Desk Top Lab

ICS++ library Ver.3.7x
Function reference manual
(15 channel waveform viewer support version)

For RENESAS RL78 series
CA78K0R (CS+ V4.01.00)
CC-RL (CS+ V6.01.00)
EWRL Ver3.10

Index

1. はじめに 4
1.1. はじめに 4
1.2. 注意事項 4
2. ICS ハードウェアによる機能の違い 5
2.1. ICS / ICS++ の種類 5
2.1.1. ICS++ W2002 （2018/3 時点、販売機種） 5
2.1.2. T2001C / T2006A 搭載 サブセット ICS++（2018/3 時点、販売機種） 6
2.1.3. ICS++ W1004（販売終了品） 7
2.1.4. ICS W1001（販売終了品） 7
2.1.5. ICS W1003（販売終了品） 8
2.1.6. T2001A, T2001B, T2002A, T2002B 搭載 サブセット ICS（販売終了品） 8
2.1.7. ICS++ W2001 （販売終了品） 8
2.2. 各シリーズの機能の差 9
2.3. 転送レートの設定方法（重要） 9
2.3.1. ICS / ICS++ ハードウェアによる制約 10
2.3.2. ターゲット CPU／クロックによる制約 10
2.4. 実際のシステムにおける通信レート設定例 10
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 10
2.5.1. W2002, W2001, W1004, T2001C, T2006A の場合 10
2.5.2. 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） 11
2.5.3. 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） 11
3. ICS++ライブラリの基本仕様・動作 12
3.1. 通信規約・ソースコード 12
3.2. データ転送間隔の制限 12
3.3. 転送モード 0, 1, 2, 3, 4, 5, 6 の違い 13
3.4. 数値表示ウィンドウ使用時の制約 14
3.5. ファイル構成・ライブラリ 15
4. 開発環境への設定 16
4.1. 開発環境の設定（CA78K0R CS+ V4.01.00） 16
4.1.1. 変数情報生成のための、map file への変数情報の追加 16
4.1.2. 変数情報の生成 17
4.1.3. メモリーモデルの設定 18
4.2. 開発環境の設定（CC-RL CS+ V6.01.00） 19
4.2.1. 変数情報生成のための、map file への変数情報の追加 19
4.2.1. 変数情報の生成１（新ツール） 20
4.2.2. 変数情報の生成１（旧ツール） 22
4.2.3. メモリーモデルの設定 23
4.3. 開発環境の設定 （EWRL V3.10） 24
4.3.1. 一般オプション → ターゲット 24
4.3.2. C/C++コンパイラ → 言語１ 24
4.3.3. C/C++コンパイラ → 言語２ 25
4.3.4. C/C++コンパイラ → 最適化 25
4.3.5. C/C++コンパイラ → プリプロセッサー 26
4.3.6. ビルドアクション → （ポストビルドコマンドライン） 26
5. 使用資源・ライブラリの説明 28

5.1. RL78/G1F シリーズ 28
5.1.1. RL78G1F シリーズ使用資源 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通） 28
5.1.2. RL78G1F シリーズ関数説明 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通） 30
5.1.3. RL78G1F CS+ CA78K0R コンパイラ使用方法 32
5.1.4. RL78G1F CS+ CCRL コンパイラ使用方法 35
5.1.5. RL78G1F EWRL V3.10 コンパイラ 関数使用方法 38
5.2. RL78/F14 シリーズ 40
5.2.1. RL78F14 シリーズ使用資源 （CS+ CA78K0R, CS+ CC-RL, ~~EWRL V3.10~~ 共通） 40
5.2.2. RL78F14 シリーズ関数説明 （CS+ CA78K0R, CS+ CC-RL, ~~EWRL V3.10~~ 共通） 42
5.2.3. RL78F14 CS+ CA78K0R コンパイラ使用方法 44
5.2.4. RL78F14 CS+ CC-RL コンパイラ使用方法 46
5.2.5. RL78F14 EWRL V3.10 コンパイラ 関数使用方法 49
5.3. RL78/G14 シリーズ 51
5.3.1. RL78G14 シリーズ使用資源 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通） 51
5.3.2. RL78G14 シリーズ関数説明 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通） 54
5.3.3. RL78G14 CS+ CA78K0R コンパイラ使用方法 56
5.3.4. RL78G14 CS+ CCRL コンパイラ使用方法 59
5.3.5. RL78G14 EWRL V3.10 コンパイラ 関数使用方法 62
6. 改訂履歴 65

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, W1003, T2001A, T2001B および ICS++ シリーズ W1004, W2001, W2002, T2001C, T2006A 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。

2. ICS ハードウェアによる機能の違い

2.1. ICS / ICS++ の種類

ICS / ICS++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、機種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 （2018/3 時点、販売機種）

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、最大 15ch 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。

2.1.2. T2001C / T2006A 搭載 サブセット ICS++（2018/3 時点、販売機種）

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。

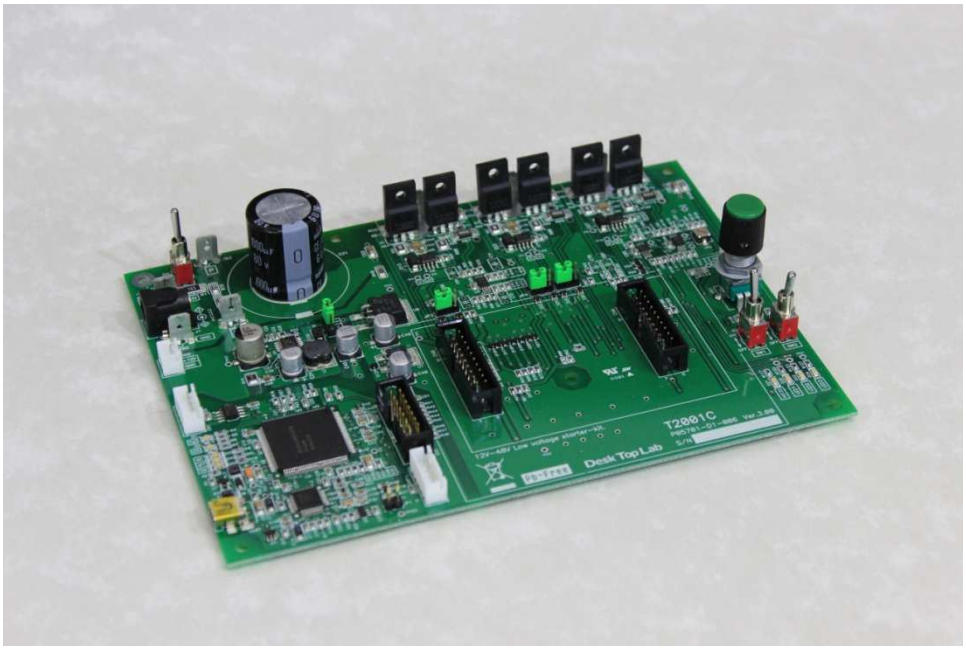


図 2 T2001C 低電圧インバータ（T2001B の後継機種）

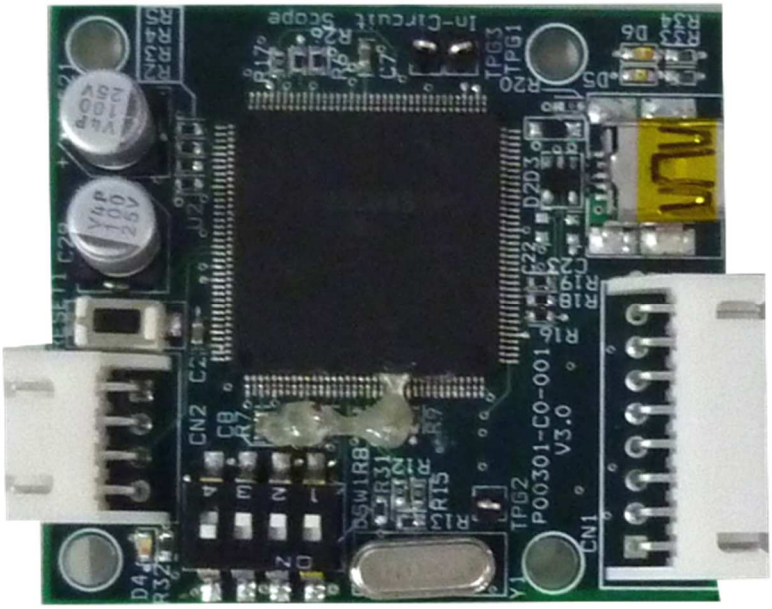
図 3 T2006A 低電圧インバータ（三相インバータ ３ポート版）

2.1.3. ICS++ W1004（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。
Firmware が Ver.1.35 まで、 通信レート 0.5Mbps～1.25Mbps
Firmware が Ver.1.52 以降、通信レート 0.5Mbps～1.25Mbps、および、 1.5Mbps, 3Mbps

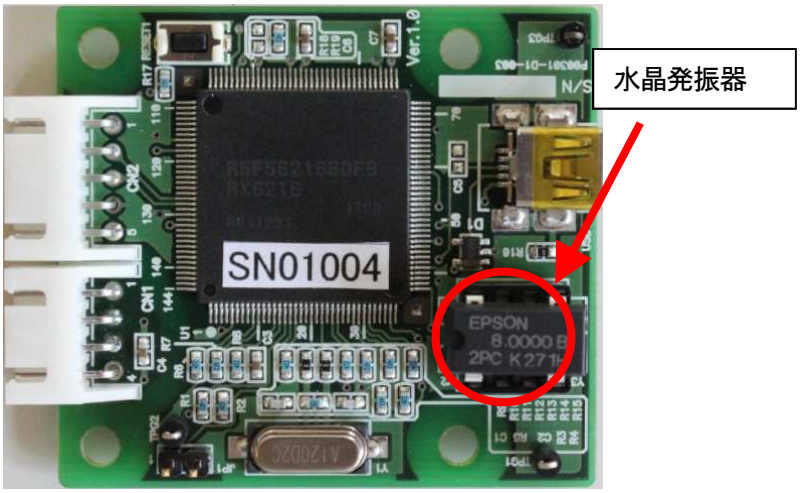
2.1.4. ICS W1001（販売終了品）

W1001
下記の写真のような、1Mbps 固定タイプの ICS です。



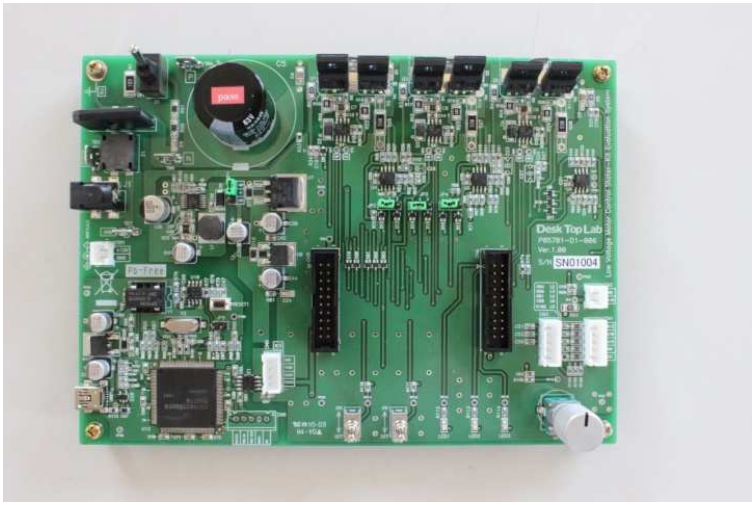
2.1.5. ICS W1003（販売終了品）

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. T2001A, T2001B, T2002A, 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 （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 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: V1.0 16bit mode 8ch 32bit mode 12ch W2002: V1.2 16bit mode 15ch 32bit mode 12ch 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

『**エラー! 参照元が見つかりません。**』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、RL78G1F の場合、以下のようになります。

$$\text{通信レート} = \frac{fclk}{2 \times (speed + 1)}[Mbps] \qquad (\text{speed} \geqq 2)$$

ここで、fclk は、実際に使用する RL78/G1F のクロック周波数。speed は、2 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）RL78/G1F　fclk = 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
2	5.333Mbps
3	4Mbps
4	3.2Mbps
5	2.67Mbps
6	2.28Mbps
7	2Mbps
15	1Mbps

W2002 の場合、

0.5Mbps～8Mbps が選択できるので、5.333Mbps 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。

W1003　の場合

0.5Mbps～1.25Mbps が選択できるので、1Mbps 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

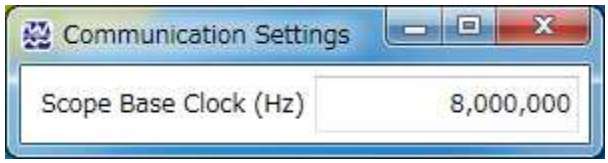
本ライブラリを使用する場合、ＣＰＵ側のクロックの設定に従って、ICS++ボード上のクロックを下記のように選択してください。

