78K0 ファミリから RL78 ファミリへの置き換えガイド（CcnvCA78K0）

R01AN3471JJ0100
Rev.1.00
2016.09.30

要旨

本アプリケーションノートでは、78K0 用プログラムを RL78 用プログラムに置き換える方法について説明します。

対象デバイス

78K0 ファミリ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. 78K0 ファミリから RL78 ファミリへのプログラム置き換え方法

78K0 用プログラムを RL78 用プログラムに置き換える方法について説明します。
最初に、C ソースコンバータ CcnvCA78K0 を使用して、C コンパイラ CA78K0（または CC78K0）用拡張機能を C コンパイラ CC-RL 用拡張機能に変換します。
次に、統合開発環境 CS+または e2studio でプロジェクトを作成します。78K0 ファミリと RL78 ファミリは周辺機能が異なるため 78K0 ファミリの周辺機能用プログラムを使用せず、RL78 ファミリ用コード生成ツールを利用して RL78 ファミリの周辺機能用プログラムを生成します。
CcnvCA78K0 で変換したプログラムと上記周辺機能用プログラムを組み合わせて、プログラムを置き換えます。

2. CcnvCA78K0 によるプログラム変換

2.1 CcnvCA78K0 について

CcnvCA78K0 は、CA78K0（または CC78K0）用 C ソース・プログラムに対して、拡張言語仕様であるマクロ名、予約語、#pragma 指令、拡張機能の記述を CC-RL の拡張言語仕様に変換します。

なお、CcnvCA78K0 は、CA78K0 用プログラムから CC-RL 用プログラムへの移行を支援するためのソフトウェアです。変換後のプログラムが完全に動作することを保証されていません。必ず、変換後のプログラムでシステムの動作確認をしてください。

また、配置アドレス、SFR へのアクセスおよびアセンブラ記述などデバイスに依存した記述は変換できません。必要に応じて、手動で RL78 ファミリ用に変換してください。

詳細は、「C ソースコンバータ CcnvCA78K0 ユーザーズマニュアル(R20UT3684JJ0100)」を参照してください。

2.2 CcnvCA78K0 の使用方法

CcnvCA78K0 を使用したプログラムの変換方法を下記に示します。

(1) CcnvCA78K0(CcnvCA78K0.exe)と CA78K0 用プログラムを任意の同じフォルダに置きます。

(2) Windows のコマンドプロンプトを起動します。

(3) CcnvCA78K0 が格納されているフォルダへカレントディレクトリを変更します。

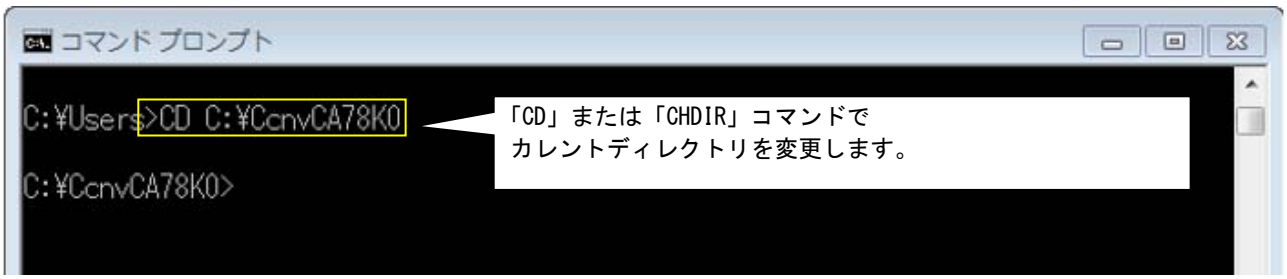


図 1.1 コマンドプロンプト画面

(4) -o オプションで出力ファイル名を指定して実行します。実行後、CC-RL 用プログラムが出力されます。
また、メッセージを指定したファイルに出力する場合は、-r オプションを指定します。

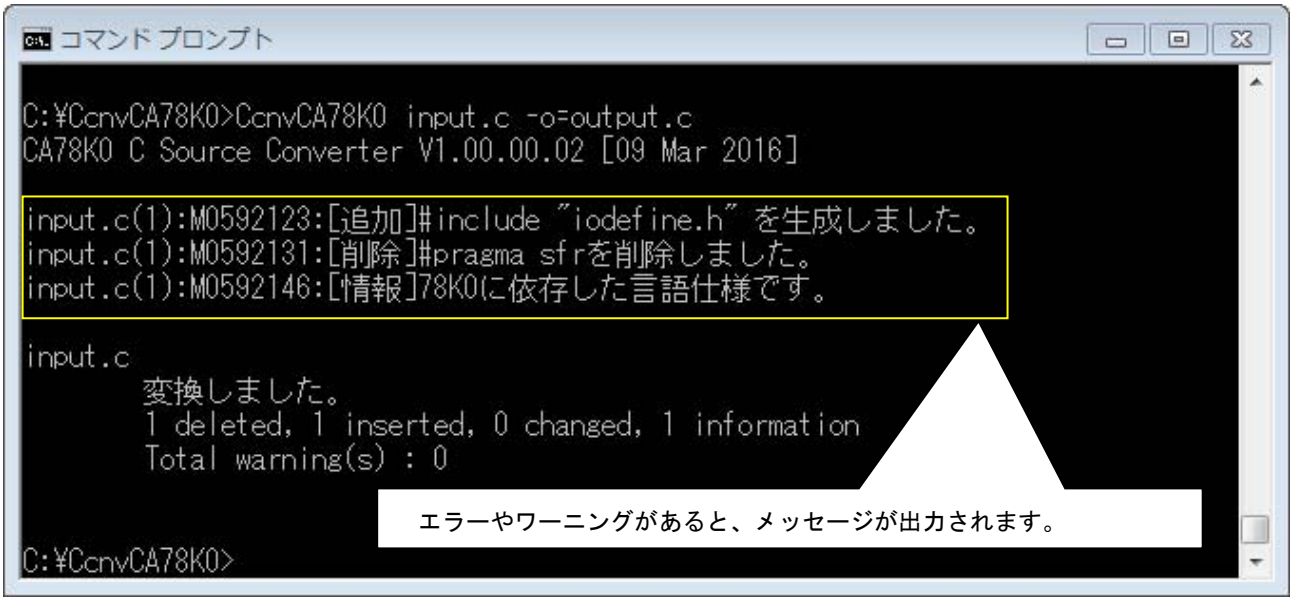


図 1.2 CcnvCA78K0 実行画面

(5)　複数のファイルを同時に変換する場合は、リスト・ファイルを作成し、-l オプションを実行します。
実行後、指定したフォルダに CC-RL 用プログラムが出力されます。

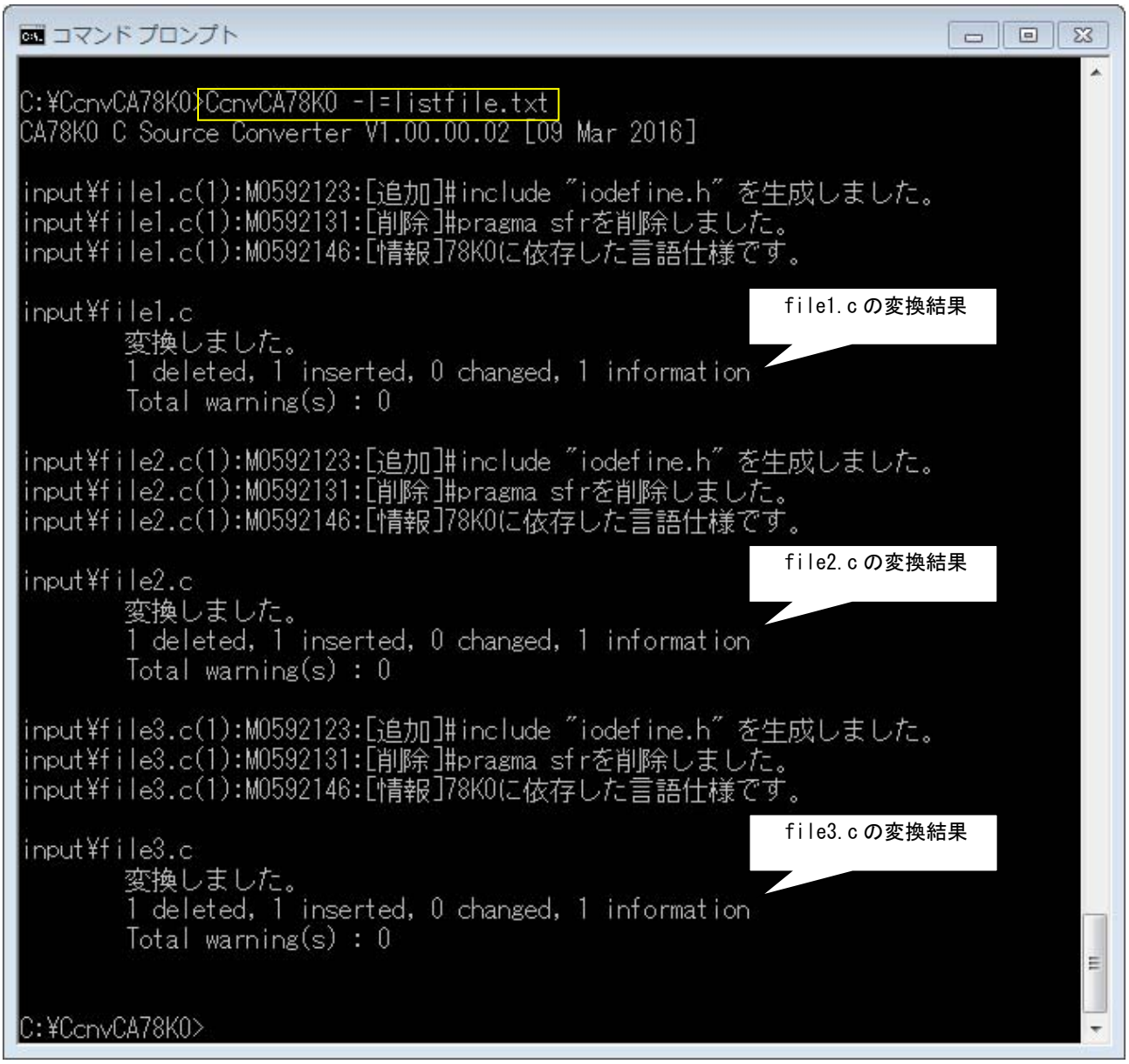


図 1.3　CcnvCA78K0 実行画面(複数ファイル)

リスト・ファイルの記載例は下記となります。

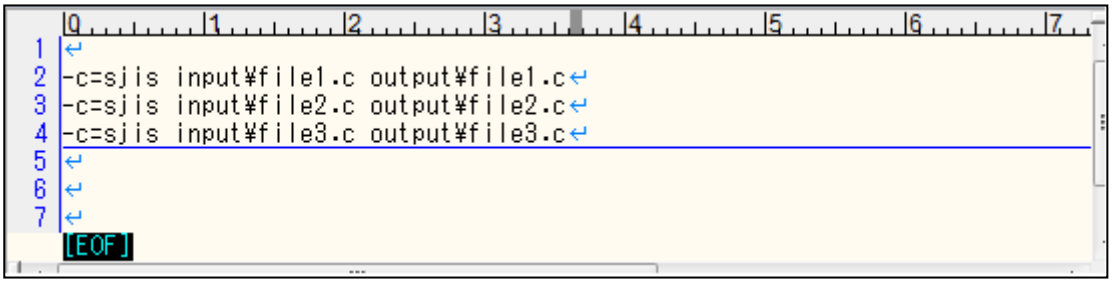


図 1.4　リスト・ファイル記載例

(6) CcnvCA78K0 で変換されない箇所について修正します。修正箇所は「C ソースコンバータ CcnvCA78K0 ユーザーズマニュアル(R20UT3684JJ0100)」の「コンバータ変換仕様」を参照してください。

2.3 CcnvCA78K0 を使用しない場合

CcnvCA78K0 を使用しない場合は、CA78K0（または CC78K0）用拡張機能を手動で CC-RL 用拡張機能に変換しなければなりません。CC-RL の拡張言語仕様については、「CC-RL コンパイラ ユーザーズマニュアル(R20UT3123JJ0103)」を参照してください。

3. 周辺機能のプログラム変換

3.1 プログラムの自動生成

78K0 ファミリで使用していた周辺機能と同種類の RL78 ファミリの周辺機能に対して、統合開発環境 CS+ または e2studio の RL78 ファミリ用コード生成ツールでプログラムの自動生成を行います。コード生成ツールの操作方法については、「CS+コード生成ツール 統合開発環境 ユーザーズマニュアル 周辺機能操作編 (R20UT3104JJ0100)」を参照してください。

