpoint()関数呼び出しで１回のサンプリング） ※注意：W1001, W1003, W1004 では使用できません。 通常は、１を設定します。波形更新レートを速くしたい場合には２を設定します。ただし、波形表示のチャンネル数は、４チャンネルになります。

転送関数の呼び出し　　void　　ics2_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。

ICS　W1001,　ICS W1003, T2001B, T2002B,　ICS++ W1004 の場合

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

通信速度が 1Mbps の場合、上の式に数値を代入すると。

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

ICS+　W2001, W2002、T2001C, T2006A の場合

$$最小通信間隔=1/(通信速度[bps])\times 180+30[us]$$

※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。
※注意 : DTC や SCI をユーザー側のソフトウェアで多用する場合、バスアクセスが多くなり DTC の転送が間に合わなくなり上記にタイミングが間に合わなくなるケースがあります。

使用割り込み関数

割り込み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。

○CS+上で動作する CC-RL 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。
例として、SCI1 を使用する場合、チャンネルに応じて、下記のように RXI 割込みに、ics_int_sci_rxi()関数の呼び出しを追記してください。

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

○SmartConfigurator を使用してコードを生成する場合
SmartConfigurator は、SmartConfigurator で指定した割込み関数しか定義しないため、ICS で使用する SCI の割込みを全て、手作業で記述することが必要です。ただし、TXI は、空の関数で結構です。
ベクトル番号は、SCI のポートによって異なるため、注意してください。

```
#pragma interrupt (Excep_SCI1_RXI1(vect=60))
#pragma interrupt (Excep_SCI1_TXI1(vect=61))
void Excep_SCI1_RXI1(void){ ics_int_sci_rxi(); }
void Excep_SCI1_TXI1(void){ /* no code */ }
```

4.8.3. RX72T 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）DTC テーブルを適切なメモリーに配置する。
実現するために、A) B)のいずれかの方法を使用してください。サンプルでは、A)を使用しています。

E） DTC テーブルを、#pragma address 直接指令を使用して絶対アドレスに配置する。

```
#pragma address dtc_table=0x0F000
uint32_t dtc_table[256];
```

F） 開発環境のセクション指定で、dtc_table のセクションアドレスを指定する。
下記のように、#pragma 指令で dtc_table のセクションを指定する。

```
#pragma section DTCTBL
uint32_t dtc_table[256];        // caution alignment 0x000
#pragma section
```

さらに、開発環境の
プロジェクトツリー
→ビルド・ツール
→右クリックでプロパティーを表示
→リンクオプション
→セクション
のセクション指定部で、BDTCTBL のアドレスを指定する。

※注意
dtc_table のアドレスの下位 12bit が０になるような RAM 上のアドレスに割り当てます。

２）ics2_init()を下記のように呼び出す。

初期化関数 ics2_init((void*)dtc_table, ICS_SCI6_PB0_PB1, 6, 0, 1）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 dtc_table 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 ICS2_<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。
第 4 パラメータは、通信速度に応じて選択してください。
第 5 パラメータは、転送モードに応じて選択してください。通常は、１にしてください。

---------- List 1　main.c ----------

```
#pragma address dtc_table=0x0F000
uint32_t dtc_table[256];

void main(void)
{
      ics2_init(dtc_table, ICS_SCI6_PB0_PB1, 6, 0, 1);
      while(1)
      {   nop();   }
}
```

３）ics2_watchpoint()関数の組込み

　このソフトでは、メインルーチンで ics2_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、各 ICS ユニットで可能な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指定間隔で呼び出される場合、List2 のように ics2_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　ics2_watchpoint()の間引き ----------

```
int     deci = 0;


void    int_TM0(void)     /* 100us間隔 */
{
        deci = deci + 1;
        if (deci >=3)
        {
            deci = 0;
            ics2_watchpoint();
        }
}
```

４）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。

SCI1 の場合

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

SCI5 の場合

```
void Excep_SCI5_RXI5(void){ ics_int_sci_rxi(); }
```

※注意

SmartConfigurator を使用している場合、TXI の割込み関数の説明部に記載の通り、空の関数と#prgma 宣言も追加してください。

5. 改訂履歴

バージョン	変更日	変更内容
Ver.1.00	2017-08-30	・初版作成、RX23T, RX24T, RX62T RX63T, RX71M を掲載
Ver.1.01	2017-11-16	・表現を修正
Ver.1.02	2018-11-28	・各 ICS のサポートする転送モードを表として追記 ・RX66T を追加
Ver.1.03	2019-07-09	・RX72T を追加
Ver.1.04	2020-04-21	・RX13T を追加
Ver.1.05	2020-06-25	・割込み関数記述に関する注意事項を追加

ICS++ Library　Function manual

発行年月日　2020 年 6 月 25 日　　Ver.1.05JP

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