2.5.1. W2002, W2001, W1004, 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　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。

2.5.3.　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、通信レート 1Mbps=通信クロック 8MHz 以外のクロックでの使用はできません。

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　W2002, W2001, 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

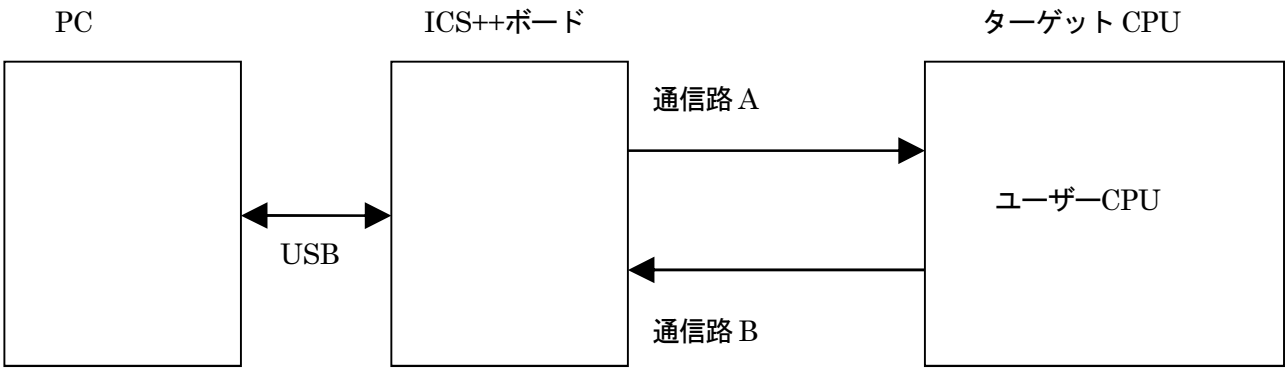


図 4　ICS++システムの通信路

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。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、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、2020 年 6 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)

５）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 ics2_watchpoint() 関数の実行時間が短い	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。