(1) プロジェクト・ツリーの中からコード生成(設計ツール)の「クロック発生回路」をクリックし、「端子の割り当て設定」を行います。
なお、端子割り当て設定を一度確定させると端子割り当て設定は変更できません。

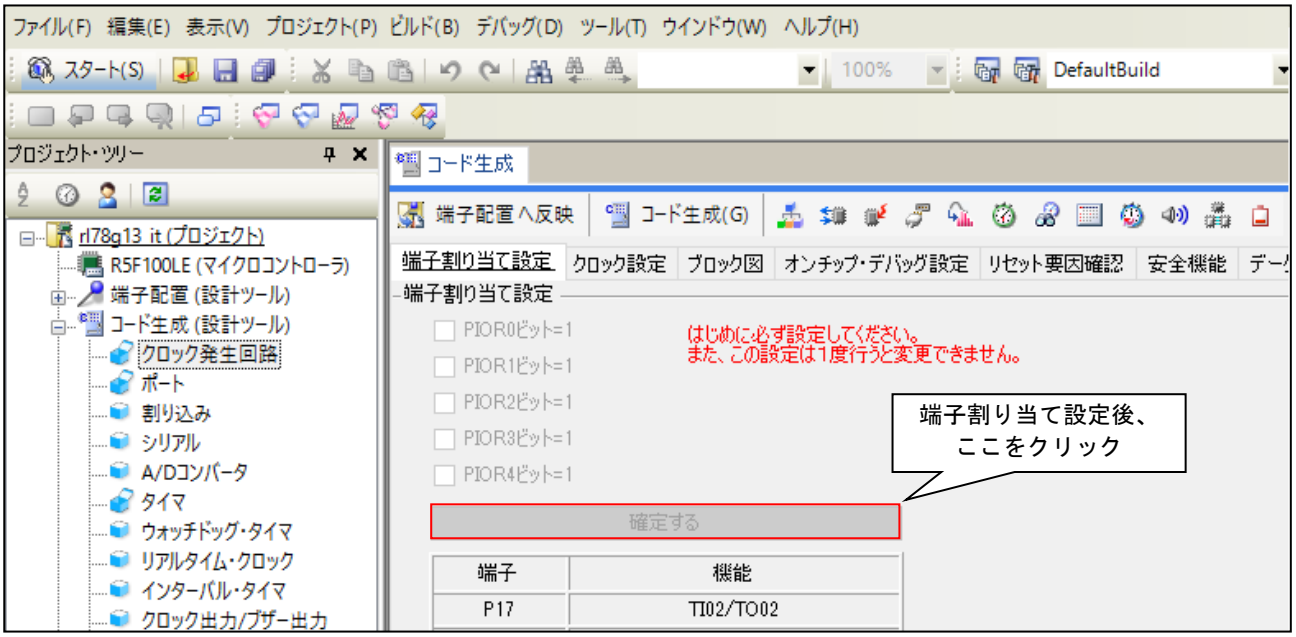


図 3.1 コード生成ツールの設定画面(1)

(2) 78K0 ファミリのプログラムを参照して、各機能の設定を行います。

(3)　全ての周辺機能の設定が完了したら、画面上部にある「コード生成(G)」ボタンをクリックして、コード生成(プログラムの自動生成)を行います。自動生成した周辺機能の関数をプログラムの置き換えに使用します。

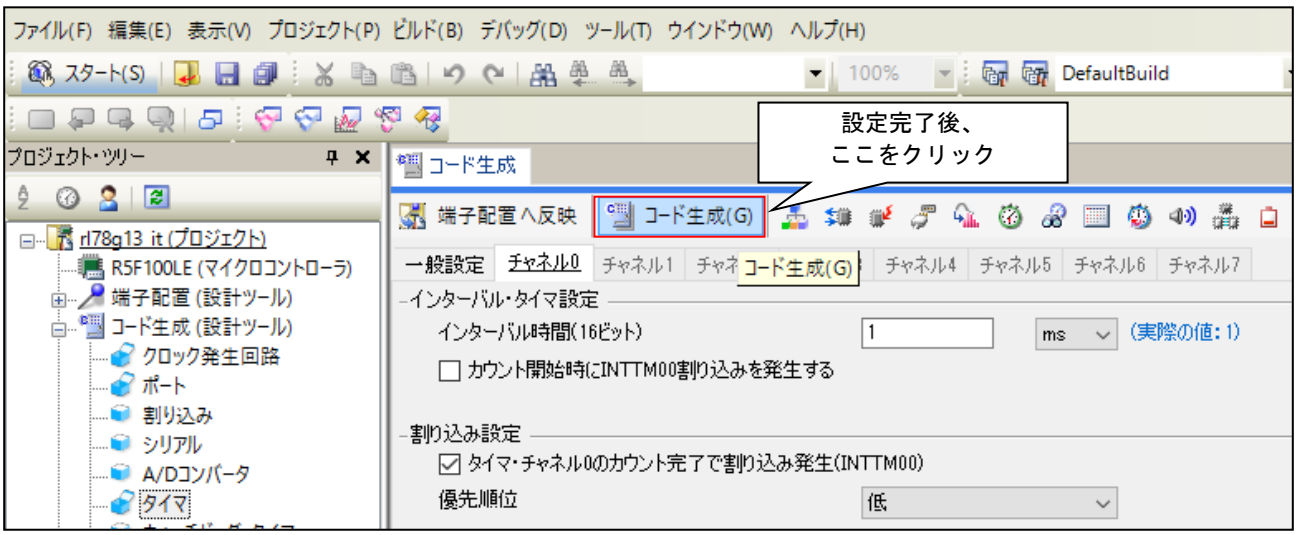


図 3.2　コード生成ツールの設定画面(2)

3.2 プログラムの追加

コード生成ツールで自動生成できないプログラム（main 関数、割り込み関数処理、変数など）を追加します。

自動生成された各ファイルの”/* Start user code for adding. Do not edit comment generated here */”と”/* End user code. Do not edit comment generated here */”の間にプログラムを追加します。プログラム追加は、手動で行う必要があります。なお、上記範囲外に追加したプログラムは、プログラムの自動生成時に、自動的に削除されます。

必ず、追加したプログラムでシステムの動作確認をしてください。

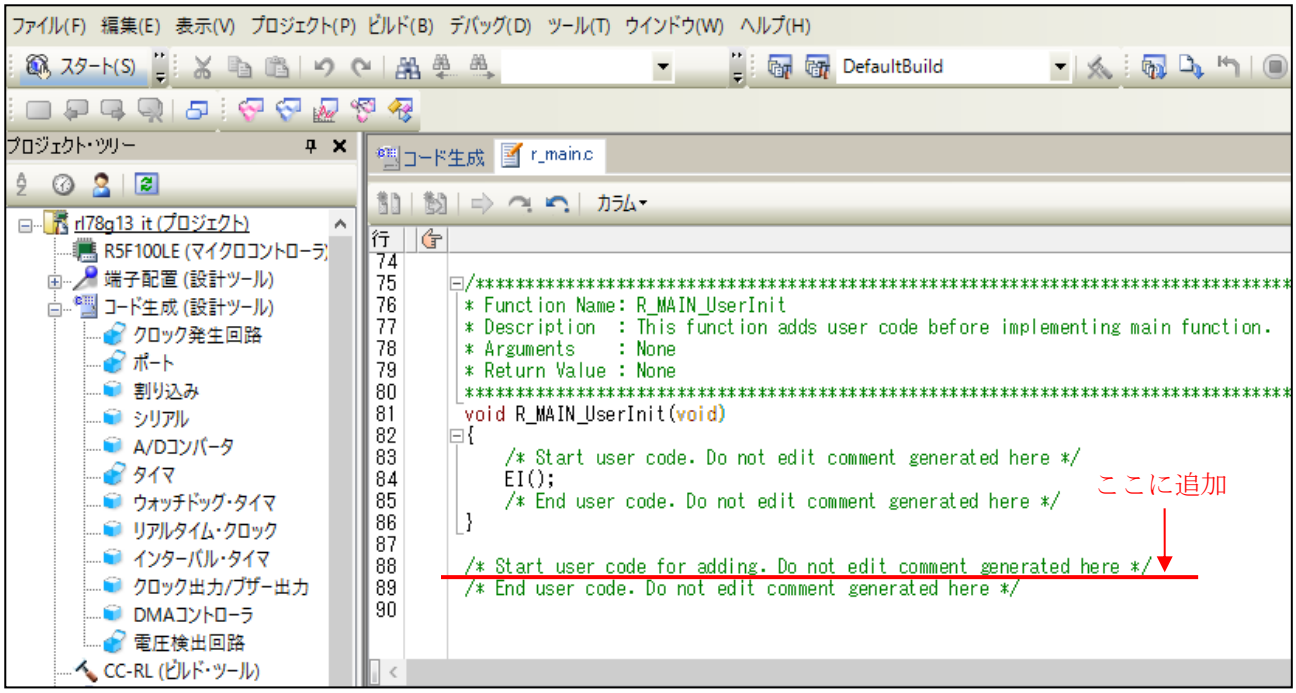


図 3.3 既存のプログラムの追加

3.3 コード生成ツールを使用しない場合

コード生成ツールを使用しない場合は、統合開発環境 CS+または e2studio で新規プロジェクトを作成した後、手動で周辺機能用プログラムを作成しなければなりません。周辺機能の詳細については、RL78 ファミリのユーザーズマニュアルを確認してください。

4. 置き換え例

4.1 78K0/Kx1 サンプル・プログラム(シリアル・インタフェース UART0)

78K0/Kx1,78K0/Kx1+シリアル通信プログラム集に掲載されているシリアル・インタフェース UART0 のプログラムを RL78/G13 用プログラムに置き換えます。置き換え後のプロジェクトファイルは「r01an3471_rl78g13_serial」です。

このプログラムでは UART0 を使用し、2ms 経過毎に 9600bps の転送速度で 0x55 のデータ送信を繰り返します。CPU クロックは高速システム・クロックの 10MHz です。

4.1.1 CcnvCA78K0 を使用した CC-RL へのソース移植

(1) リスト・ファイルを作成し、変換する C ソース・ファイルを指定します。

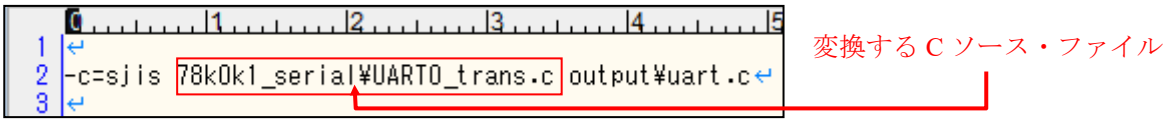


図 4.1 リスト・ファイルの記載例（シリアル・インタフェース UART0）

(2) コマンドプロンプトを起動し、リスト・ファイルで指定した C ソース・ファイルを変換します。また、出力した変換結果ファイルに変更箇所が記載されます。

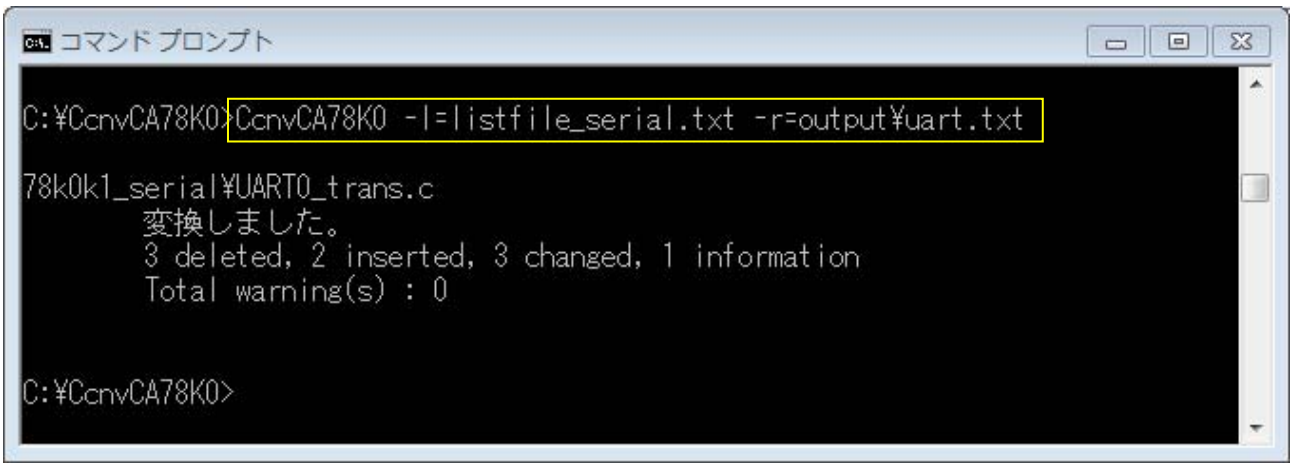


図 4.2 CcnvCA78K0 実行画面(シリアル・インタフェース UART0)