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, もしくは ics2_<CPUNAME>.o

通常関数として、

```
void    ics2_init( void*    addr,     char    unitpin,    char    level , char    speed, char mode );
void    ics2_watchpoint(void);
uint32_t    ics2_version(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、次節で説明する状態で動作を確認しています。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。

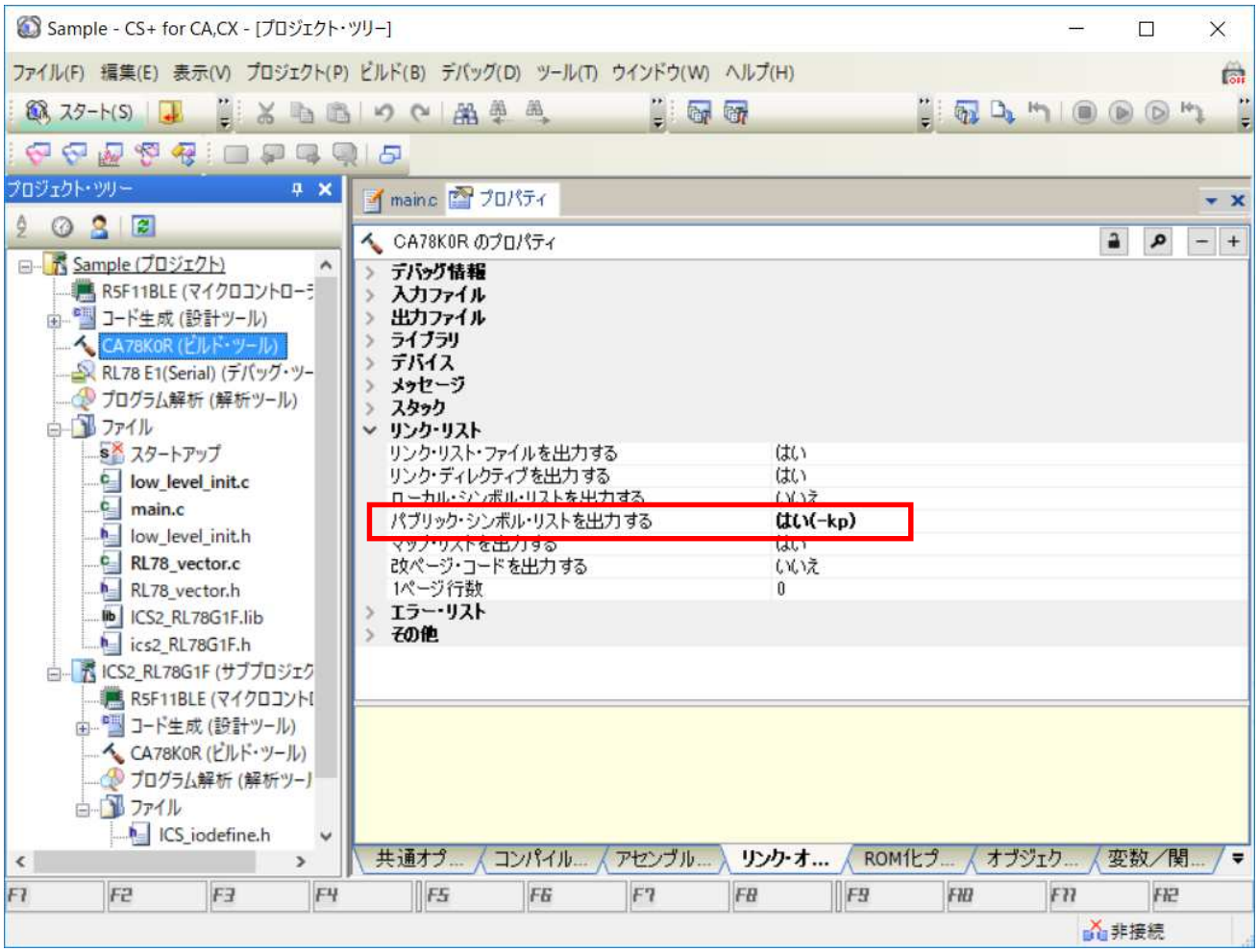
4. 開発環境への設定

4.1. 開発環境の設定 (CA78K0R　CS+ V4.01.00)

使用するプロジェクトで ICS を使用する場合、ソースコードにライブラリを組み込むだけではなく、いくつかの設定をしないと便利に使うことができません。以下では、CA78K0R コンパイラに対する組み込み方法を説明します。基本的には、デフォルトの設定で使用するようになっていますが、最低限以下の設定をする必要があります。

4.1.1. 変数情報生成のための、map file への変数情報の追加

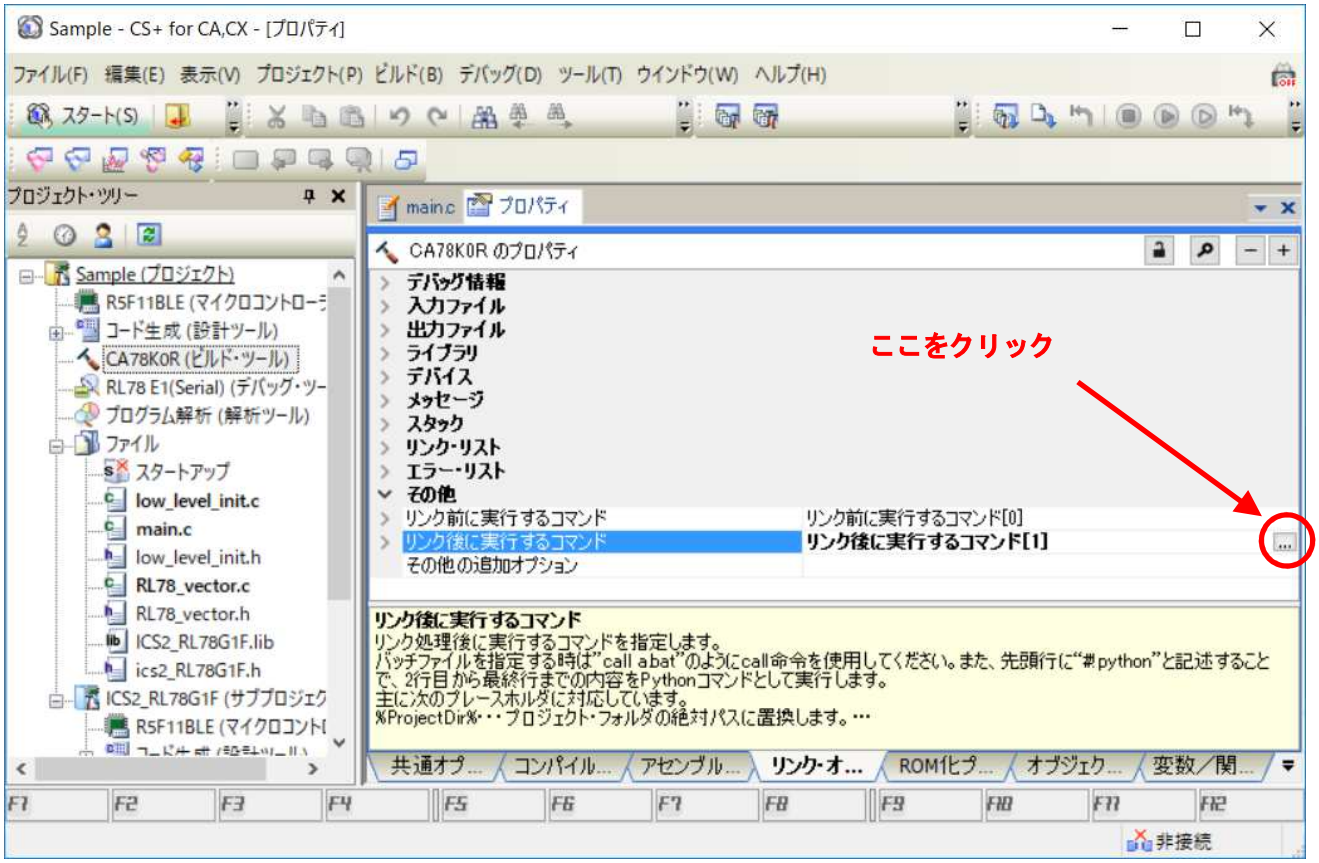
プロジェクトツリー
→リンク・オプション
　→リンクリスト
　　→パブリンク・シンボル・リストを出力する　　「はい(-kp)」に変更する。

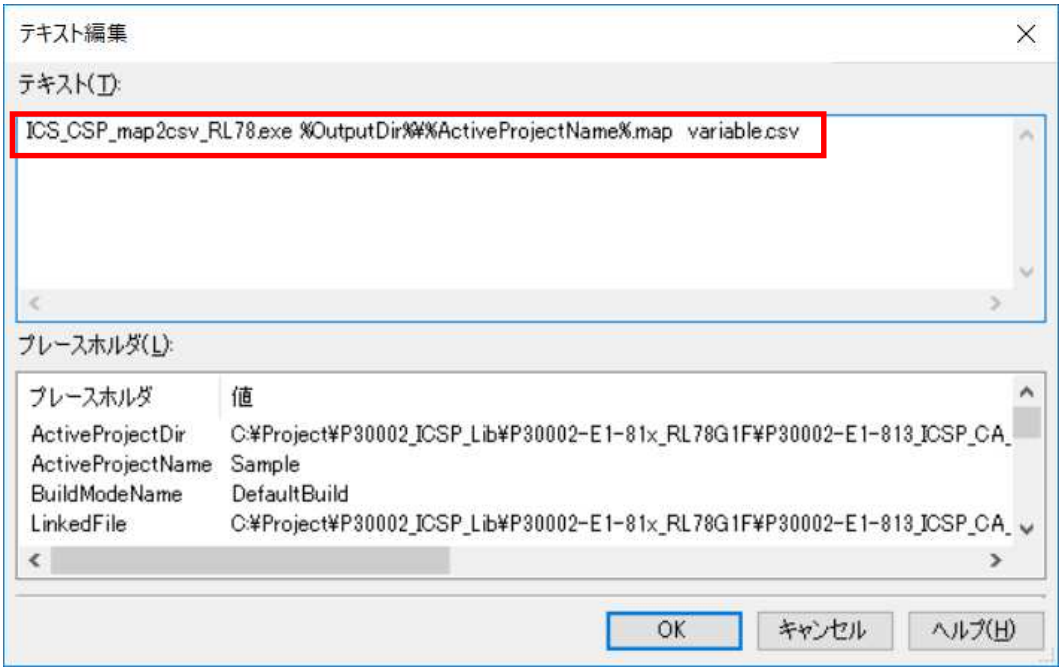


4.1.2. 変数情報の生成

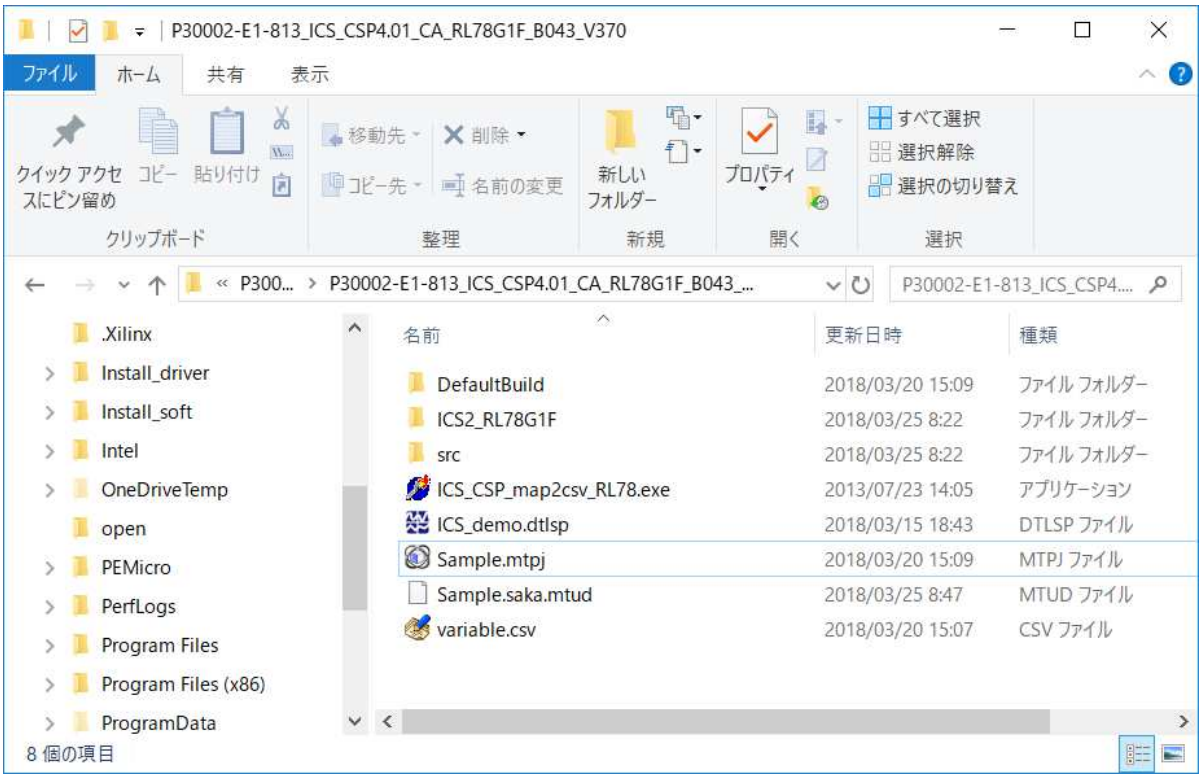
プロジェクトツリー
→リンク・オプション
→その他
→リンク後に実行するコマンド　　に下記のコマンドを指定する

ICS_CSP_map2csv_RL78.exe %OutputDir%¥%ActiveProjectName%.map　variable.csv



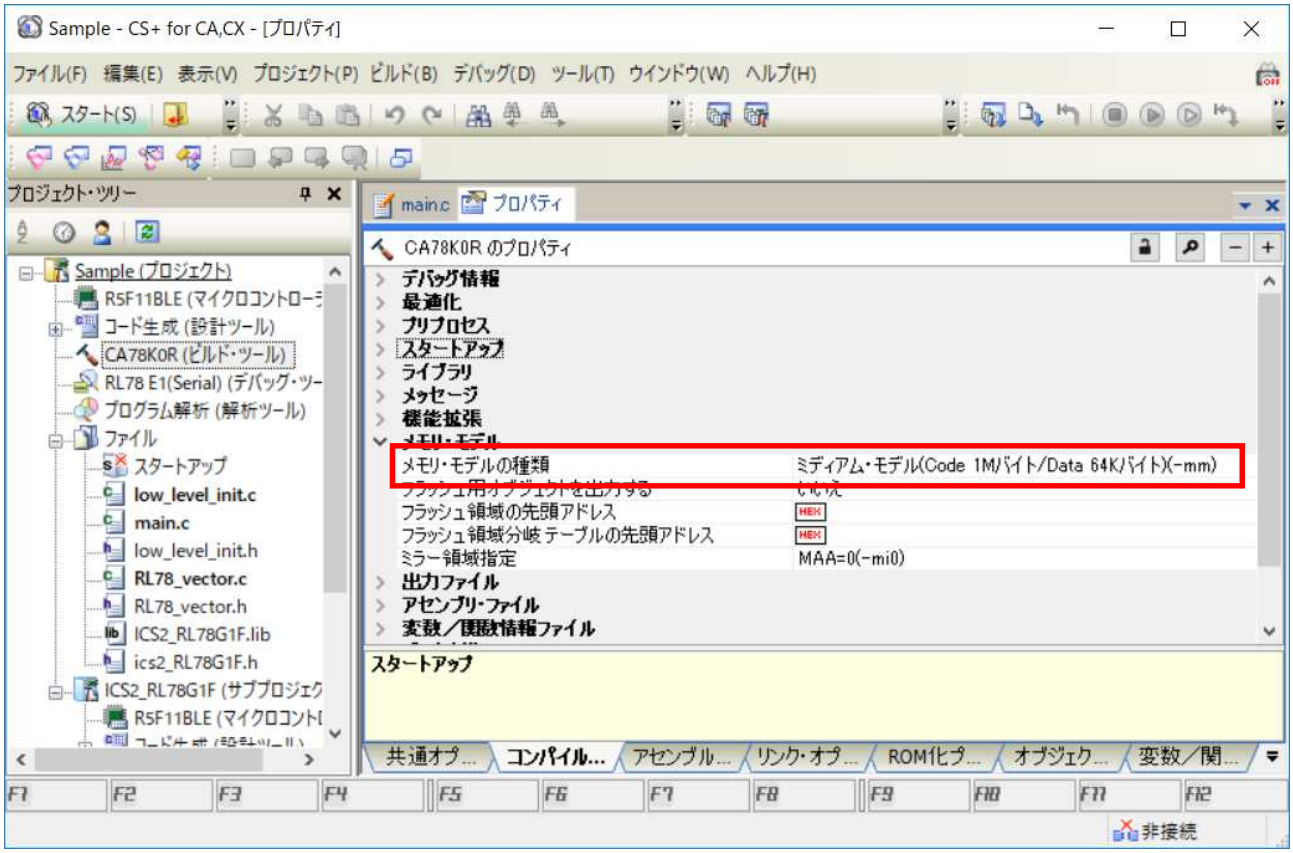


さらに、下のように、プロジェクトフォルダーに、ICS_CSP_map2csv_RL78.exe をコピーする



4.1.3. メモリーモデルの設定

コンパイラのデフォルト設定です。ミディアム・モデルで使用してください。

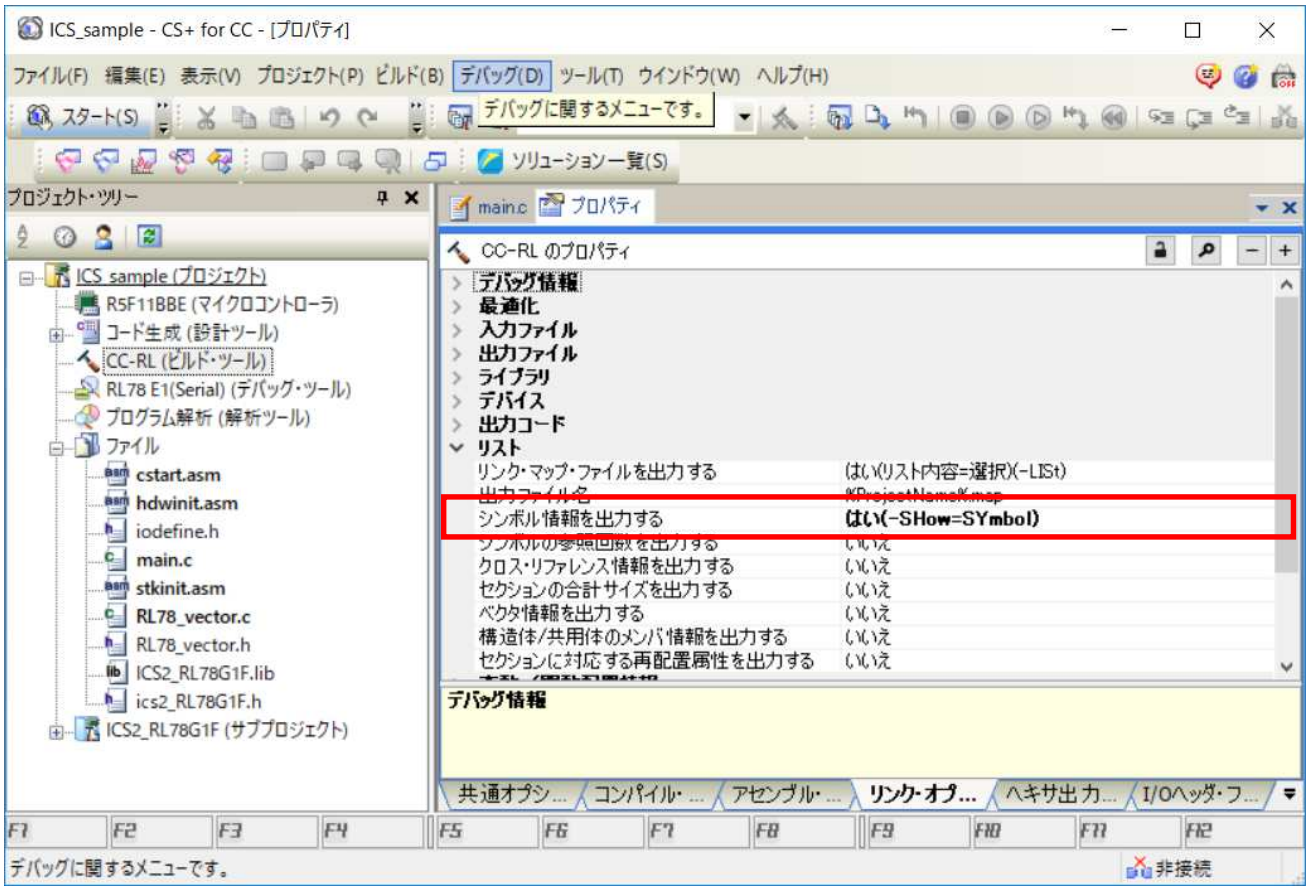


4.2. 開発環境の設定 (CC-RL　CS+ V6.01.00)

使用するプロジェクトで ICS を使用する場合、ソースコードにライブラリを組み込むだけではなく、いくつかの設定をしないと便利に使うことができません。以下では、CC-RL コンパイラに対する組み込み方法を説明します。

4.2.1. 変数情報生成のための、map file への変数情報の追加

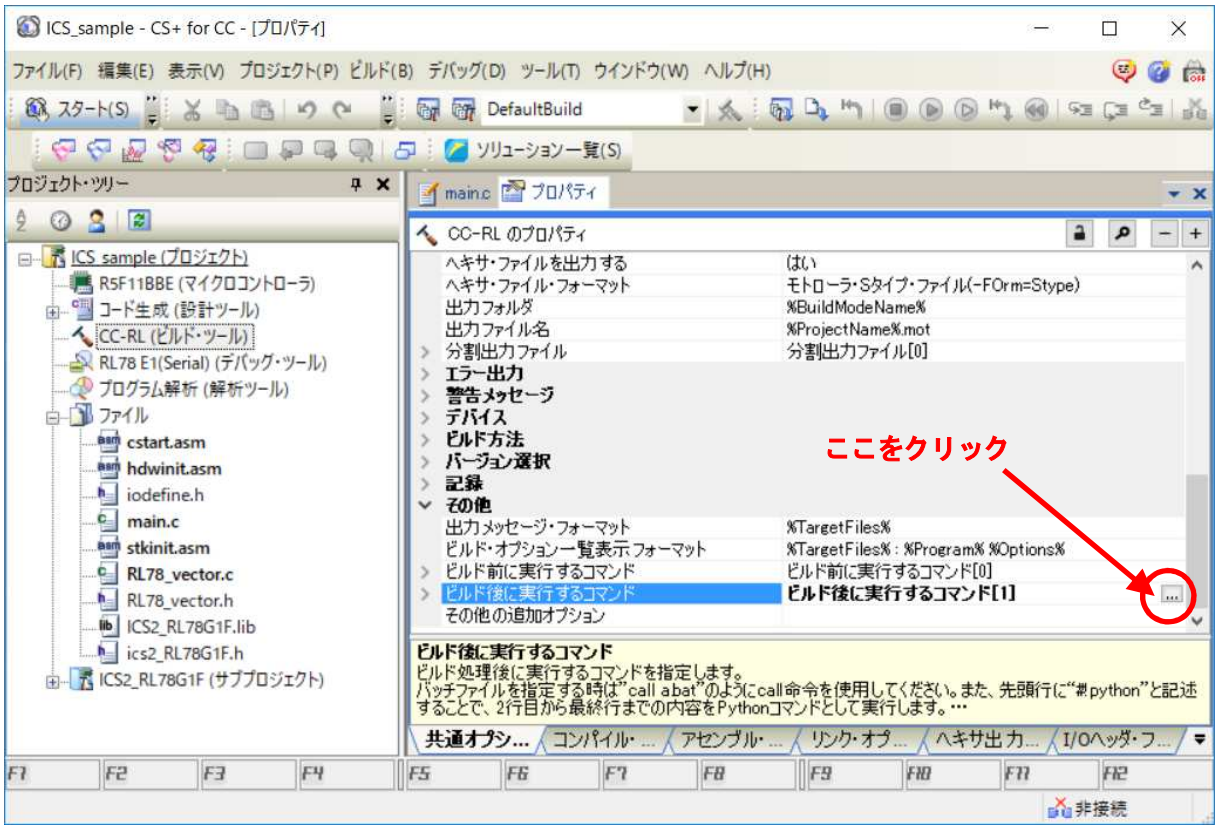
プロジェクトツリー
→リンク・オプション
　→リスト
　　→シンボル情報を出力する　　「はい(-SHow=SYmbol)」に変更する。

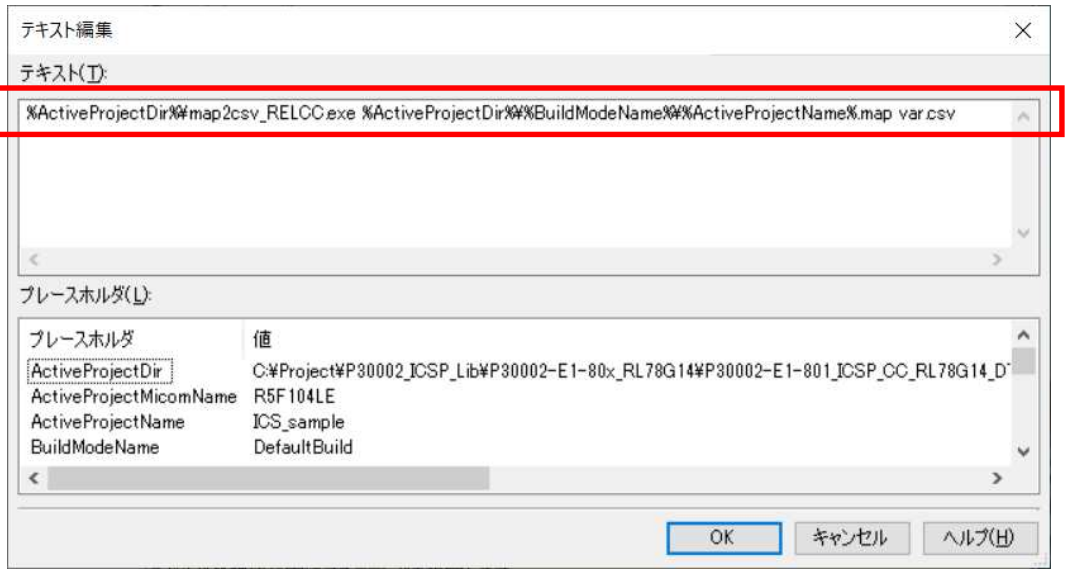


4.2.1. 変数情報の生成１（新ツール）

プロジェクトツリー
→共通オプション
→その他
→ビルド後に実行するコマンド　に下記のコマンドを指定する
下記は、３行に分けているが、スペース区切りで１行に書くこと

%ActiveProjectDir%¥map2csv_RELCC.exe
%ActiveProjectDir%¥%BuildModeName%¥%ActiveProjectName%.map
var.csv





map2csv_RELCC.exe は、パラメータをファイルで指定することが可能です。