変換結果ファイルには下記のように変換結果が記載されます。変換結果の詳細については、「C ソースコンバータ CcnvCA78K0 ユーザーズマニュアル(R20UT3684JJ0100)」を参照してください。

```
CA78K0 C Source Converter V1.00.00.02 [09 Mar 2016]
78k0k1_serial¥UART0_trans.c(16):M0592123:[追加]#include "iodefine.h" を生成しました。
78k0k1_serial¥UART0_trans.c(16):M0592131:[削除]#pragma sfrを削除しました。
78k0k1_serial¥UART0_trans.c(16):M0592146:[情報]78K0に依存した言語仕様です。
78k0k1_serial¥UART0_trans.c(17):M0592131:[削除]#pragma nopを削除しました。
78k0k1_serial¥UART0_trans.c(18):M0592131:[削除]#pragma eiを削除しました。
78k0k1_serial¥UART0_trans.c(51):M0592121:[追加]I/Oレジスタのビット・アクセス用マクロを生成しました。
78k0k1_serial¥UART0_trans.c(51):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
78k0k1_serial¥UART0_trans.c(91):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
78k0k1_serial¥UART0_trans.c(92):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
```

図 4.3 変換結果の詳細（シリアル・インタフェース UART0）

(3)　変換した C ソース・ファイルを修正します。

SFR および saddr 変数に対するビット・アクセスは、下記のようにビットフィールドの型宣言とマクロに置き換えられます。8 ビットの SFR に対してビット・アクセスを行う場合は、unsigned int を unsigned char に変更します。

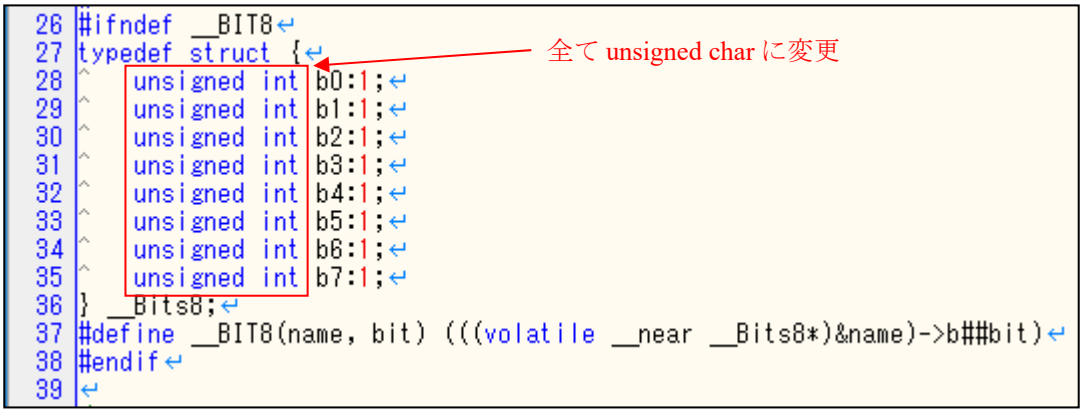


図 4.4　ビット・アクセスの記述変更

4.1.2 プログラムの自動生成

(1) 統合開発環境 CS+または e2studio で新規プロジェクトを作成します。

(2) コード生成ツールで各機能を設定します。

CPU クロックを高速システム・クロック 10MHz に設定します。

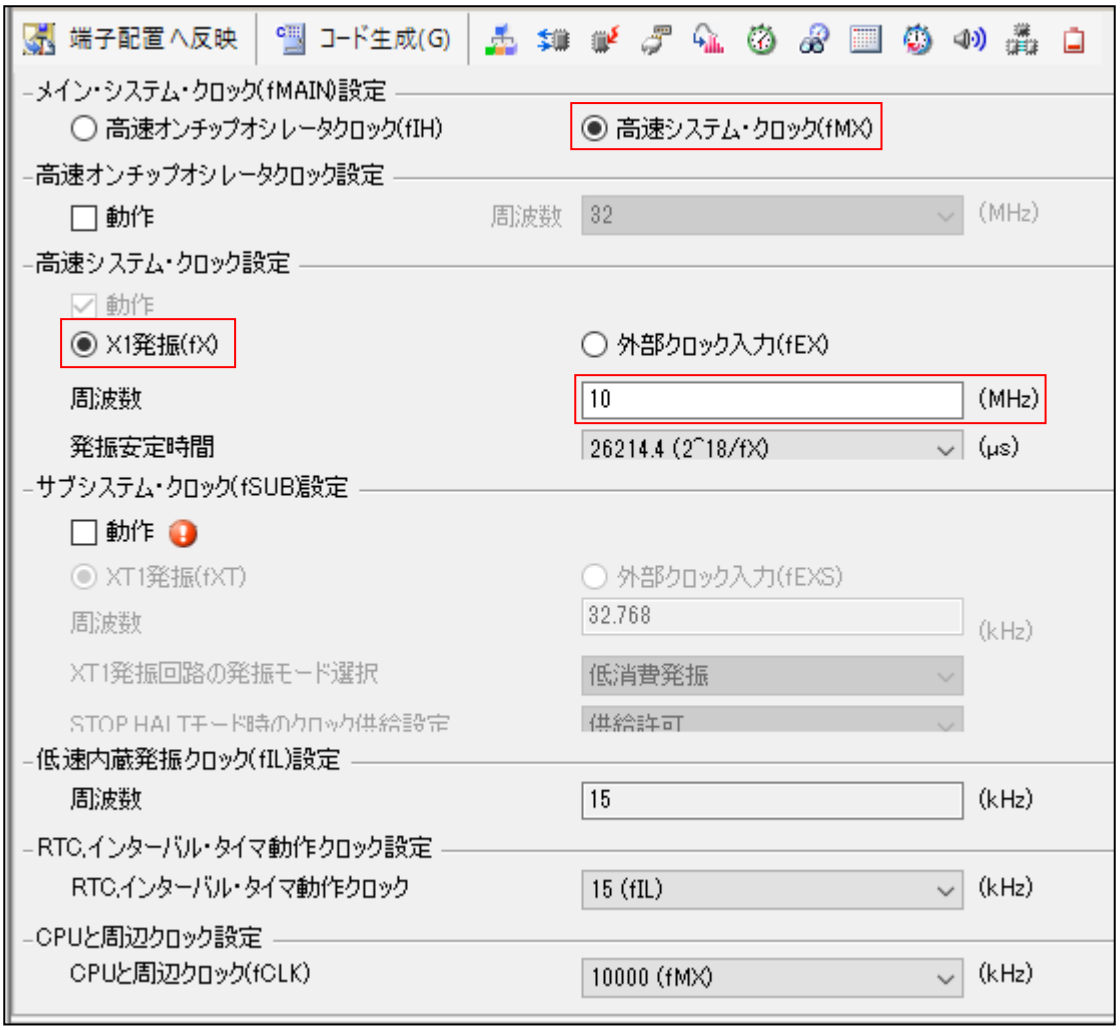


図 4.5 コード生成ツール設定画面（クロック）

78K0 ファミリのシリアル・インタフェース UART0 と同等機能であるシリアル・アレイ・ユニットの UART0 を設定します。

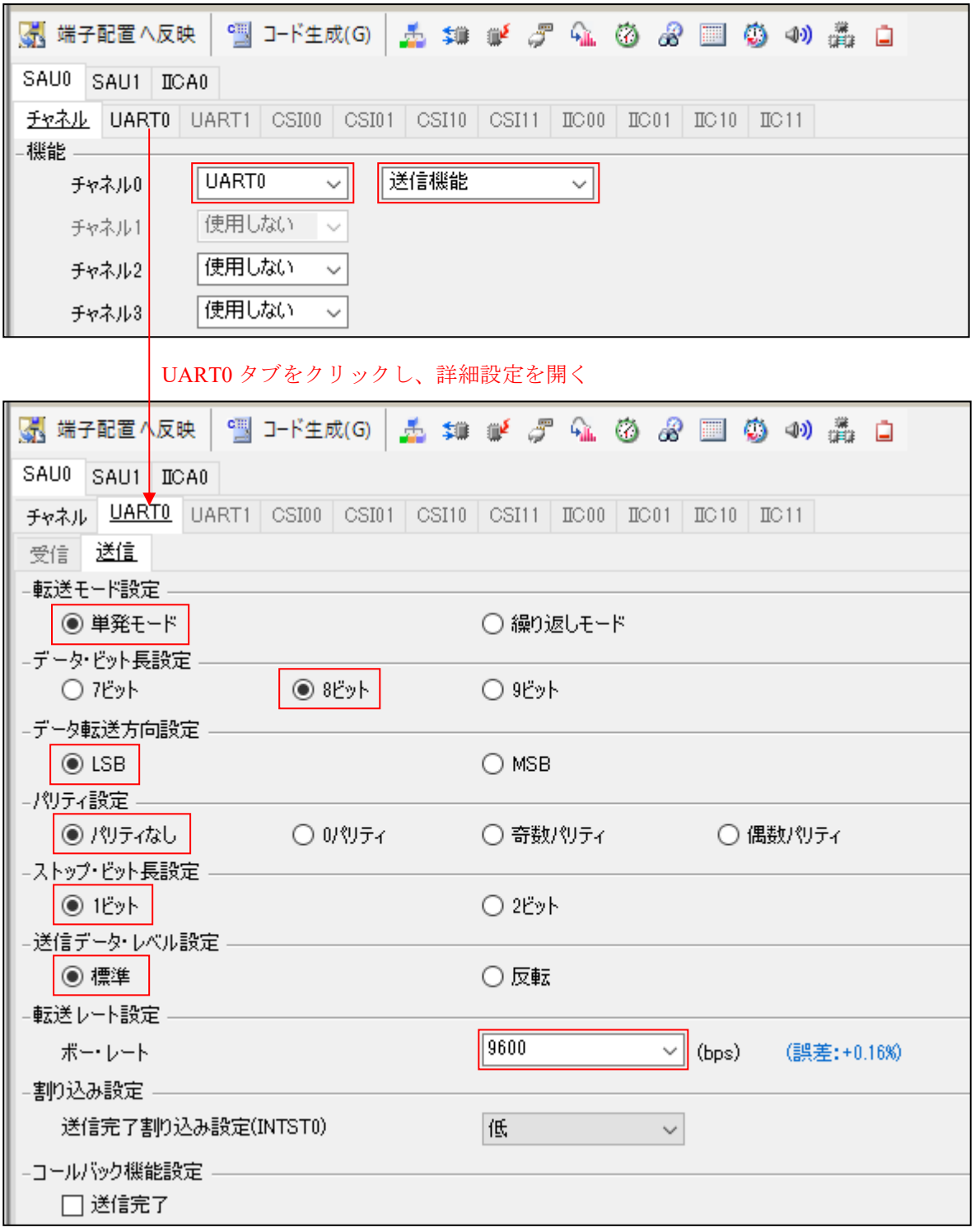


図 4.6　コード生成ツール設定画面(シリアル・アレイ・ユニット)

(3)　その他の「ポート」、「ウォッチドッグ・タイマ」、「電圧検出回路」をそれぞれ設定します。

(4)　「コード生成(G)」をクリックし、ファイルを生成します。

4.1.3 プログラム追加

コード生成したプログラムにシンボル定義と main 関数の処理を追加します。その他のプログラム（クロック設定や UART0 機能の設定等）については、コード生成したプログラムを使用します。

・シンボル定義

シンボル定義を r_cg_userdefine.h に追加します。

78K0 用プログラム

```
61 ------------------------------------------------------------------------
62     Constants/Variables
63 ------------------------------------------------------------------------
64 */
65
66 #define UART_BAUDRATE_M0     0x3
67 #define UART_BAUDRATE_K0     0x10
68
69 /*status list definition*/
70 #define TRUE                 1
71 #define FALSE                0
```

RL78/G13 用 r_cg_userdefine.h ファイル

```
32 /***********************************************************************
33 User definitions
34 ***********************************************************************
35
36 /* Start user code for function. Do not edit comment generated here */
37 #define TRUE                 1
38 #define FALSE                0
39 /* End user code. Do not edit comment generated here */
```

図 4.7　シンボル定義の置き換え

・main 関数

RL78/G13 用コード生成ツールを使用すると、main 関数実行前に R_Systeminit 関数が実行されます。R_Systeminit 関数では、クロックや UART0 の初期設定を行っています。このため、赤枠の処理のみ手動でプログラムを追加します。R_UART_Start 関数では、UART0 動作開始を行います。

78K0 用プログラム

```
82 void main( void )
83 {
84     unsigned short i;
85
86     PCC  = 0x00;                        /* CPU clock: fx */
87     WDTM = 0x77;                        /* Watchdog Timer Stop */
88
89     /* Waiting for oscillation stable time */
90 //[CcnvCA78K0]     while( OSTC.0 == 0);
91     while( __BIT8( OSTC, 0 ) == 0);
92     MCM0 = 1;                           /* supply clock: X1 */
93     /* Waiting for X1 clock change */
94     while( MCS == 0 );
95
96     UART0_Init();                       /* UART0 initialization function */
97     UART0_Enable();                     /* UART0 enable function */
98
99     while(TRUE)                         /* main loop */
100    {
101        TXS0 = 0x55;
102        /* Waiting for the completion of transmitting */
103        while( DUALIF0 == 0 );
104        DUALIF0 = 0;
105
106        for( i=0 ; i<1000 ; i++ );      /* wait 2 ms */
107    }
108 }
```

RL78/G13 の r_main.c ファイル

```
59 void main(void)
60 {
61     R_MAIN_UserInit();
62     /* Start user code. Do not edit comment generated here */
63     {
64         unsigned short i = 0;
65
66         R_UART0_Start();
67
68         while(TRUE)                         /* main loop */
69         {
70             TXD0 = 0x55;
71             /* Waiting for the completion of transmitting */
72             while( STIF0 == 0 );
73             STIF0 = 0;
74
75             for( i=0 ; i<4000 ; i++ );      /* wait 2 ms */
76         }
77     }
78     /* End user code. Do not edit comment generated here */
79 }
```

図 4.8　main 関数の置き換え

4.1.4 その他修正事項

(1) コード生成ツールで UART0 を設定した場合、割り込み処理が自動生成されます。今回は、割り込みを使用していないため、割り込み処理を無効します。

(2) 割り込み機能を使用しないため、R_MAIN_UserInit 関数の「EI();」を「DI();」に変更します。

(3) ソフトウェアタイマの処理時間を再調整します。コンパイラが異なるため、処理時間が変わる可能性があります。

4.1.5 置き換え後のサンプルコード

ルネサス エレクトロニクスホームページからサンプルコード「an-r01an3471jj0100-rl78-migrate.zip」を入手してください。「workspace」フォルダ内の「rl78g13_migrate_serial」が 78K0/Kx1,78K0/Kx1+シリアル通信プログラム集に掲載されているシリアル・インタフェース UART0 のプログラムを置き換えたサンプルコードとなります。

4.2　78K0/Kx2 サンプル・プログラム（インターバル・タイマ）

78K0/Kx2 サンプル・プログラム インターバル・タイマ編(U19031JJ2V0AN00)に掲載されているプログラムを RL78/G13 用プログラムに置き換えます。置き換え後のプロジェクトファイルは「r01an3471_rl78g13_timer」です。

このプログラムでは、16 ビット・タイマ/イベント・カウンタ 00 を使用し、1ms 毎にインターバル割り込みを発生させます。インターバル割り込み処理内で P10 を反転させます。また、インターバル割り込み 100 回発生毎に P11 を反転させる処理を行います。

4.2.1　CcnvCA78K0 を使用した CC-RL へのソース移植

(1)　リスト・ファイルを作成し、変換する C ソース・ファイルを指定します。

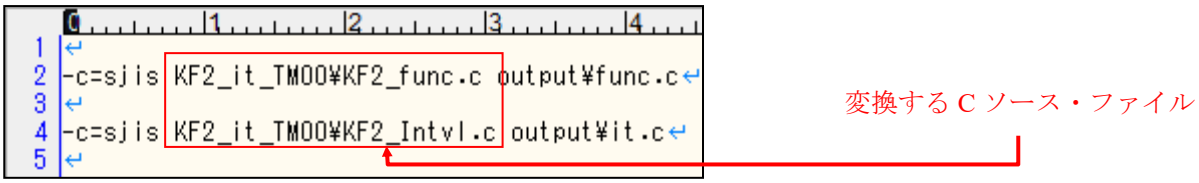


図 4.9　リスト・ファイルの記載例（インターバル・タイマ）

(2)　コマンドプロンプトを起動し、リスト・ファイルで指定した C ソース・ファイルを変換します。また、出力した変換結果ファイルに変更箇所が記載されます。

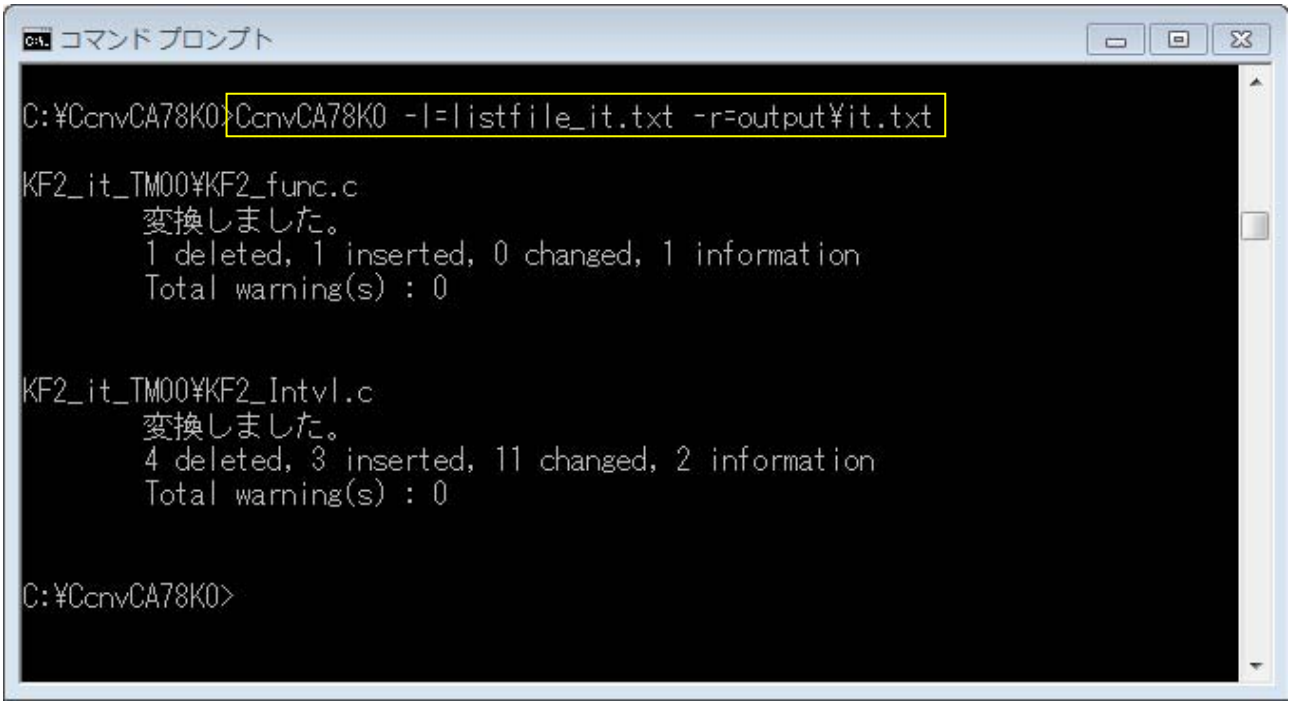


図 4.10　CcnvCA78K0 実行画面（インターバル・タイマ）

変換結果ファイルには下記のように変換結果が記載されます。変換結果の詳細については、「C ソースコンバータ CcnvCA78K0 ユーザーズマニュアル(R20UT3684JJ0100)」を参照してください。

```
CA78K0 C Source Converter V1.00.00.02 [09 Mar 2016]
KF2_it_TM00¥KF2_func.c(16):M0592123:[追加]#include "iodefine.h" を生成しました。
KF2_it_TM00¥KF2_func.c(16):M0592131:[削除]#pragma sfrを削除しました。
KF2_it_TM00¥KF2_func.c(16):M0592146:[情報]78K0に依存した言語仕様です。
KF2_it_TM00¥KF2_Intvl.c(47):M0592123:[追加]#include "iodefine.h" を生成しました。
KF2_it_TM00¥KF2_Intvl.c(47):M0592131:[削除]#pragma sfrを削除しました。
KF2_it_TM00¥KF2_Intvl.c(47):M0592146:[情報]78K0に依存した言語仕様です。
KF2_it_TM00¥KF2_Intvl.c(48):M0592131:[削除]#pragma diを削除しました。
KF2_it_TM00¥KF2_Intvl.c(49):M0592131:[削除]#pragma eiを削除しました。
KF2_it_TM00¥KF2_Intvl.c(50):M0592131:[削除]#pragma nopを削除しました。
KF2_it_TM00¥KF2_Intvl.c(51):M0592113:[変更]#pragma interruptをCC-RLの形式に変更しました。
KF2_it_TM00¥KF2_Intvl.c(51):M0592146:[情報]78K0に依存した言語仕様です。
KF2_it_TM00¥KF2_Intvl.c(87):M0592111:[変更]DIを__DIに変換しました。
KF2_it_TM00¥KF2_Intvl.c(274):M0592111:[変更]EIを__EIに変換しました。
KF2_it_TM00¥KF2_Intvl.c(277):M0592111:[変更]NOPを__nopに変換しました。
KF2_it_TM00¥KF2_Intvl.c(287):M0592122:[追加]#pragma interrupt NO_VECTを生成しました。
KF2_it_TM00¥KF2_Intvl.c(287):M0592113:[変更]__interruptをCC-RLの形式に変更しました。
KF2_it_TM00¥KF2_Intvl.c(291):M0592121:[追加]I/Oレジスタのビット・アクセス用マクロを生成しました。
KF2_it_TM00¥KF2_Intvl.c(291):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
KF2_it_TM00¥KF2_Intvl.c(291):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
KF2_it_TM00¥KF2_Intvl.c(292):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
KF2_it_TM00¥KF2_Intvl.c(301):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
KF2_it_TM00¥KF2_Intvl.c(301):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
KF2_it_TM00¥KF2_Intvl.c(302):M0592112:[変更]I/Oレジスタのビット指定アクセスをマクロ呼び出しに変換しました。
```