１） remove.def
変数リストから削除する変数名を指定可能です。これにより、波形表示ツール上に表示される不要な変数名を削除することが可能です。
２） map2csv.def
6 行のデータです。
RL78 の場合デフォルトは、下記のデータが入っています。

1:0
2:3
4:5
x:0
s:F0000
e:FFEDF

１行目は、変数が１バイトの時の変数タイプの指定、0: UINT8,　1:INT8
２行目は、変数が２バイトの時の変数タイプの指定、2: UINT16,　3:INT16
３行目は、変数が４バイトの時の変数タイプの指定、4:UINT32,　5:INT32
４行目は、変数が上記以外の場合の変数タイプの指定 0-6
５行目は、変数範囲開始アドレス
６行目は、変数範囲終了アドレス

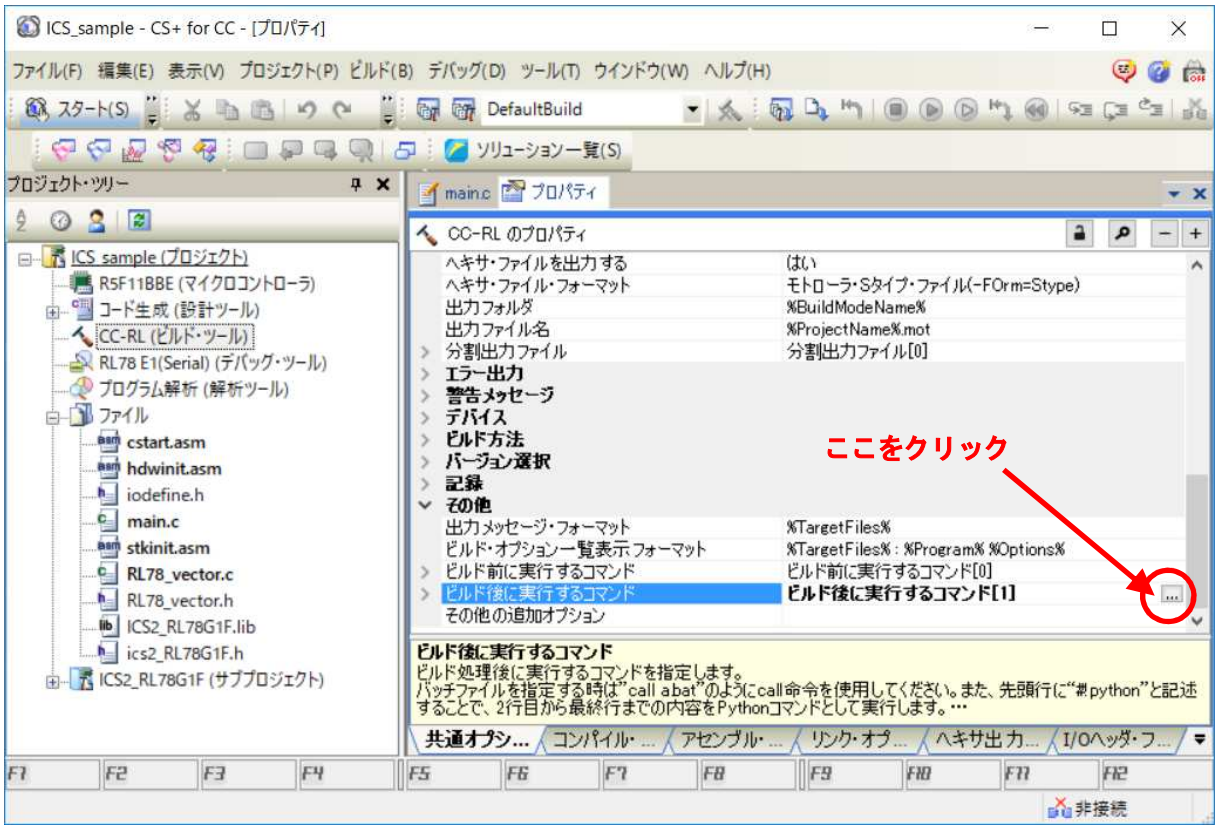
※注意１
最初の２文字は読み飛ばされるので、注意してください。
１から４行目までは最初から３番目の文字だけで判断されます。
※注意２
変数は、５行目、６行目で指定されるアドレス範囲に適合する変数のみは、変数情報として出力されます。

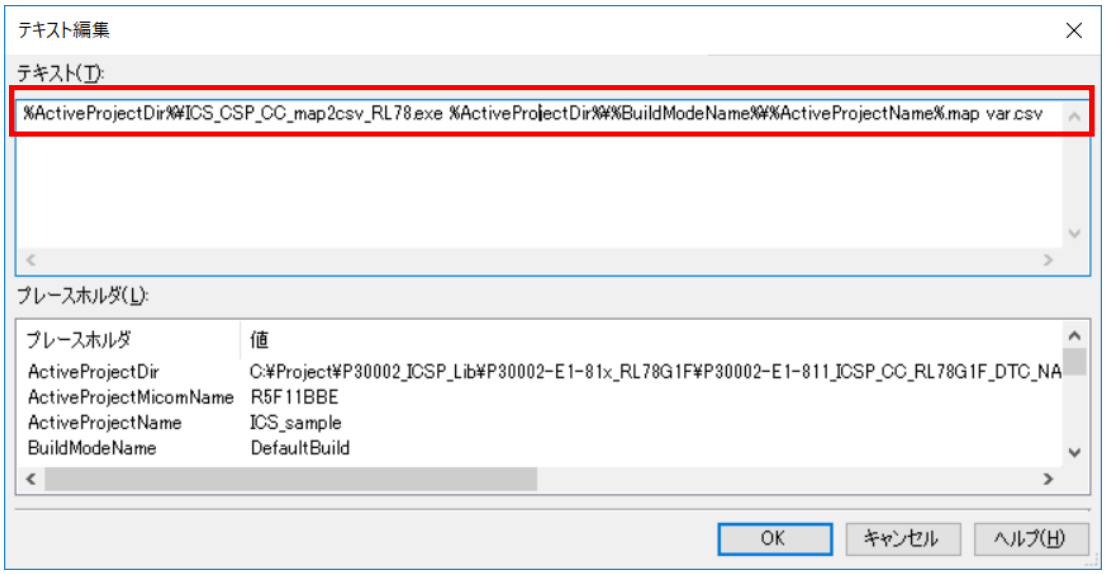
4.2.2. 変数情報の生成１（旧ツール）

※注意　本ツールは、互換性のために残しています。変数情報の生成１（新ツール）4.2.1 **で紹介している新ツールを推奨します。**

プロジェクトツリー
→共通オプション
→その他
→ビルド後に実行するコマンド　　に下記のコマンドを指定する
下記は、３行に分けているが、スペース区切りで１行に書くこと

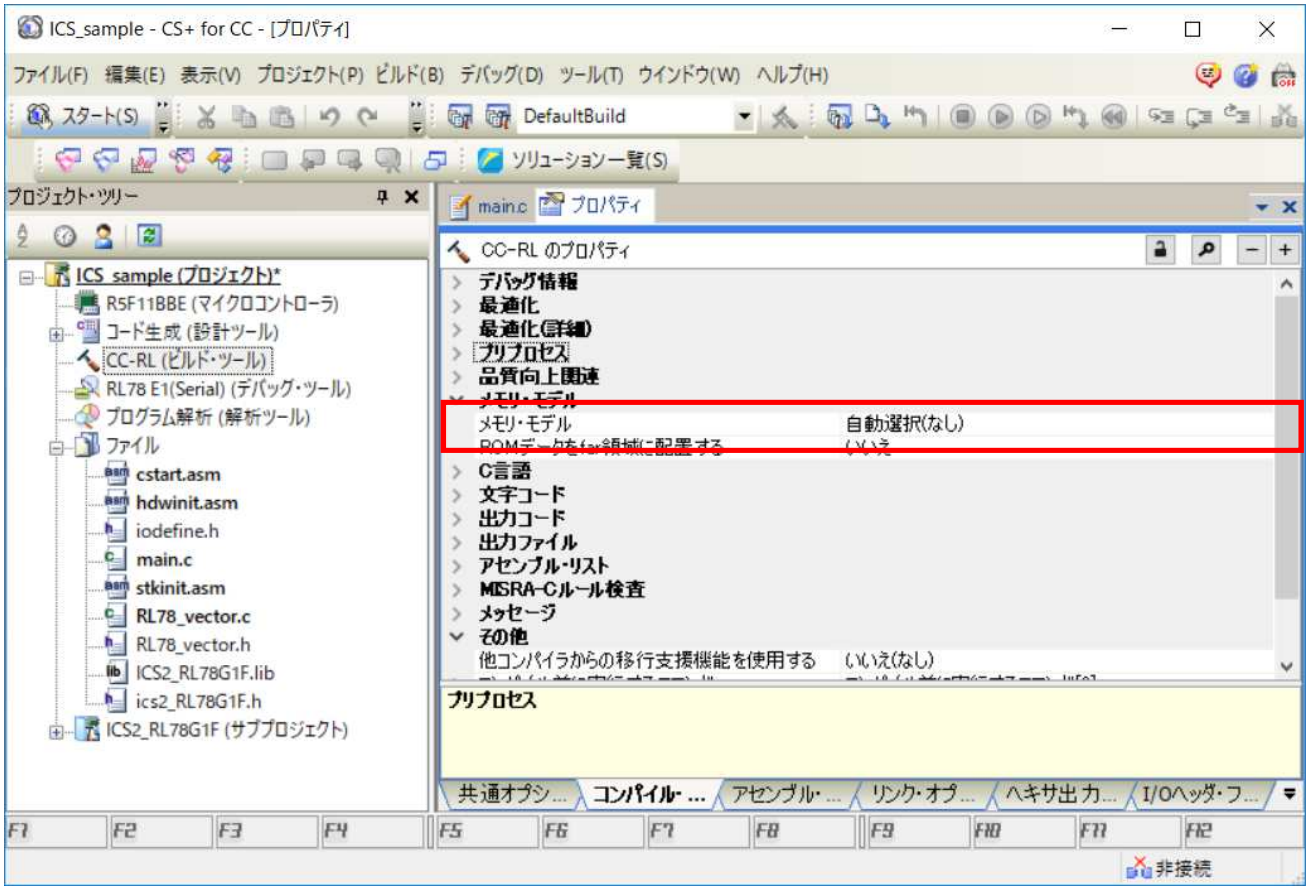
%ActiveProjectDir%¥ICS_CSP_CC_map2csv_RL78.exe
%ActiveProjectDir%¥%BuildModeName%¥%ActiveProjectName%.map
var.csv





4.2.3. メモリーモデルの設定

コンパイラのデフォルト設定です。自動選択で使用してください。

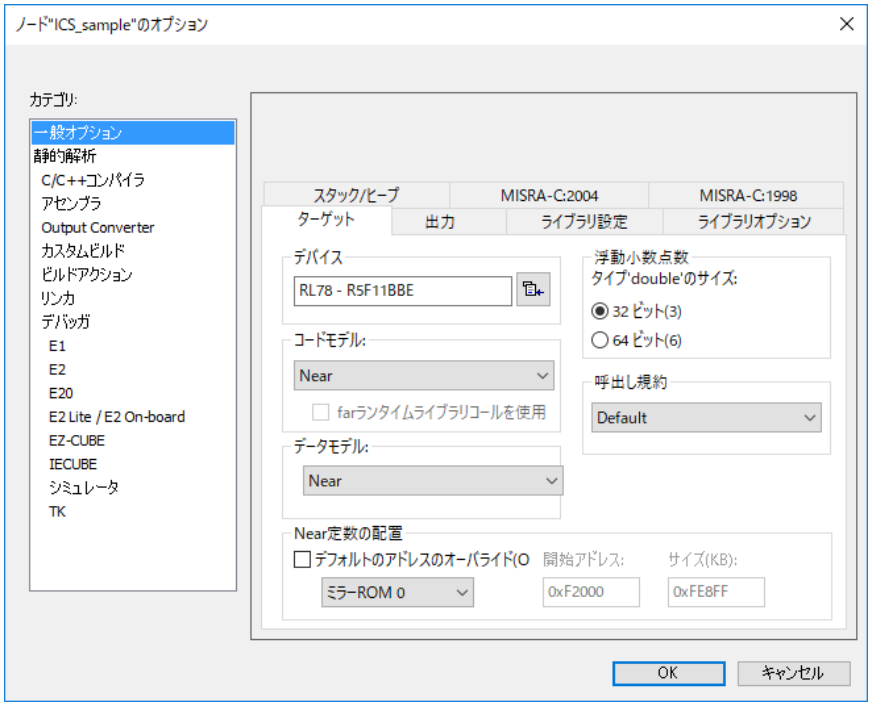


4.3. 開発環境の設定 (EWRL V3.10)

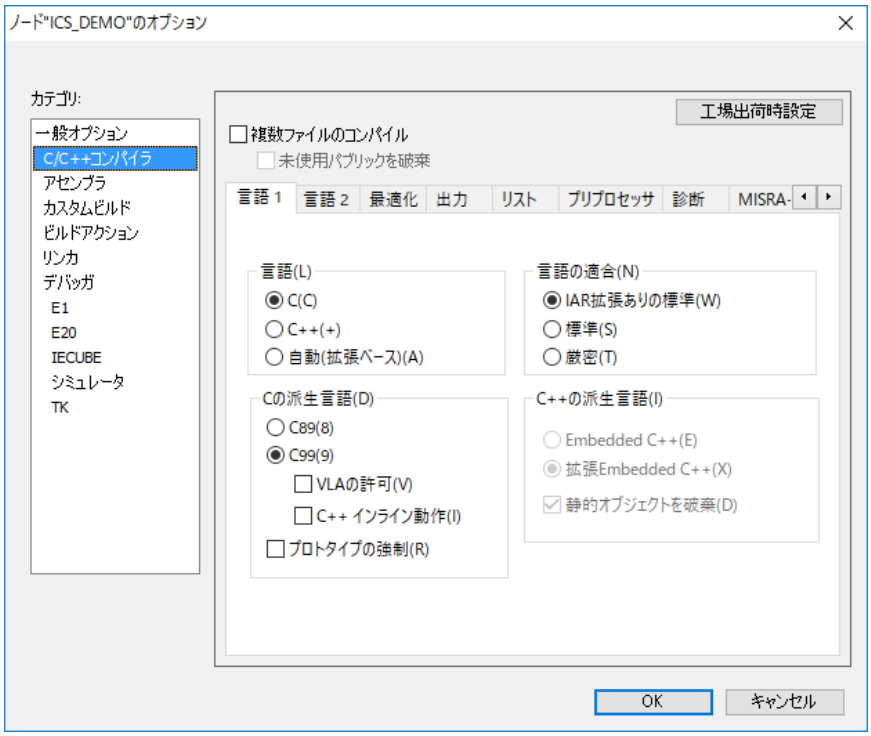
使用するプロジェクトで ICS を使用する場合、ソースコードにライブラリを組み込むだけではなく、いくつかの設定をしないと便利に使うことができません。以下では、EWRL V3.10 コンパイラに対する組み込み方法を説明します。

4.3.1. 一般オプション → ターゲット

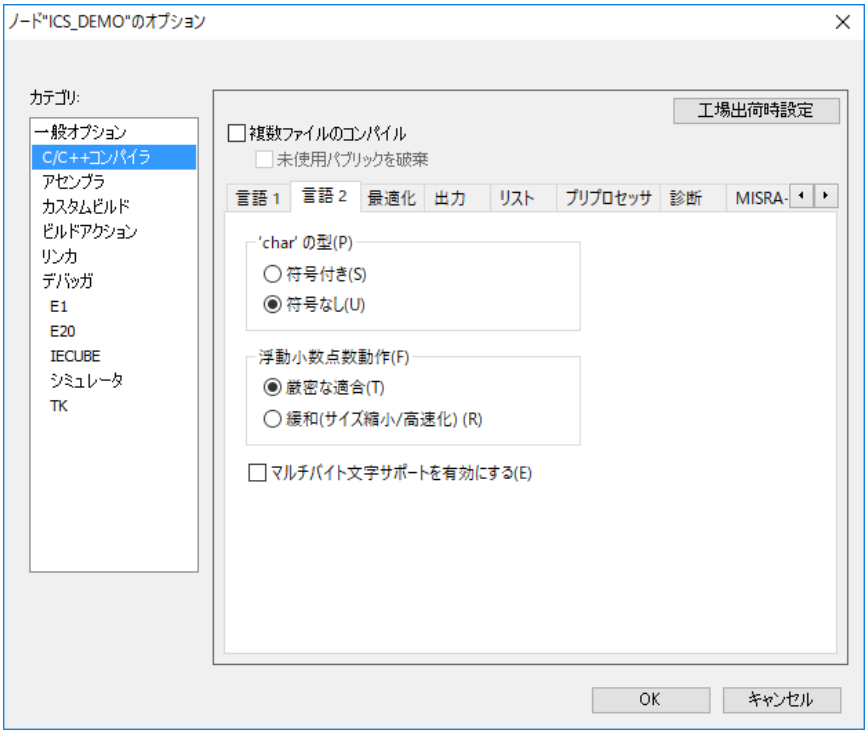
選択する CPU は、お客様のご使用になる CPU のモデルに合わせてください。



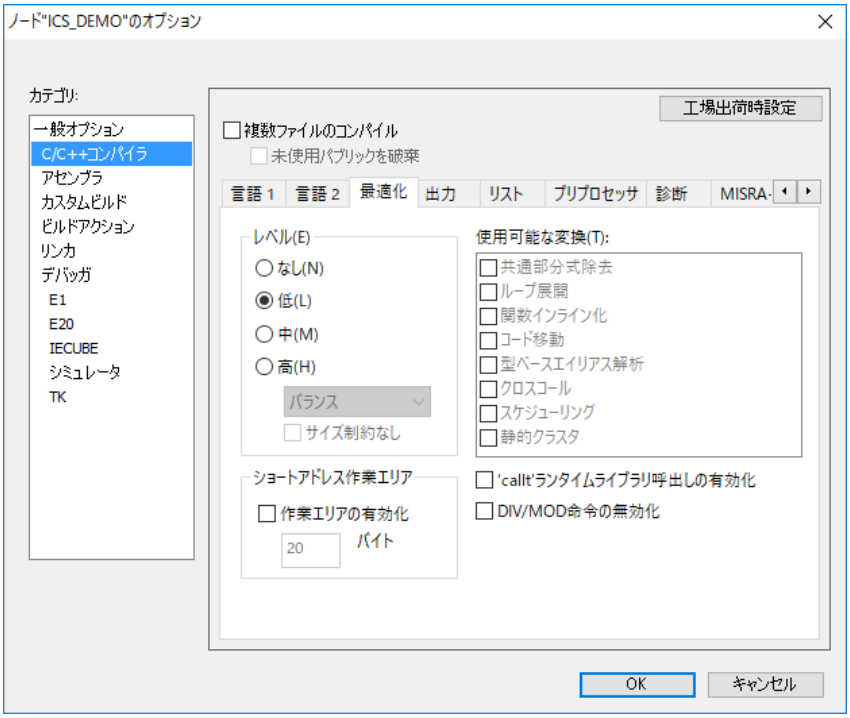
4.3.2. C/C++コンパイラ → 言語 1



4.3.3. C/C++コンパイラ → 言語２



4.3.4. C/C++コンパイラ → 最適化

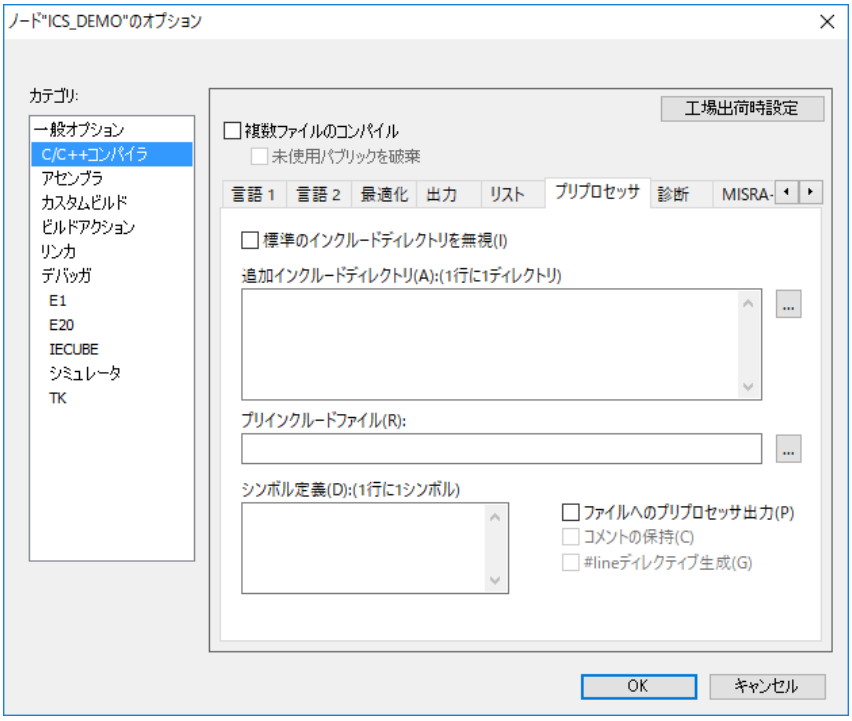


本ライブラリは、「最適化：低」で動作確認を行っております。最適化レベルを上げる場合、アプリケーショ

ンによっては、マニュアル通りの動作にならない場合がありますので、ご了承ください。

4.3.5. C/C++コンパイラ → プリプロセッサー

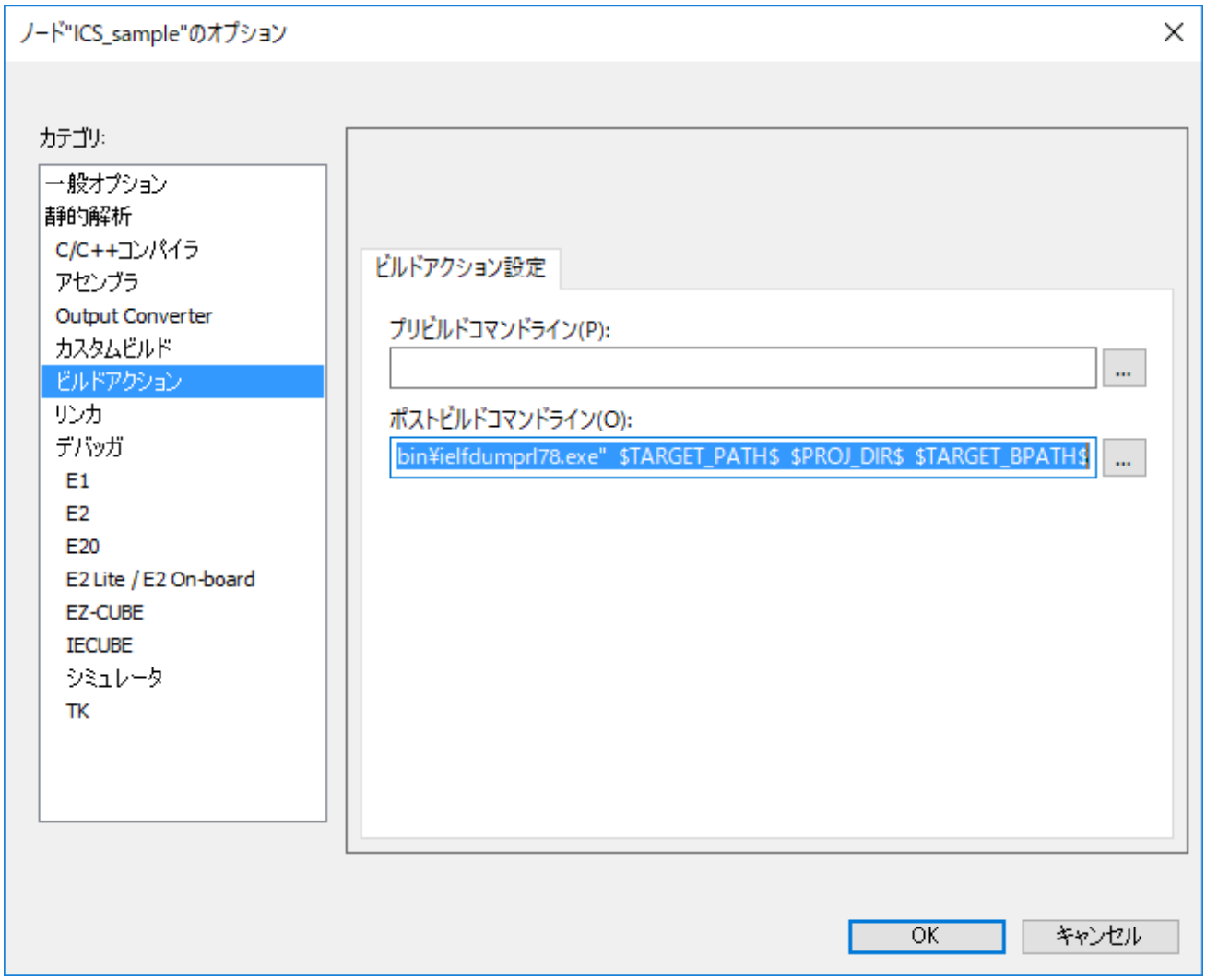
追加インクルードディレクトリは、お客様の環境に合わせてください。本指定は、ICS サンプルプログラムの例です。



4.3.6. ビルドアクション → （ポストビルドコマンドライン）

この行は、ICS を使用する際に変数情報を必要としますが、その変数情報を生成するためのツールを起動するための設定です。下記のの 5 行に書かれている文字列を 一行にしてポストビルドコマンドラインの行に入れてください。