図 4.11　変換結果の詳細（インターバル・タイマ）

(3)　変換した C ソース・ファイルを修正します。

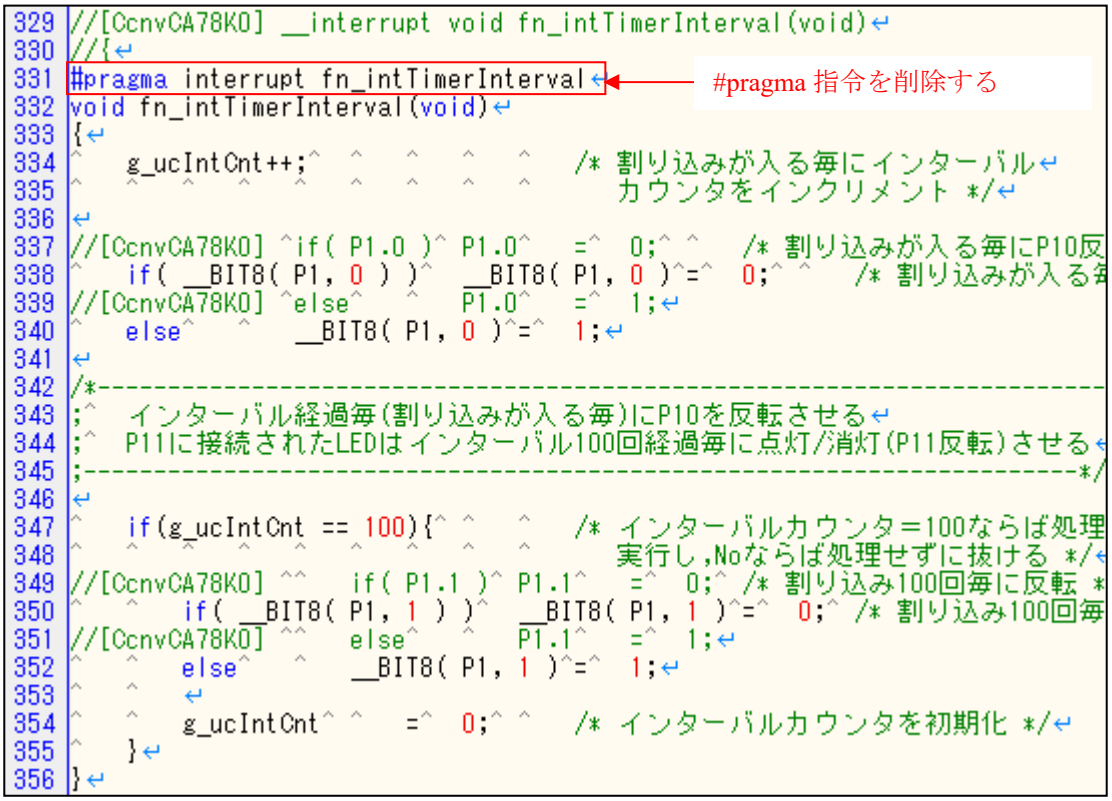
SFR および saddr 変数に対するビット・アクセスは、下記のようにビットフィールドの型宣言とマクロに置き換えられます。8 ビットの SFR に対してビット・アクセスを行う場合は、unsigned int を unsigned char に変更します。

```
//[CcnvCA78K0]
#include "iodefine.h"
#ifndef __BIT8
typedef struct {
    unsigned int b0:1;
    unsigned int b1:1;
    unsigned int b2:1;
    unsigned int b3:1;
    unsigned int b4:1;
    unsigned int b5:1;
    unsigned int b6:1;
    unsigned int b7:1;
} __Bits8;
#define __BIT8(name, bit) (((volatile __near __Bits8*)&name)->b##bit)
#endif
```

全て unsigned char に変更

図 4.12　ビット・アクセスの記述変更

割り込み関数宣言が重複する場合があります。CC-RL ではエラーとなるため、変換した#pragma 指令を削除します。




```
329 //[CcnvCA78K0] __interrupt void fn_intTimerInterval(void)
330 //{
331 #pragma interrupt fn_intTimerInterval
332 void fn_intTimerInterval(void)
333 {
334     g_ucIntCnt++;        /* 割り込みが入る毎にインターバル
335                             カウンタをインクリメント */
336
337 //[CcnvCA78K0] if( P1.0 ) P1.0   =  0;    /* 割り込みが入る毎にP10反
338     if( __BIT8( P1, 0 ) )   __BIT8( P1, 0 )=  0;      /* 割り込みが入る
339 //[CcnvCA78K0] else       P1.0   =  1;
340     else        __BIT8( P1, 0 )=  1;
341
342 /*-----------------------------------------------------------------------
343 ;  インターバル経過毎(割り込みが入る毎)にP10を反転させる
344 ;  P11に接続されたLEDはインターバル100回経過毎に点灯/消灯(P11反転)させる
345 ;-----------------------------------------------------------------------*/
346
347     if(g_ucIntCnt == 100){      /* インターバルカウンタ＝100ならば処理
348                                 実行し,Noならば処理せずに抜ける */
349 //[CcnvCA78K0]     if( P1.1 ) P1.1   =  0; /* 割り込み100回毎に反転 *
350         if( __BIT8( P1, 1 ) )   __BIT8( P1, 1 )=  0; /* 割り込み100回毎
351 //[CcnvCA78K0]     else       P1.1   =  1;
352         else        __BIT8( P1, 1 )=  1;
353
354         g_ucIntCnt  =  0;       /* インターバルカウンタを初期化 */
355     }
356 }
```

図 4.13　割り込み関数宣言の変更

4.2.2　　プログラムの自動生成

(1)　統合開発環境 CS+または e2studio で新規プロジェクトを作成します。

(2)　コード生成ツールで各機能を設定します。

CPU クロックを高速オンチップ・オシレータ・クロック 8MHz に設定します。

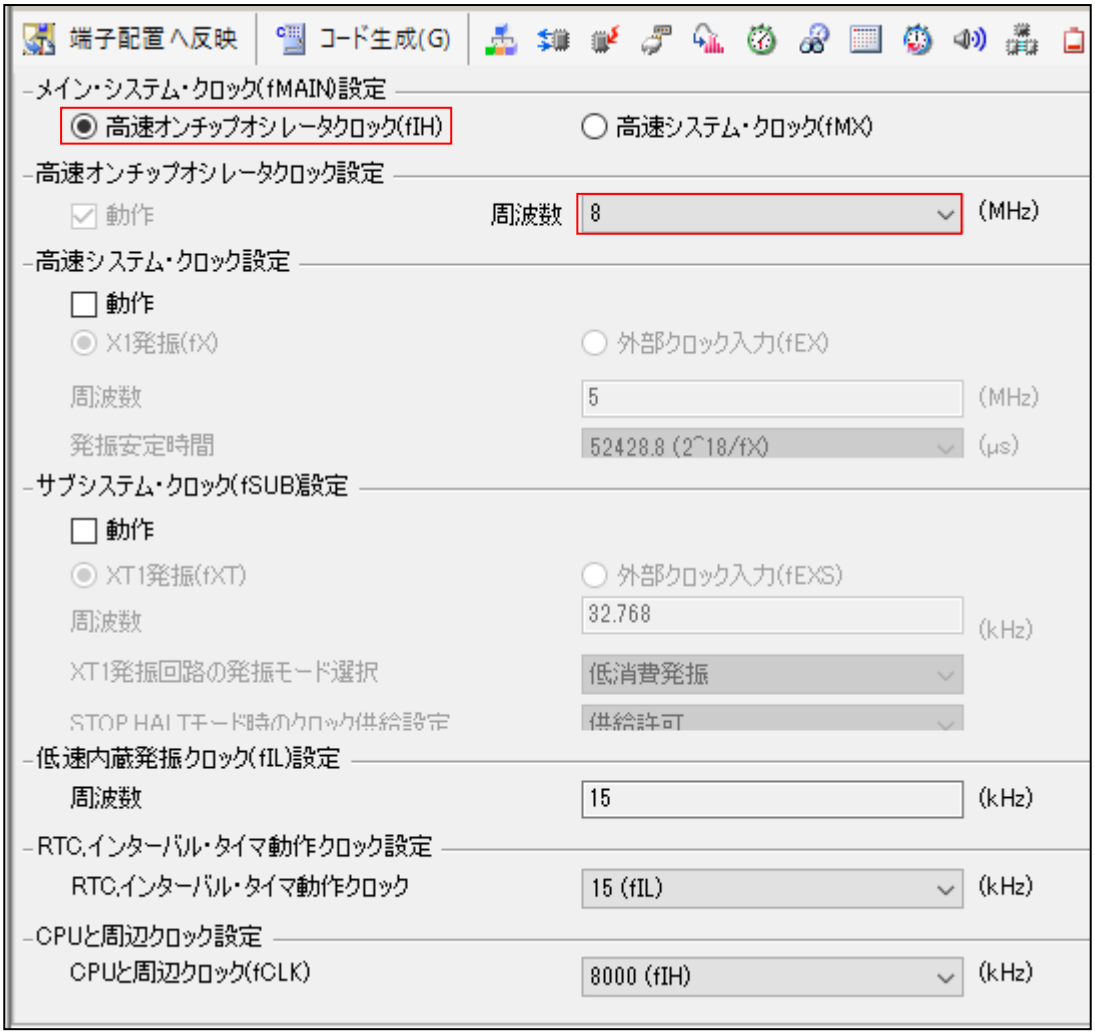


図 4.14　コード生成ツール設定画面（クロック）

78K0 ファミリの 16 ビット・タイマ/イベント・タイマ 00(TM00)と同等機能であるタイマ・アレイ・ユニットのインターバル・タイマ機能を設定します。

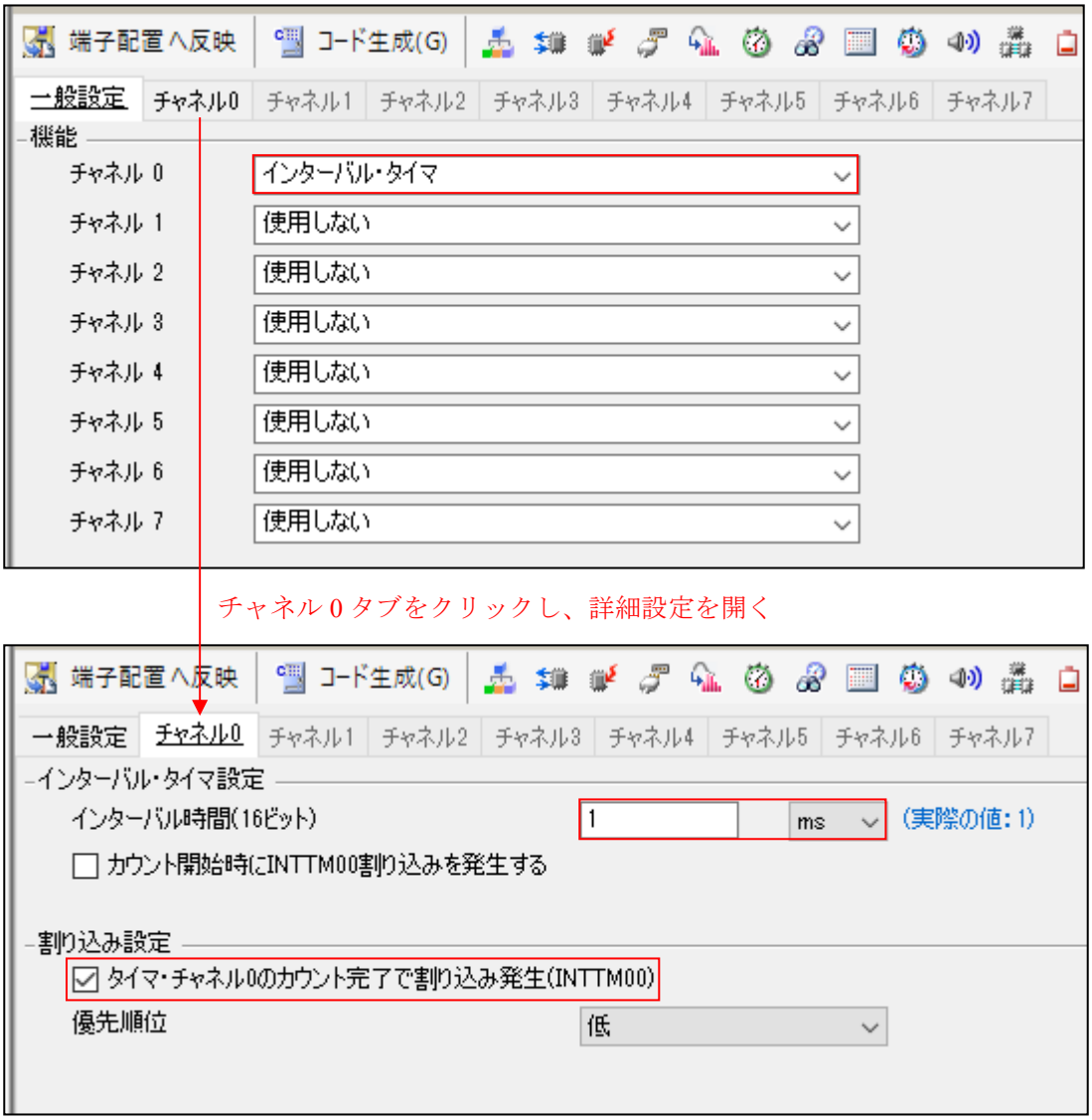


図 4.15　コード生成ツール設定画面(タイマ・アレイ・ユニット)

(3)　その他の「ポート」、「ウォッチドッグ・タイマ」、「電圧検出回路」をそれぞれ設定します。

(4)　コード生成をクリックし、ファイルを生成します。

4.2.3 プログラム追加

コード生成したプログラムに変数と main 関数、割り込み関数の処理を追加します。その他のプログラム（クロック設定やタイマ・アレイ・ユニットの設定等）については、コード生成したプログラムを使用します。

・変数

変数を r_main.c、 r_cg_timer_user.c に追加します。また、ビットフィールドの型宣言についても r_cg_timer_user.c に追加します。

78K0 用プログラム

```
112 /*=============================================================================
113 ;^  使用する変数の定義(グローバル変数の定義)
114 ;=============================================================================*/
115 static^ unsigned^   char^   g_ucIntCnt;
116 ^   ^   ^   ^   ^   ^   ^   ^   /* インターバルカウンタ
117 ^   ^   ^   ^   ^   ^   ^   ^   (割り込みをカウントするカウンタ) */
118
```

RL78/G13 の r_main.c ファイル

```
46 /***********************************************************************************************
47 Global variables and functions
48 ***********************************************************************************************
49 /* Start user code for global. Do not edit comment generated here */
50 unsigned    char    g_ucIntCnt = 0;    /* インターバルカウンタ(割り込みをカウントす
51 /* End user code. Do not edit comment generated here */
```

RL78/G13 の r_cg_timer_user.c ファイル

```
45 /***********************************************************************************************
46 Global variables and functions
47 ***********************************************************************************************
48 /* Start user code for global. Do not edit comment generated here */
49 #ifndef __BIT8
50 typedef struct {
51     unsigned char b0:1;
52     unsigned char b1:1;
53     unsigned char b2:1;
54     unsigned char b3:1;
55     unsigned char b4:1;
56     unsigned char b5:1;
57     unsigned char b6:1;
58     unsigned char b7:1;
59 } __Bits8;
60 #define __BIT8(name, bit) (((volatile __near __Bits8*)&name)->b##bit)
61 #endif
62
63 extern    unsigned    char    g_ucIntCnt;
64
65 /* End user code. Do not edit comment generated here */
```

ビットフィールドの
型宣言

図 4.16 変数の置き換え

・main 関数

78K0 用プログラムの main 関数処理を RL78/13 の r_main.c の main 関数に追加します。タイマ・アレイ・ユニットの動作開始については、RL78/G13 用コード生成ツールで自動生成した R_TAU0_Channel_Start() に変更します。

78K0 用プログラム

```
311 void main(void){
312 
313     g_ucIntCnt = 0;          /* インターバルカウンタを初期化*/
314     fn_InitTimer();          /* インターバルタイマの初期化  */
315 //[CcnvCA78K0] EI();                        /* ベクタ割り込み許可 */
316     __EI();                  /* ベクタ割り込み許可          */
317 
318     while (1){
319 //[CcnvCA78K0]     NOP();
320         __nop();
321     }
322 }
```

RL78/G13 の r_main.c ファイル

```
60 void main(void)
61 {
62     R_MAIN_UserInit();
63     /* Start user code. Do not edit comment generated here */
64     g_ucIntCnt = 0;              /* インターバルカウンタを初期化 */
65     R_TAU0_Channel0_Start();     /* タイマ・アレイ・ユニット動作開始 */
66     __EI();                      /* ベクタ割り込み許可 */
67 
68     while (1)
69     {
70         __nop();
71     }
72     /* End user code. Do not edit comment generated here */
73 }
```

図 4.17　main 関数の置き換え

・割り込み関数

割り込み処理を r_cg_timer_user.c の r_tau0_channel0_interrupt()に追加します。

78K0 用プログラム

```
#pragma interrupt fn_intTimerInterval
void fn_intTimerInterval(void)
{
    g_ucIntCnt++;                /* 割り込みが入る毎にインターバル
                                    カウンタをインクリメント */

//[CcnvCA78K0] if( P1.0 ) P1.0   =  0;    /* 割り込みが入る毎にP10反転 */
    if( __BIT8( P1, 0 ) )   __BIT8( P1, 0 )=  0;      /* 割り込みが入る毎にP10反転 */
//[CcnvCA78K0] else       P1.0   =  1;
    else        __BIT8( P1, 0 )=  1;

/*-------------------------------------------------------------------------
;   インターバル経過毎(割り込みが入る毎)にP10を反転させる
;   P11に接続されたLEDはインターバル100回経過毎に点灯/消灯(P11反転)させる
;-------------------------------------------------------------------------*/

    if(g_ucIntCnt == 100){      /* インターバルカウンタ＝100ならば処理
                                   実行し,Noならば処理せずに抜ける */
//[CcnvCA78K0]    if( P1.1 ) P1.1   =  0;  /* 割り込み100回毎に反転 */
        if( __BIT8( P1, 1 ) )   __BIT8( P1, 1 )=  0;  /* 割り込み100回毎に反転 */
//[CcnvCA78K0]    else       P1.1   =  1;
        else        __BIT8( P1, 1 )=  1;

        g_ucIntCnt  =  0;       /* インターバルカウンタを初期化 */
    }
}
```

RL78/G13 の r_cg_timer_user.c

```
static void __near r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    g_ucIntCnt++;                   /* 割り込みが入る毎にインターバルカウンタをインクリメント */

    if( __BIT8( P1, 0 ) ) __BIT8( P1, 0 ) = 0;        /* 割り込みが入る毎にP10反転 */
    else __BIT8( P1, 0 ) = 1;

/*-------------------------------------------------------------------------
;    インターバル経過毎(割り込みが入る毎)にP10を反転させる
;    P11に接続されたLEDはインターバル100回経過毎に点灯/消灯(P11反転)させる
;-------------------------------------------------------------------------*/

    if(g_ucIntCnt == 100){          /* インターバルカウンタ＝100ならば処理実行し,Noならば処理せず
        if( __BIT8( P1, 1 ) ) __BIT8( P1, 1 ) = 0;    /* 割り込み100回毎に反転 */
        else __BIT8( P1, 1 ) = 1;

        g_ucIntCnt = 0;         /* インターバルカウンタを初期化 */
    }
    /* End user code. Do not edit comment generated here */
}
```

図 4.18　割り込み関数の置き換え

4.2.4　　置き換え後のサンプルコード

ルネサス エレクトロニクスホームページからサンプルコード「an-r01an3471jj0100-rl78-migrate.zip」を入手してください。「workspace」フォルダ内の「rl78g13_migrate_timer」が 78K0/Kx2 サンプル・プログラムインターバル・タイマ編(U19031JJ2V0AN00)に掲載されているプログラムを置き換えたサンプルコードとなります。

4.3　78K0/Kx2 サンプル・プログラム（A/D コンバータ）

78K0/Kx2 サンプル・プログラム A/D コンバータ(ZUD-CC-10-0016)に掲載されているプログラムを RL78/G13 用プログラムに置き換えます。置き換え後のプロジェクトファイルは「r01an3471_rl78g13_ad」です。

このプログラムでは、アナログ入力チャネルを 4 チャネル使用し、1ms 間隔毎にチャネルを変えて A/D 変換を行います。32ms を 1 周期とし、4 チャネル*8 回分のサンプリングを行った後、平均値を各変数に保存します。平均化された値に応じて、アナログ入力チャネルに対応する LED の点灯/消灯を行います。

4.3.1　CcnvCA78K0 を使用した CC-RL へのソース移植

(1)　リスト・ファイルを作成し、変換する C ソース・ファイルを指定します。

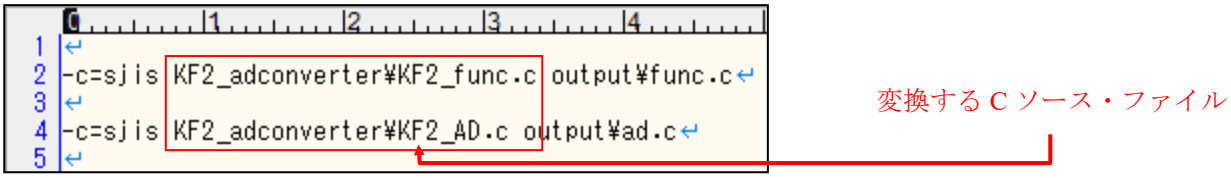


図 4.19　リスト・ファイルの記載例(A/D コンバータ)

(2)　コマンドプロンプトを起動し、リスト・ファイルで指定した C ソース・ファイルを変換します。また、出力した変換結果ファイルに変更箇所が記載されます。

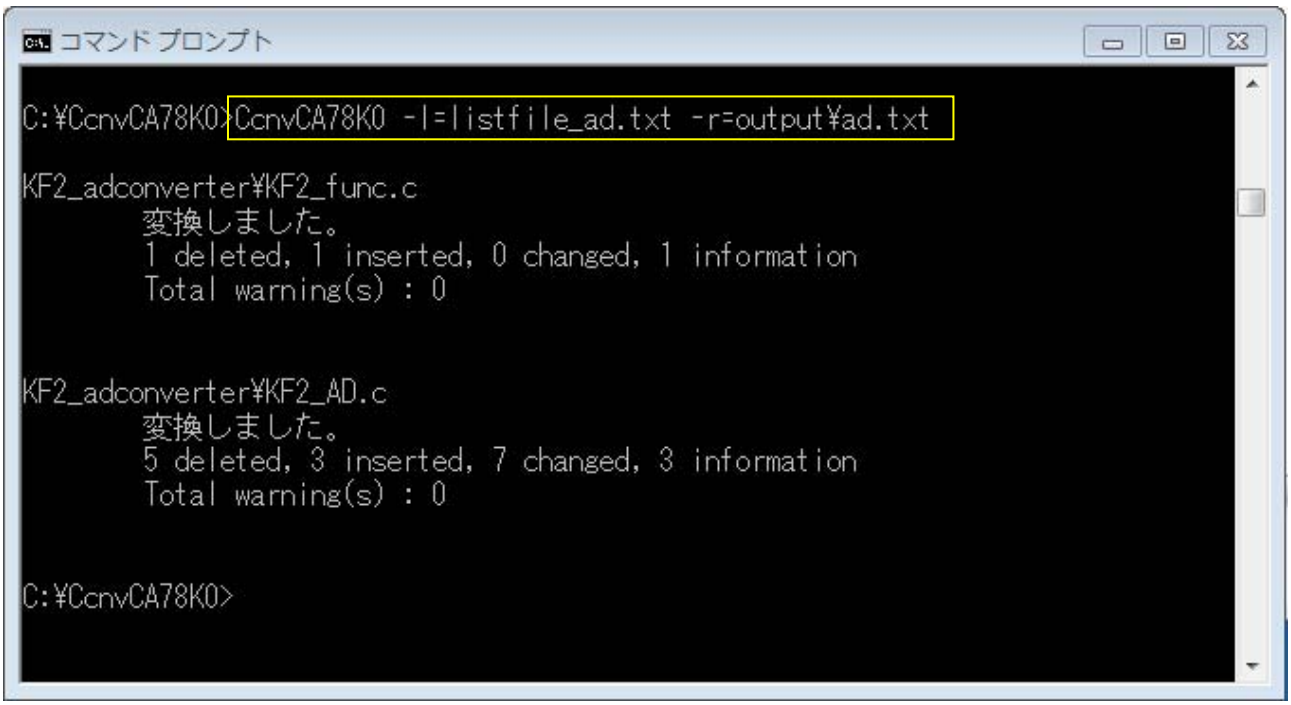


図 4.20　CcnvCA78K0 実行画面（A/D コンバータ）

変換結果ファイルには下記のように変換結果が記載されます。変換結果の詳細については、「C ソースコンバータ CcnvCA78K0 ユーザーズマニュアル(R20UT3684JJ0100)」を参照してください。

```
CA78K0 C Source Converter V1.00.00.02 [09 Mar 2016]
KF2_adconverter¥KF2_func.c(16):M0592123:[追加]#include "iodefine.h" を生成しました。
KF2_adconverter¥KF2_func.c(16):M0592131:[削除]#pragma sfrを削除しました。
KF2_adconverter¥KF2_func.c(16):M0592146:[情報]78K0に依存した言語仕様です。
KF2_adconverter¥KF2_AD.c(44):M0592123:[追加]#include "iodefine.h" を生成しました。
KF2_adconverter¥KF2_AD.c(44):M0592131:[削除]#pragma sfrを削除しました。
KF2_adconverter¥KF2_AD.c(44):M0592146:[情報]78K0に依存した言語仕様です。
KF2_adconverter¥KF2_AD.c(45):M0592131:[削除]#pragma diを削除しました。
KF2_adconverter¥KF2_AD.c(46):M0592131:[削除]#pragma eiを削除しました。
KF2_adconverter¥KF2_AD.c(47):M0592131:[削除]#pragma haltを削除しました。
KF2_adconverter¥KF2_AD.c(48):M0592131:[削除]#pragma nopを削除しました。
KF2_adconverter¥KF2_AD.c(49):M0592113:[変更]#pragma interruptをCC-RLの形式に変更しました。
KF2_adconverter¥KF2_AD.c(49):M0592146:[情報]78K0に依存した言語仕様です。
KF2_adconverter¥KF2_AD.c(51):M0592113:[変更]#pragma interruptをCC-RLの形式に変更しました。
KF2_adconverter¥KF2_AD.c(51):M0592146:[情報]78K0に依存した言語仕様です。
KF2_adconverter¥KF2_AD.c(87):M0592111:[変更]DIを__DIに変換しました。
KF2_adconverter¥KF2_AD.c(349):M0592111:[変更]EIを__EIに変換しました。
KF2_adconverter¥KF2_AD.c(352):M0592111:[変更]NOPを__nopに変換しました。
KF2_adconverter¥KF2_AD.c(361):M0592122:[追加]#pragma interrupt NO_VECTを生成しました。
KF2_adconverter¥KF2_AD.c(361):M0592113:[変更]__interruptをCC-RLの形式に変更しました。
KF2_adconverter¥KF2_AD.c(404):M0592122:[追加]#pragma interrupt NO_VECTを生成しました。
KF2_adconverter¥KF2_AD.c(404):M0592113:[変更]__interruptをCC-RLの形式に変更しました。
```