```
$PROJ_DIR$¥tools¥temp.bat
  "$TOOLKIT_DIR$¥bin¥ielfdumprl78.exe"
    $TARGET_PATH$
      $PROJ_DIR$
        $TARGET_BPATH$
```


ノード"ICS_sample"のオプション
カテゴリ:
一般オプション
静的解析
C/C++コンパイラ
アセンブラ
Output Converter
カスタムビルド
ビルドアクション
リンカ
デバッガ
E1
E2
E20
E2 Lite / E2 On-board
EZ-CUBE
IECUBE
シミュレータ
TK
ビルドアクション設定
プリビルドコマンドライン(P):
ポストビルドコマンドライン(O):
bin¥ielfdumprl78.exe" $TARGET_PATH$ $PROJ_DIR$ $TARGET_BPATH$
OK
キャンセル

5. 使用資源・ライブラリの説明

5.1. RL78/G1F シリーズ

5.1.1. RL78G1F シリーズ使用資源 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通）

CPU 名	RL78G1F シリーズ	
開発環境	CS+ Ver.4.01.00 (CA78K0R) CS+ Ver.6.01.00 (CC-RL) EWRL Ver.3.10	
Library version	Ver3.70	
通信レート	0.5Mbps – 5.33Mbps 通信レート $= \frac{f_{CLK}}{2\times(speed+1)}[Mbps]$ (speed≧2) 標準通信レート $f_{CLK} = 32MHz$, speed = 15 の場合 1Mbps	
ステイタス	SCI0, SCI1, SCI2 サポート	
ライブラリ	CS+ CA78K0R, CS+ CC-RL “ics2_RL78G1F.Lib” EWRL78 V3.10 “ics2_RL78G1F.o” ライブラリは、全 RL78G1F デバイス共通です。	
ヘッダファイル	ics2_RL78G1F.h” ヘッダファイルは、全 RL78G1F デバイス共通です。	
メモリーモデル	CS+ CA78K0R, CS+ CC-RL Medium (ROM=1MB, RAM=64kB) EWRL V3.10 Code near, Data near	
サポートポート	R5F11B7Cx R5F11B7Ex (24pin)	#define ICS_R5F11B7_SCI0_P51_P50 #define ICS_R5F11B7_SCI1_P00_P01 #define ICS_R5F11B7_SCI1_P72_P73 #define ICS_R5F11B7_SCI2_P13_P14
	R5F11BBCx R5F11BBEx (32pin)	#define ICS_R5F11BB_SCI0_P51_P50 #define ICS_R5F11BB_SCI0_P17_P16 #define ICS_R5F11BB_SCI1_P00_P01 #define ICS_R5F11BB_SCI1_P72_P73 #define ICS_R5F11BB_SCI2_P13_P14
	~~R5F11BCCx~~ ~~R5F11BCEx~~ ~~(36pin)~~	~~#define ICS_R5F11BC_SCI0_P51_P50~~ ~~#define ICS_R5F11BC_SCI0_P17_P16~~ ~~#define ICS_R5F11BC_SCI1_P00_P01~~ ~~#define ICS_R5F11BC_SCI2_P13_P14~~

	R5F11BGCx R5F11BGEx (48pin)	#define ICS_R5F11BG _SCI0_P51_P50 #define ICS_R5F11BG _SCI0_P17_P16 #define ICS_R5F11BG _SCI1_P00_P01 #define ICS_R5F11BG _SCI1_P72_P73 #define ICS_R5F11BG _SCI2_P13_P14
	R5F11BLCx R5F11BLEx (64pin)	#define ICS_R5F11BL_SCI0_P51_P50 #define ICS_R5F11BL _SCI0_P17_P16 #define ICS_R5F11BL _SCI1_P02_P03 #define ICS_R5F11BL _SCI1_P77_P76 #define ICS_R5F11BL _SCI2_P13_P14
CPU 使用リソース		サポート変数タイプ
・内部リソース DTC SCIx 全資源 PFC 外部ピン Pxx for TXDx (使用したポート) Pxx for RXDx (使用したポート) CLOCK SCI0, SCI1 の場合：SPS0 bit4～7 SCI2 の場合：SPS1 bit4～7 INTC STPR0x STPR1x SRPR0x SRPR1x SREPR0x SREPR1x		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

5.1.2. RL78G1F シリーズ関数説明 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通）

Lib Ver.3.70

初期化関数の呼び出し

void ics2_init(uint16_t addr, uint8_t pin, uint8_t level, uint8_t num, uint8_t speed, uint8_t 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。 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。

R5F11BLE の場合、下記から選択します。

#define ICS_R5F11BL_SCI0_P51_P50
#define ICS_R5F11BL _SCI0_P17_P16
#define ICS_R5F11BL _SCI1_P02_P03
#define ICS_R5F11BL _SCI1_P77_P76
#define ICS_R5F11BL _SCI2_P13_P14

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。

第５パラメータ

通信速度の定義

$$\text{通信レート} = \frac{f_{CLK}}{2\times(speed+1)}[Mbps] \quad (\text{speed}\geqq 2)$$

第６パラメータ

通信モードの設定

０：8/16bit８チャンネルモード（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）

４：8/16bit１５チャンネルモード（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング）

６：16bit８チャンネルモード（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）

その他：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+　W2001, W2002 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。 ICS　W1001,　ICS W1003,　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。 $最小通信間隔 = 1/(1[Mbps]) \times 180 + 70[us] = 250[us]$
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 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。
関数実行時間（ics2_watchpoint）
CC-RL　　CS+ Ver.6.01.00　@32MHz Mode 0　　4.05us Mode 4　　2.7us～6.0us Mode 6　　2.9us CA78K0R　CS+ Ver.4.01.00　@32MHz Mode 0　　4.2us Mode 4　　2.7us～6.4us Mode 6　　2.8us CC-RL　　EWRL V3.10　　@32MHz Mode 0　　4.4us Mode 4　　2.7us～6.5us Mode 6　　2.9us

5.1.3. RL78G1F CS+ CA78K0R コンパイラ使用方法

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）ics2_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 ics2_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。ics2_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。ics2_RL78G1F.h で定義されている文字列をお使いください。
R5F11BLE の場合、下記のピンが使用可能です。