図 4.21　変換結果の詳細（A/D コンバータ）

(3)　変換した C ソース・ファイルを修正します。

割り込み関数宣言が重複する場合があります。CC-RL ではエラーとなるため、変換した#pragma 指令を削除します。

```
391 //[CcnvCA78K0] __interrupt void fn_intAdConverter(void)
392 //{
393 #pragma interrupt fn_intAdConverter          <-- #pragma 指令を削除する
394 void fn_intAdConverter(void)
395 {
396     if(g_ucAdCnt == 0){
397         for(g_ucAdCh = 0; g_ucAdCh < 4; g_ucAdCh++){
398             g_ucAdData[g_ucAdCh] = 0;
399                                     /* A/D変換結果バッファのクリア        *
400         }
401         g_ucAdCh = 0;
402     }
403
404     g_ucAdData[g_ucAdCh] += ADCR >> 6;
405                                     /* A/D変換結果の下位6bitを削除        *
406     g_ucAdCh++;                        /* A/D変換チャネルをインクリメント
407
408     g_ucAdCh &= 0b00000011;
409     ADS = g_ucAdCh;
410                                     /* A/D変換チャネル変更                *
411
412     g_ucAdCnt++;                    /* A/D変換回数をインクリメント        *
413
414     if(g_ucAdCnt >= 32){
415         for(g_ucAdCh = 0; g_ucAdCh < 4; g_ucAdCh++){
416
417             g_ucAdCnt = g_ucAdCnt << 1;              /* シフト移動 */
418             if(g_ucAdData[g_ucAdCh]    >= 612){          /* ANI端子が3V以
419                 g_ucAdCnt &= 0b11111110;
420             }
421             else{                               /* ANI端子が3V未満の場合 */
422                 g_ucAdCnt |= 0b00000001;
423             }
424         }
425         g_ucAdCnt &= 0b11111111;
426         P1 = g_ucAdCnt;
427         g_ucAdCnt = 0;          /* A/D変換回数をゼロクリア            */
428     }
429     ADCS = 0;                  /* AD変換停止                        */
430     ADIF = 0;
431 }
432 /*****************************************************************************
433 ;
434 ;    インターバルタイマ割り込み処理(INTTM000)
435 ;
436 ;*****************************************************************************/
437 //[CcnvCA78K0] __interrupt void fn_intTimerInterval(void)
438 //{
439 #pragma interrupt fn_intTimerInterval        <-- #pragma 指令を削除する
440 void fn_intTimerInterval(void)
441 {
442     ADCS    =    1;                 /* AD変換開始
443 }
```