```
#define    ICS_R5F11BL_SCI0_P51_P50
#define    ICS_R5F11BL _SCI0_P17_P16
#define    ICS_R5F11BL _SCI1_P02_P03
#define    ICS_R5F11BL _SCI1_P77_P76
#define    ICS_R5F11BL _SCI2_P13_P14
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78G1F ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

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

#include "low_level_init.h"
#include "ics2_RL78G1F.h"
/*********** KEEP DTC TABLE AREA ***********/
#pragma section @@DATA   @@DTCTBL at 0xFFE00
char   dtc_tbl[0xD0];
#pragma section @@DATA   @@DATA

    ics2_init(0xFE00, ICS_R5F11BL_SCI0_P51_P50, 2, 0x40, 2, 0);      // W2002 の場合  5.333Mbps
  // ics2_init(0xFE00, ICS_R5F11BL_SCI0_P51_P50, 2, 0x40, 15, 0);     // W1004 の場合   1Mbps
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

割込関数において

```
{
    // 制御ルーチン開始

    // 制御ルーチン終了


    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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();}
```

SCI2 の場合

```
__interrupt void   Excep_INTST2(void)   {int_ics_sci_tx();}
__interrupt void   Excep_INTSR2(void)   {int_ics_sci_rx();}
```

5.1.4. RL78G1F CS+ CCRL コンパイラ使用方法

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 address    dtc_tbl = 0xFFE00
char    dtc_tbl[0xD0];
```

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）ics2_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 ics2_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。ics2_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。ics2_RL78G1F.h で定義されている文字列をお使いください。
R5F11BLE の場合、下記のピンが使用可能です。

```
#define    ICS_R5F11BL_SCI0_P51_P50
#define    ICS_R5F11BL _SCI0_P17_P16
#define    ICS_R5F11BL _SCI1_P02_P03
#define    ICS_R5F11BL _SCI1_P77_P76
#define    ICS_R5F11BL _SCI2_P13_P14
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78G1F ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

```
#include "iodefine.h"
#include "ics2_RL78G1F.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma address    dtc_tbl = 0xFFE00
char    dtc_tbl[0xD0];

void main(void)
{
    ics2_init(0xFE00, ICS_R5F11BL_SCI0_P51_P50, 2, 0x40, brr, 0);
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

```
割込関数において

{
    // 制御ルーチン開始

    // 制御ルーチン終了


    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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

```
#pragma interrupt Excep_INTST0    (vect=INTST0)
#pragma interrupt Excep_INTSR0    (vect=INTSR0)
void    Excep_INTST0(void)    { int_ics_sci_tx();    }
void    Excep_INTSR0(void)    { int_ics_sci_rx();    }
```

SCI1 の場合

```
#pragma interrupt Excep_INTST1    (vect=INTST1)
#pragma interrupt Excep_INTSR1    (vect=INTSR1)
void    Excep_INTST1(void)    { int_ics_sci_tx();    }
void    Excep_INTSR1(void)    { int_ics_sci_rx();    }
```

SCI2の場合

```
#pragma interrupt Excep_INTST2    (vect=INTST2)
#pragma interrupt Excep_INTSR2    (vect=INTSR2)
void    Excep_INTST2(void)    { int_ics_sci_tx();    }
void    Excep_INTSR2(void)    { int_ics_sci_rx();    }
```

5.1.5. RL78G1F EWRL V3.10 コンパイラ 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例を紹介します。詳細は、サンプルコードを参照してください。

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 256 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 location=0xFFE00
__no_init volatile char dtc_tbl[256];
```

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）ics2_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 ics2_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。ics2_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, 0xE0, 0xE8, 0xF0, 0xF8 が使用可能です。
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。ics2_RL78G1F.h で定義されている文字列をお使いください。
R5F11BLE の場合、下記のピンが使用可能です。

```
#define    ICS_R5F11BL_SCI0_P51_P50
#define    ICS_R5F11BL _SCI0_P17_P16
#define    ICS_R5F11BL _SCI1_P02_P03
#define    ICS_R5F11BL _SCI1_P77_P76
#define    ICS_R5F11BL _SCI2_P13_P14
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78G1F ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

```
#include   "ics2_RL78G1F.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma location=0xFFE00
__no_init volatile char dtc_tbl[256];

void   main(void)
{
   ics2_init(0xFE00, ICS_R5F11BL_SCI0_P51_P50, 2, 0x40, 2, 0);    //W2002 5.333Mbps @32MHz
}
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

割込関数において

```
{
      // 制御ルーチン開始

      // 制御ルーチン終了


      int_cnt = int_cnt +1;
      if (int_cnt>2)
      {
          int_cnt = 0;
          RPECTL = 0x80U;
          ics2_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();}
```

5.2. RL78/F14 シリーズ

5.2.1. RL78F14 シリーズ使用資源 （CS+ CA78K0R, CS+ CC-RL, ~~EWRL V3.10~~ 共通）

CPU 名	RL78F14 シリーズ	
開発環境	CS+ Ver.4.01.00 (CA78K0R) CS+ Ver.6.01.00 (CC-RL) ~~EWRL Ver.3.10~~	
Library version	Ver3.70	
通信レート	0.5Mbps – 5.33Mbps 通信レート $= \frac{f_{CLK}}{2\times(speed+1)}[Mbps]$ (speed≧2) 標準通信レート $f_{CLK} = 32MHz$, speed = 15 の場合 1Mbps	
ステイタス	SCI0, SCI1 サポート	
ライブラリ	CS+ CA78K0R, CS+ CC-RL “ics2_RL78F14.Lib” ~~EWRL78 V3.10~~ ~~“ics2_RL78F14.e”~~ ライブラリは、全 RL78F14 デバイス共通です。	
ヘッダファイル	ics2_RL78F14.h” ヘッダファイルは、全 RL78F14 デバイス共通です。	
メモリーモデル	CS+ CA78K0R, CS+ CC-RL Medium (ROM=1MB, RAM=64kB) ~~EWRL V3.10~~ ~~Code near, Data near~~	
サポートポート	R5F10PAx	ICS_R5F10PAx_SCI0_P15_P16 ICS_R5F10PAx_SCI1_P12_P11
	R5F10PBx	ICS_R5F10PBx_SCI0_P62_P61 ICS_R5F10PBx_SCI0_P15_P16 ICS_R5F10PBx_SCI1_P12_P11
	R5F10PGx	ICS_R5F10PGx_SCI0_P62_P61 ICS_R5F10PGx_SCI0_P15_P16 ICS_R5F10PGx_SCI1_P12_P11
	R5F10PLx	ICS_R5F10PLx_SCI0_P62_P61 ICS_R5F10PLx_SCI0_P15_P16 ICS_R5F10PLx_SCI1_P12_P11 ICS_R5F10PLx_SCI1_P74_P75
	R5F10PMx	ICS_R5F10PMx_SCI0_P62_P61 ICS_R5F10PMx_SCI0_P15_P16 ICS_R5F10PMx_SCI1_P12_P11 ICS_R5F10PMx_SCI1_P74_P75
	R5F10PPx	ICS_R5F10PPx_SCI0_P62_P61 ICS_R5F10PPx_SCI0_P15_P16 ICS_R5F10PPx_SCI1_P12_P11 ICS_R5F10PPx_SCI1_P74_P75

CPU 使用リソース	サポート変数タイプ
・内部リソース DTC SCIx 全資源 PFC 外部ピン 　Pxx　for TXDx （使用したポート） 　Pxx　for RXDx （使用したポート） CLOCK 　SCI0, SCI1 の場合：SPS0　bit4～7 INTC 　　STPR0x 　　STPR1x 　　SRPR0x 　　SRPR1x 　　SREPR0x 　　SREPR1x	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

5.2.2. RL78F14 シリーズ関数説明 （CS+ CA78K0R, CS+ CC-RL, ~~EWRL V3.10~~ 共通）

Lib Ver.3.70

初期化関数の呼び出し

void ics2_init(uint16_t addr, uint8_t pin, uint8_t level, uint8_t num, uint8_t speed, uint8_t 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。 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。

R5F10PLJ の場合、下記から選択します。

#define ICS_R5F10PLJ_SCI1_P12_P11

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。

第５パラメータ

通信速度の定義

$$\text{通信レート} = \frac{f_{CLK}}{2\times(speed+1)}[Mbps] \quad (\text{speed} \geqq 2)$$

第６パラメータ

通信モードの設定

０：8/16bit８チャンネルモード（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）

４：8/16bit１５チャンネルモード（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング）

６：16bit８チャンネルモード（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）

その他：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+　W2001, W2002 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。 ICS　W1001,　ICS W1003,　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。 $最小通信間隔 = 1/(1[Mbps]) \times 180 + 70[us] = 250[us]$
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。
関数実行時間（ics2_watchpoint）
~~CC-RL　　CS+ Ver.6.01.00~~ ~~Mode 0　　4.05us~~ ~~Mode 4　　2.7us～6.0us~~ ~~Mode 6　　2.9us~~ ~~CA78K0R　CS+ Ver.4.01.00~~ ~~Mode 0　　4.2us~~ ~~Mode 4　　2.7us～6.4us~~ ~~Mode 6　　2.8us~~ ~~CC-RL　　EWRL V3.10~~ ~~Mode 0　　4.4us~~ ~~Mode 4　　2.7us～6.5us~~ ~~Mode 6　　2.9us~~

5.2.3. RL78F14 CS+ CA78K0R コンパイラ使用方法

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、0xFFD00 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 0xFFD00
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）ics2_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 ics2_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。ics2_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFD00 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 0xFD00 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。ics2_RL78F14.h で定義されている文字列をお使いください。
R5F10PPx の場合、下記のピンが使用可能です。

```
#define    ICS_R5F10PPx_SCI0_P62_P61
#define    ICS_R5F10PPx _SCI0_P15_P16
#define    ICS_R5F10PPx _SCI1_P74_P75
#define    ICS_R5F10PPx_SCI1_P12_P11
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78F14 ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

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

#include "low_level_init.h"
#include "ics2_RL78F14.h"
/*********** KEEP DTC TABLE AREA ***********/
#pragma section @@DATA　@@DTCTBL at 0xFFD00
char　dtc_tbl[0xD0];
#pragma section @@DATA　@@DATA

    ics2_init(0xFD00, ICS_R5F10PLJ_SCI1_P12_P11, 2, 0x40, 2, 0);    // W2002 の場合　5.333Mbps
  // ics2_init(0xFD00, ICS_R5F10PLJ_SCI1_P12_P11, 2, 0x40, 15, 0);    // W1004 の場合　1Mbps
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

割込関数において

```
{
    // 制御ルーチン開始

    // 制御ルーチン終了

    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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();}
```

5.2.4. RL78F14 CS+ CC-RL コンパイラ使用方法

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、0xFFD00 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 address    dtc_tbl = 0xFFD00
char    dtc_tbl[0xD0];
```

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）ics2_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 ics2_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。ics2_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFD00 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 0xFD00 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。ics2_RL78F14.h で定義されている文字列をお使いください。
R5F10PPx の場合、下記のピンが使用可能です。

```
#define    ICS_R5F10PPx_SCI0_P62_P61
#define    ICS_R5F10PPx _SCI0_P15_P16
#define    ICS_R5F10PPx _SCI1_P74_P75
#define    ICS_R5F10PPx_SCI1_P12_P11
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78G1F ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

```
#include "iodefine.h"
#include "ics2_RL78F14.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma address    dtc_tbl = 0xFFD00
char    dtc_tbl[0xD0];

void main(void)
{
    ics2_init(0xFE00, ICS_R5F10PPx_SCI0_P15_P16, 2, 0x40, 2, 0);    // W2002 の場合　5.333Mbps
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

割込関数において

```
{
    // 制御ルーチン開始

    // 制御ルーチン終了


    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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

```
#pragma interrupt Excep_INTST0    (vect=INTST0)
#pragma interrupt Excep_INTSR0    (vect=INTSR0)
void    Excep_INTST0(void)    { int_ics_sci_tx();    }
void    Excep_INTSR0(void)    { int_ics_sci_rx();    }
```

SCI1 の場合

```
#pragma interrupt Excep_INTST1    (vect=INTST1)
#pragma interrupt Excep_INTSR1    (vect=INTSR1)
void    Excep_INTST1(void)    { int_ics_sci_tx();    }
void    Excep_INTSR1(void)    { int_ics_sci_rx();    }
```

5.2.5. RL78F14 EWRL V3.10 コンパイラ 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例を紹介します。詳細は、サンプルコードを参照してください。

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、0xFFD00 から 256 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 location=0xFFD00
__no_init volatile char dtc_tbl[256];
```

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）ics2_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 ics2_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。ics2_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFD00 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, 0xE0, 0xE8, 0xF0, 0xF8 が使用可能です。
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。ics2_RL78F14.h で定義されている文字列をお使いください。
R5F10PPx の場合、下記のピンが使用可能です。

```
#define    ICS_R5F10PPx_SCI0_P62_P61
#define    ICS_R5F10PPx _SCI0_P15_P16
#define    ICS_R5F10PPx _SCI1_P74_P75
#define    ICS_R5F10PPx_SCI1_P12_P11
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78F14 ライブラリでは、0, 4, 6 が使用可能です。

---------- List 1　main.c ----------

```
#include  "ics2_RL78F14.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma location=0xFFD00
__no_init volatile char dtc_tbl[256];

void  main(void)
{
    ics2_init(0xFE00, ICS_R5F10PLJ_SCI1_P12_P11, 2, 0x40, 2, 0 );    // W2002 の場合
}
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

```
割込関数において

{
    // 制御ルーチン開始

    // 制御ルーチン終了


    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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();}
```

5.3. RL78/G14 シリーズ

5.3.1. RL78G14 シリーズ使用資源 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通）

CPU 名	RL78G14 シリーズ	
開発環境	CS+ Ver.4.01.00 (CA78K0R) CS+ Ver.6.01.00 (CC-RL) EWRL Ver.3.10	
Library version	Ver3.70	
通信レート	0.5Mbps – 5.33Mbps 通信レート $= \frac{f_{CLK}}{2\times(speed+1)}[Mbps]$ (speed≧2) 標準通信レート $f_{CLK} = 32MHz$, speed = 15 の場合 1Mbps	
ステイタス	SCI0, SCI1, SCI2, SCI3 サポート	
ライブラリ	CS+ CA78K0R, CS+ CC-RL “ics2_RL78G14.Lib” EWRL78 V3.10 “ics2_RL78G14.o” ライブラリは、全 RL78G1F デバイス共通です。	
ヘッダファイル	ics2_RL78G14.h” ヘッダファイルは、全 RL78G14 デバイス共通です。	
メモリーモデル	CS+ CA78K0R, CS+ CC-RL Medium (ROM=1MB, RAM=64kB) EWRL V3.10 Code near, Data near	
サポートポート	R5F104Ax (30pin)	#define ICS_R5F104Ax_SCI0_P51_P50 #define ICS_R5F104Ax_SCI1_P00_P01 #define ICS_R5F104Ax_SCI2_P13_P14
	R5F104Bx (32pin)	#define ICS_R5F104Bx_SCI0_P51_P50 #deifne ICS_R5F104Bx_SCI0_P17_P16 #define ICS_R5F104Bx_SCI1_P00_P01 #define ICS_R5F104Bx_SCI2_P13_P14
	R5F104Cx (36pin) (Not supported)	~~#define ICS_R5F104Cx_SCI0_P51_P50~~ ~~#deifne ICS_R5F104Cx_SCI0_P17_P16~~ ~~#define ICS_R5F104Cx_SCI1_P00_P01~~ ~~#define ICS_R5F104Cx_SCI2_P13_P14~~
	R5F104Ex (40pin)	#define ICS_R5F104Ex_SCI0_P51_P50 #deifne ICS_R5F104Ex_SCI0_P17_P16 #define ICS_R5F104Ex_SCI1_P00_P01 #define ICS_R5F104Ex_SCI2_P13_P14
	R5F104Fx (44pin)	#define ICS_R5F104Fx_SCI0_P51_P50 #deifne ICS_R5F104Fx_SCI0_P17_P16 #define ICS_R5F104Fx_SCI1_P00_P01 #define ICS_R5F104Fx_SCI2_P13_P14
	R5F104Gx	#define ICS_R5F104Gx_SCI0_P51_P50

	(48pin)	#deifne ICS_R5F104Gx_SCI0_P17_P16 #define ICS_R5F104Gx_SCI0_P12_P11 #define ICS_R5F104Gx_SCI1_P00_P01 #define ICS_R5F104Gx_SCI2_P13_P14
	R5F104Jx (52pin)	#define ICS_R5F104Jx_SCI0_P51_P50 #deifne ICS_R5F104Jx_SCI0_P17_P16 #define ICS_R5F104Jx_SCI1_P02_P03 #define ICS_R5F104Jx_SCI2_P77_P76 #define ICS_R5F104Jx_SCI2_P13_P14
	R5F104Lx (64pin)	#define ICS_R5F104Lx_SCI0_P51_P50 #deifne ICS_R5F104Lx_SCI0_P17_P16 #define ICS_R5F104Lx_SCI0_P12_P11 #define ICS_R5F104Lx_SCI1_P02_P03 #define ICS_R5F104Lx_SCI2_P77_P76 #define ICS_R5F104Lx_SCI2_P13_P14
	R5F104Mx (80pin)	#define ICS_R5F104Mx_SCI0_P51_P50 #deifne ICS_R5F104Mx_SCI0_P17_P16 #define ICS_R5F104Mx_SCI0_P12_P11 #define ICS_R5F104Mx_SCI1_P02_P03 #define ICS_R5F104Mx_SCI2_P77_P76 #define ICS_R5F104Mx_SCI2_P13_P14 #define ICS_R5F104Mx_SCI3_P144_P143
	R5F104Px (100pin)	#define ICS_R5F104Px_SCI0_P51_P50 #deifne ICS_R5F104Px_SCI0_P17_P16 #define ICS_R5F104Px_SCI0_P12_P11 #define ICS_R5F104Px_SCI1_P02_P03 #define ICS_R5F104Px_SCI1_P82_P81 #define ICS_R5F104Px_SCI2_P77_P76 #define ICS_R5F104Px_SCI2_P13_P14 #define ICS_R5F104Px_SCI3_P144_P143
CPU 使用リソース		サポート変数タイプ
・内部リソース DTC SCIx 全資源 PFC 外部ピン Pxx for TXDx (使用したポート) Pxx for RXDx (使用したポート) CLOCK SCI0, SCI1 の場合：SPS0 bit4～7 SCI2 の場合：SPS1 bit4～7 INTC STPR0x STPR1x SRPR0x SRPR1x		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

SREPR0x SREPR1x	

5.3.2. RL78G14 シリーズ関数説明 （CS+ CA78K0R, CS+ CC-RL, EWRL V3.10 共通）

Lib Ver.3.70

初期化関数の呼び出し

void ics2_init(uint16_t addr, uint8_t pin, uint8_t level, uint8_t num, uint8_t speed, uint8_t 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。 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。R5F104LE の場合、下記から選択します。

#define ICS_R5F104Lx_SCI0_P51_P50
#deifne ICS_R5F104Lx_SCI0_P17_P16
#define ICS_R5F104Lx_SCI0_P12_P11
#define ICS_R5F104Lx_SCI1_P02_P03
#define ICS_R5F104Lx_SCI2_P77_P76
#define ICS_R5F104Lx_SCI2_P13_P14

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。

第５パラメータ

通信速度の定義

$$\text{通信レート} = \frac{f_{CLK}}{2\times(speed+1)}[Mbps] \quad (\text{speed}\geqq 2)$$

第６パラメータ

通信モードの設定

０：8/16bit８チャンネルモード（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）
４：8/16bit１５チャンネルモード（２回の ics2_watchpoint()関数呼び出しで１回のサンプリング）
６：16bit８チャンネルモード（１回の ics2_watchpoint()関数呼び出しで１回のサンプリング）
その他：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+　W2001, W2002 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。 ICS　W1001,　ICS W1003,　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。 $最小通信間隔 = 1/(1[Mbps]) \times 180 + 70[us] = 250[us]$
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。
関数実行時間（ics2_watchpoint）
CC-RL　　CS+ Ver.6.01.00　@32MHz Mode 0　　4.05us Mode 4　　2.7us～6.0us Mode 6　　2.9us CA78K0R　CS+ Ver.4.01.00　@32MHz Mode 0　　4.2us Mode 4　　2.7us～6.4us Mode 6　　2.8us CC-RL　　EWRL V3.10　　@32MHz Mode 0　　4.4us Mode 4　　2.7us～6.5us Mode 6　　2.9us

5.3.3. RL78G14 CS+ CA78K0R コンパイラ使用方法

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 0xFFD00
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）ics2_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 ics2_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。ics2_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。ics2_RL78G14.h で定義されている文字列をお使いください。
R5F104LE の場合、下記から選択します。

```
#define    ICS_R5F104Lx_SCI0_P51_P50
#deifne    ICS_R5F104Lx_SCI0_P17_P16
#define    ICS_R5F104Lx_SCI0_P12_P11
#define    ICS_R5F104Lx_SCI1_P02_P03
#define    ICS_R5F104Lx_SCI2_P77_P76
#define    ICS_R5F104Lx_SCI2_P13_P14
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78G14 ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

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

#include "low_level_init.h"
#include "ics2_RL78G14.h"
/*********** KEEP DTC TABLE AREA ***********/
#pragma section @@DATA　@@DTCTBL at 0xFFD00
char　dtc_tbl[0xD0];
#pragma section @@DATA　@@DATA

    ics2_init(0xFD00, ICS_R5F104Lx_SCI0_P51_P50, 2, 0x40, 2, 0);      // W2002 の場合  5.333Mbps
  // ics2_init(0xFD00, ICS_R5F104Lx_SCI0_P51_P50, 2, 0x40, 15, 0);     // W1004 の場合　1Mbps
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

割込関数において

```
{
    // 制御ルーチン開始

    // 制御ルーチン終了


    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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();}
```

5.3.4. RL78G14 CS+ CCRL コンパイラ使用方法

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、0xFFD00 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 address    dtc_tbl = 0xFFD00
char    dtc_tbl[0xD0];
```

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）ics2_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 ics2_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。ics2_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFD00 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。ics2_RL78G14.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_R5F104Lx_SCI0_P51_P50
#deifne    ICS_R5F104Lx_SCI0_P17_P16
#define    ICS_R5F104Lx_SCI0_P12_P11
#define    ICS_R5F104Lx_SCI1_P02_P03
#define    ICS_R5F104Lx_SCI2_P77_P76
#define    ICS_R5F104Lx_SCI2_P13_P14
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78G1F ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

```
#include "iodefine.h"
#include "ics2_RL78G14.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma address    dtc_tbl = 0xFFD00
char    dtc_tbl[0xD0];

void main(void)
{
    ics2_init(0xFD00, ICS_R5F104Lx_SCI0_P51_P50, 2, 0x40, brr, 0);
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

割込関数において

```
{
    // 制御ルーチン開始

    // 制御ルーチン終了


    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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

```
#pragma interrupt Excep_INTST0    (vect=INTST0)
#pragma interrupt Excep_INTSR0    (vect=INTSR0)
#pragma interrupt Excep_INTSRE0    (vect=INTSRE0)

void    Excep_INTST0(void)    { int_ics_sci_tx();    }
```

```
void    Excep_INTSR0(void)    { int_ics_sci_rx();    }
void    Excep_INTSRE0(void)    { int_ics_sci_err();    }
```

SCI1 の場合

```
#pragma interrupt Excep_INTST1    (vect=INTST1)
#pragma interrupt Excep_INTSR1    (vect=INTSR1)
#pragma interrupt Excep_INTSRE1    (vect=INTSRE1)

void    Excep_INTST1(void)    { int_ics_sci_tx();    }
void    Excep_INTSR1(void)    { int_ics_sci_rx();    }
void    Excep_INTSRE1(void)    { int_ics_sci_err();    }
```

SCI2の場合

```
#pragma interrupt Excep_INTST2    (vect=INTST2)
#pragma interrupt Excep_INTSR2    (vect=INTSR2)
#pragma interrupt Excep_INTSRE2    (vect=INTSRE2)

void    Excep_INTST2(void)    { int_ics_sci_tx();    }
void    Excep_INTSR2(void)    { int_ics_sci_rx();    }
void    Excep_INTSRE2(void)    { int_ics_sci_err();    }
```

5.3.5. RL78G14 EWRL V3.10 コンパイラ 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例を紹介します。詳細は、サンプルコードを参照してください。

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、0xFFD00 から 256 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 location=0xFFD00
__no_init volatile char dtc_tbl[256];
```

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）ics2_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 ics2_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。ics2_init()関数には、RAM アドレスの下位 16bit を設定します。ここでは、0xFD00 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, 0xE0, 0xE8, 0xF0, 0xF8 が使用可能です。
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。ics2_RL78G1F.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_R5F104Lx_SCI0_P51_P50
#deifne    ICS_R5F104Lx_SCI0_P17_P16
#define    ICS_R5F104Lx_SCI0_P12_P11
#define    ICS_R5F104Lx_SCI1_P02_P03
#define    ICS_R5F104Lx_SCI2_P77_P76
#define    ICS_R5F104Lx_SCI2_P13_P14
```

第５パラメータは、お使いになる ICS に合わせて通信速度を選択します。
第６パラメータは、使用する転送モードを使います。
現状の RL78G14 ライブラリでは、0, 4, 6 を選択可能です。

---------- List 1　main.c ----------

```
#include  "ics2_RL78G14.h"

/*********** KEEP DTC TABLE AREA ***********/
#pragma location=0xFFD00
__no_init volatile char dtc_tbl[256];

void  main(void)
{
  ics2_init(0xFD00, ICS_R5F104Lx_SCI0_P51_P50, 2, 0x40, 2, 0);  //W2002 5.333Mbps @32MHz
}
```

３）ics2_watchpoint()関数の組込み

W2002 の場合、65us 以上 5ms 以下の間隔で呼び出されるように、ics2_watchpoint()を呼び出してください。それ以上の周期で呼び出される場合、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。

※注意　ics2_watchpoint()関数の実行中、RPECTL =0x80; とパリティーチェック機能を OFF しないとパリティーエラーによるリセットがかかる可能性があります。

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

割込関数において

```
{
    // 制御ルーチン開始

    // 制御ルーチン終了


    int_cnt = int_cnt +1;
    if (int_cnt>2)
    {
        int_cnt = 0;
        RPECTL = 0x80U;
        ics2_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();}
```

6. 改訂履歴

バージョン	変更日	変更内容
Ver.1.00JP	2018-03-23	・RL78G1F Ver3.70 ライブラリ　初版作成
Ver.1.01JP	2018-07-03	・RL78F14 暫定対応版
Ver.1.02JP	2018-11-20	・Page13 モード数の誤記修正 ・RL78F14, CC-RL, CA78K0R 対応
Ver.1.03JP	2019-03-28	・RL78G14　対応版
Ver.1.04JP	2019-09-25	・DTC address mode の行を削除：すべての CPU ・Edian の行を削除：すべての CPU ・使用資源の記述を修正（内容は同じ）
Ver.1.05JP	2020-06-25	・Parity チェック機能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