図 4.22　割り込み関数宣言の変更

4.3.2　　プログラムの自動生成

(1)　統合開発環境 CS+または e2studio で新規プロジェクトを作成します。

(2)　コード生成ツールで各機能を設定します。

CPU クロックを高速オンチップ・オシレータ・クロック 8MHz に設定します。

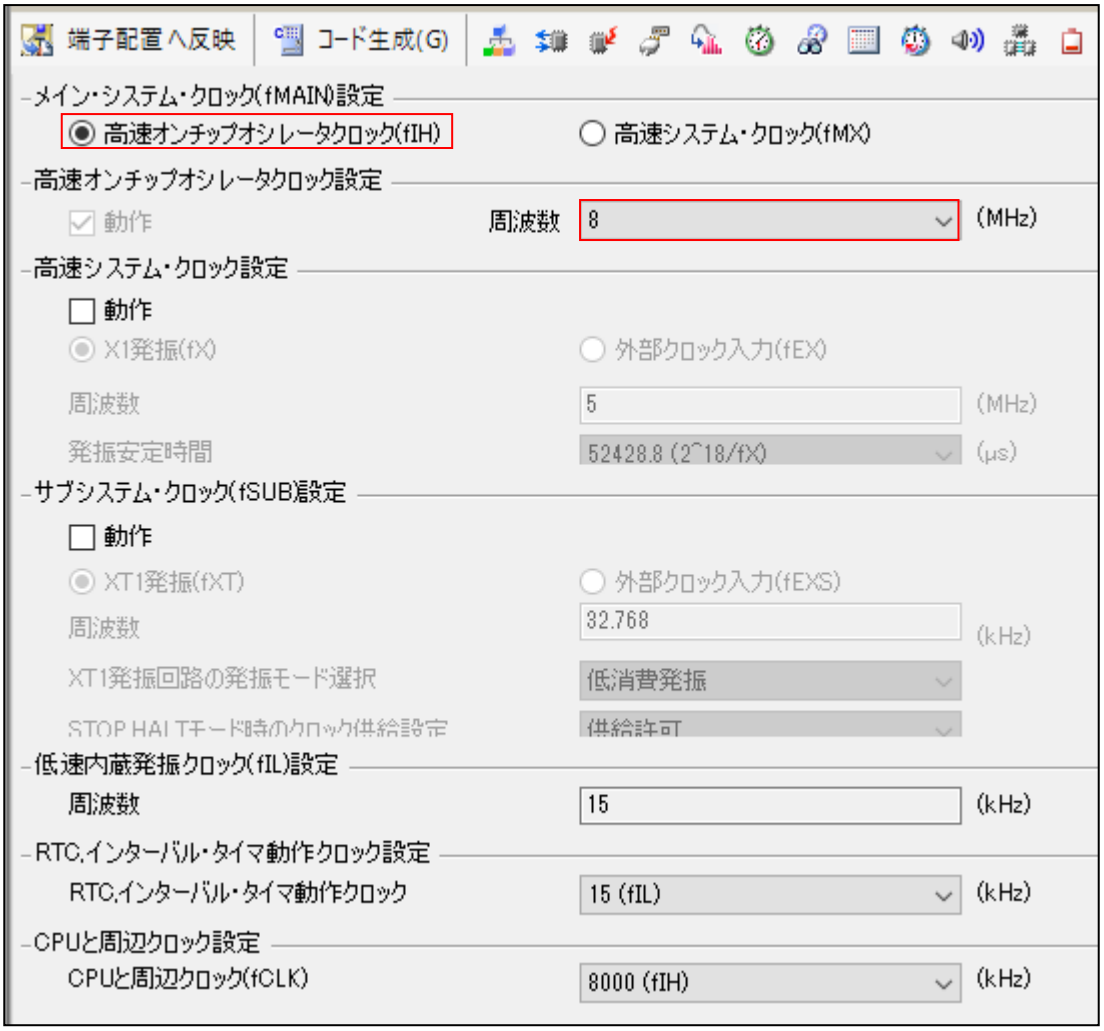


図 4.23　コード生成ツール設定画面（クロック）

A/D コンバータを設定します。

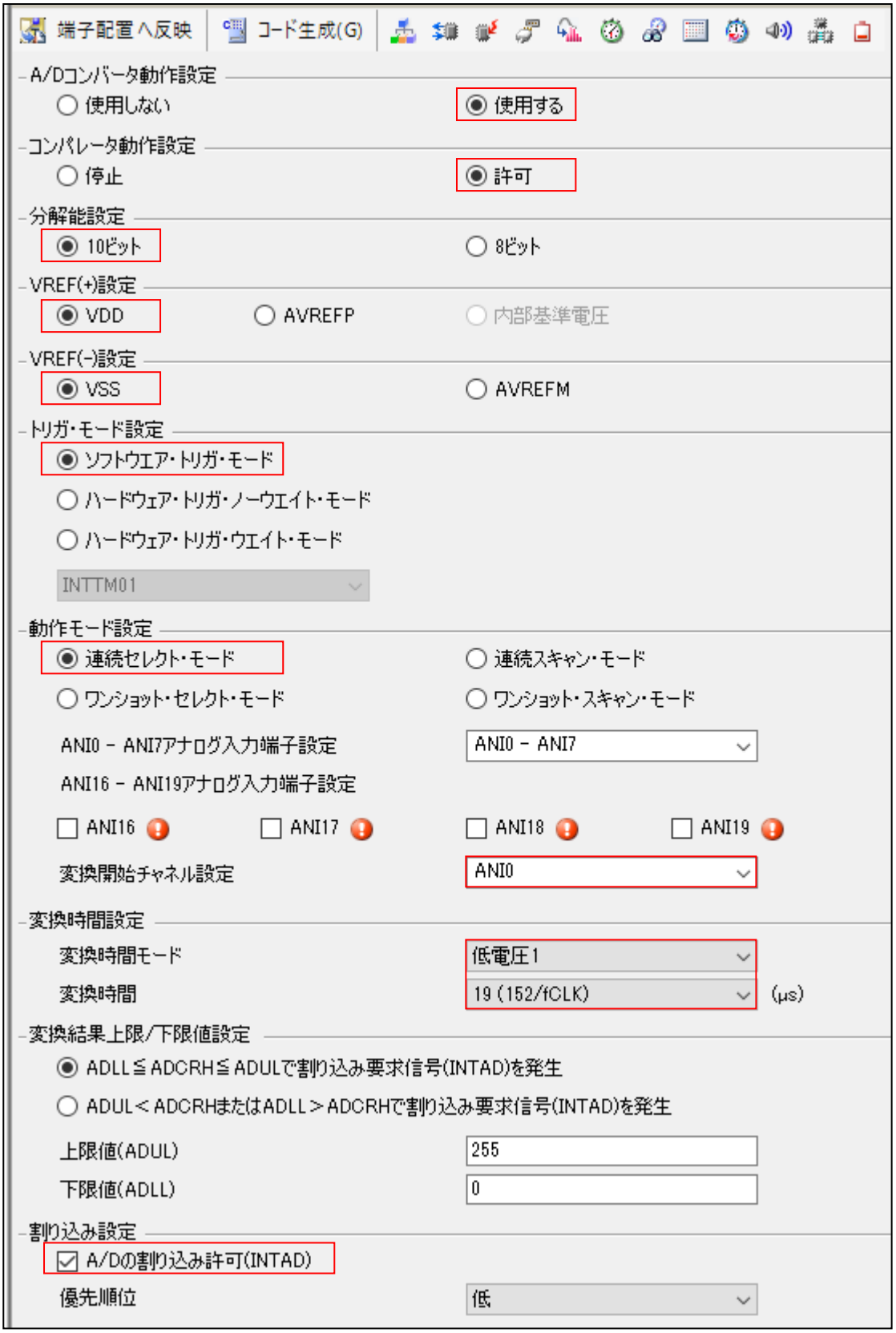


図 4.24　コード生成ツール設定画面（A/D コンバータ）

78K0 ファミリの 16 ビット・タイマ/イベント・タイマ 00(TM00)と同等機能となるタイマ・アレイ・ユニットのインターバル・タイマ機能を設定します。

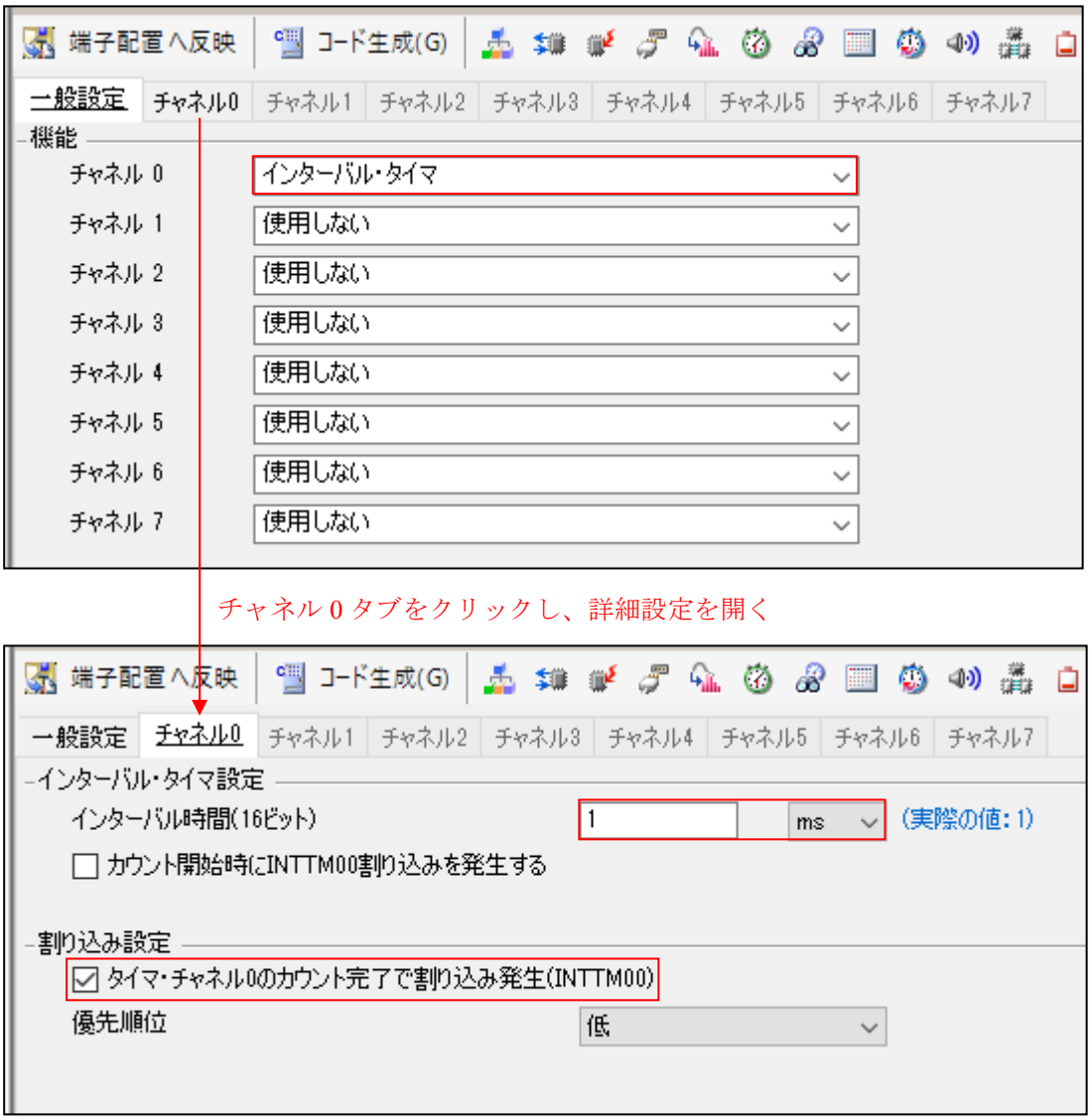


図 4.25　コード生成ツール設定画面(タイマ・アレイ・ユニット)

(3) その他の「ポート」、「ウォッチドッグ・タイマ」、「電圧検出回路」をそれぞれ設定します。

(4) コード生成をクリックし、ファイルを生成します。

4.3.3　　プログラム追加

コード生成したプログラムに変数と main 関数、割り込み関数の処理を追加します。その他のプログラム（クロック設定や A/D コンバータの設定等）については、コード生成したプログラムを使用します。

・変数

変数を r_main.c、 r_cg_adc_user.c に追加します。

78K0 用プログラム

```
100 /*=============================================================================
101 ;    使用する変数の定義(グローバル変数の定義)
102 ;=============================================================================*/
103 static    unsigned    char    g_ucAdCnt;
104 static    unsigned    char    g_ucAdCh;
105 static    unsigned    short    g_ucAdData[4];
```

RL78/G13 の r_main.c ファイル

```
47 /***********************************************************************
48 Global variables and functions
49 ***********************************************************************
50 /* Start user code for global. Do not edit comment generated here */
51 extern unsigned    char    g_ucAdCnt;
52
53 /* End user code. Do not edit comment generated here */
```

RL78/G13 の r_cg_adc_user.c ファイル

```
45 /***********************************************************************
46 Global variables and functions
47 ***********************************************************************
48 /* Start user code for global. Do not edit comment generated here */
49 unsigned char g_ucAdCnt = 0;
50 unsigned char g_ucAdCh = 0;
51 unsigned short g_ucAdData[4] = {0,0,0,0};
52 /* End user code. Do not edit comment generated here */
```

図 4.26　変数の置き換え

・main 関数

RL78/G13 用コード生成ツールを使用すると、main 関数実行前に R_Systeminit 関数が実行されます。R_Systeminit 関数では、タイマや A/D コンバータの初期設定を行っています。R_TAU0_Channel0_Start 関数でタイマ・アレイ・ユニットのカウント開始を行います。

78K0 用プログラム

```
373 void main(void){
374     fn_InitAd();
375     fn_InitTimer();
376     g_ucAdCnt     =     0;                  /* 変数初期化
377
378 //[CcnvCA78K0]      EI();                              /* ベクタ割り込み許可
379     __EI();                               /* ベクタ割り込み許可
380
381     while (1){
382 //[CcnvCA78K0]          NOP();
383         __nop();
384     }
385 }
```

変更

RL78/G13 の r_main.c ファイル

```
62 void main(void)
63 {
64     R_MAIN_UserInit();
65     /* Start user code. Do not edit comment generated here */
66     R_TAU0_Channel0_Start();
67     g_ucAdCnt = 0;                    /* 変数初期化 */
68
69     while (1)
70     {
71         __nop();
72     }
73     /* End user code. Do not edit comment generated here */
74 }
```

図 4.27　main の置き換え

・割り込み関数(タイマ・アレイ・ユニット)

割り込み関数処理を r_cg_timer_user.c の r_tau0_channel0_interrupt()に追加します。

78K0 用プログラム

```
440 void fn_intTimerInterval(void)↵
441 {↵
442     ADCS    =    1;                    /* AD変換開始                    */↵
443 }↵
```

RL78/G13 の r_cg_timer_user.c

```
57 static void __near r_tau0_channel0_interrupt(void)↵
58 {↵
59     /* Start user code. Do not edit comment generated here */↵
60     ADCS = 1;                          /* AD変換開始 */↵
61     /* End user code. Do not edit comment generated here */↵
62 }↵
```

図 4.28 割り込み関数(タイマ・アレイ・ユニット)の置き換え

・割り込み関数(A/D コンバータ)

割り込み処理を r_cg_adc_user.c の r_adc_interrupt()に追加します。

78K0 用プログラム

```
394 void fn_intAdConverter(void)
395 {
396     if(g_ucAdCnt == 0){
397         for(g_ucAdCh = 0; g_ucAdCh < 4; g_ucAdCh++){
398             g_ucAdData[g_ucAdCh] = 0;
399                                     /* A/D変換結果バッファのクリア        */
400         }
401         g_ucAdCh = 0;
402     }
403
404     g_ucAdData[g_ucAdCh] += ADCR >> 6;
405                                     /* A/D変換結果の下位6bitを削除        */
406     g_ucAdCh++;                        /* A/D変換チャネルをインクリメント    */
407
408     g_ucAdCh &= 0b00000011;
409     ADS = g_ucAdCh;
410                                     /* A/D変換チャネル変更                */
411
412     g_ucAdCnt++;                    /* A/D変換回数をインクリメント        */
413
414     if(g_ucAdCnt >= 32){
415         for(g_ucAdCh = 0; g_ucAdCh < 4; g_ucAdCh++){
416
417             g_ucAdCnt = g_ucAdCnt << 1;                /* シフト移動 */
418             g_ucAdData[g_ucAdCh] = g_ucAdData[g_ucAdCh] >> 3;    /* AD変換結果を平
419             if(g_ucAdData[g_ucAdCh]    >= 612){            /* ANI端子が3V以上の場
420                 g_ucAdCnt &= 0b11111110;
421             }
422             else{                              /* ANI端子が3V未満の場合 */
423                 g_ucAdCnt |= 0b00000001;
424             }
425         }
426         g_ucAdCnt &= 0b11111111;
427         P1 = g_ucAdCnt;
428         g_ucAdCnt = 0;            /* A/D変換回数をゼロクリア            */
429     }
430     ADCS = 0;                    /* AD変換停止                         */
431     ADIF = 0;
432 }
```

図 4.29　割り込み関数(A/D コンバータ)の置き換え(1/2)

RL78/G13 の r_adc_user.c ファイル

```
static void __near r_adc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    if(g_ucAdCnt == 0){
        for(g_ucAdCh = 0; g_ucAdCh < 4; g_ucAdCh++){
            g_ucAdData[g_ucAdCh] = 0;
                                    /* A/D変換結果バッファのクリア */
        }
        g_ucAdCh = 0;
    }

    g_ucAdData[g_ucAdCh] += ADCR >> 6;
                                    /* A/D変換結果の下位6bitを削除 */
    g_ucAdCh++;                     /* A/D変換チャネルをインクリメント */

    g_ucAdCh &= 0b00000011;
    ADS = g_ucAdCh;
                                    /* A/D変換チャネル変更 */

    g_ucAdCnt++;                    /* A/D変換回数をインクリメント */

    if(g_ucAdCnt >= 32){
        for(g_ucAdCh = 0; g_ucAdCh < 4; g_ucAdCh++){

            g_ucAdCnt = g_ucAdCnt << 1;                         /* シフト移
            g_ucAdData[g_ucAdCh] = g_ucAdData[g_ucAdCh] >> 3;   /* AD変換結
            if(g_ucAdData[g_ucAdCh]    >= 612){                 /* ANI端子が
                g_ucAdCnt &= 0b11111110;
            }
            else{                   /* ANI端子が3V未満の場合 */
                g_ucAdCnt |= 0b00000001;
            }
        }
        g_ucAdCnt &= 0b11111111;
        P1 = g_ucAdCnt;
        g_ucAdCnt = 0;              /* A/D変換回数をゼロクリア */
    }
    ADCS = 0;                       /* AD変換停止 */
    ADIF = 0;
    /* End user code. Do not edit comment generated here */
}
```

図 4.30　割り込み関数(A/D コンバータ)の置き換え(2/2)

4.3.4　置き換え後のサンプルコード

ルネサス エレクトロニクスホームページからサンプルコード「an-r01an3471jj0100-rl78-migrate.zip」を入手してください。「workspace」フォルダ内の「rl78g13_migrate_ad」が 78K0/Kx2 サンプル・プログラム A/D コンバータ(ZUD-CC-10-0016)に掲載されているプログラムを置き換えたサンプルコードとなります。

4.4　サンプル・プログラムの動作確認条件

置き換え後のサンプルコードは下記の条件で動作を確認しています。

表 4.1　動作確認条件

項目	内容
使用マイコン	RL78/G13（R5F100LEA）
統合開発環境 (CS+)	ルネサス エレクトロニクス製 CS+ for CC V4.00.00
C コンパイラ (CS+)	ルネサス エレクトロニクス製 CC-RL V1.02.00
統合開発環境 (e^2 studio)	ルネサス エレクトロニクス製 e^2 studio V4.3.1.001
C コンパイラ (e^2 studio)	ルネサス エレクトロニクス製 CC-RL V1.02.00
使用ボード	ルネサス エレクトロニクス製 RL78/G13 ターゲットボード(QB-R5F100LE-TB)

5.　サンプルコード

サンプルコードは、ルネサス エレクトロニクスホームページから入手してください。

6.　参考ドキュメント

RL78 ファミリ ユーザーズマニュアル ソフトウェア編（R01US0015J）

RL78 コンパイラ CC-RL ユーザーズマニュアル(R20UT3123J)

CS+コード生成ツール 統合開発環境ユーザーズマニュアル 周辺機能操作編(R20UT3104J)

C ソースコンバータ CcnvCA78K0 ユーザーズマニュアル(R20UT3684J)

78K0/Kx1,78K0/Kx1+ シリアル通信プログラム集

78K0/Kx2 サンプル・プログラム インターバル・タイマ編(U19031JJ2V0AN00)

78K0/Kx2 サンプル・プログラム A/D コンバータ(ZUD-CC-10-0016)

（最新版をルネサス エレクトロニクスホームページから入手してください。）

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

http://japan.renesas.com/

お問合せ先

http://japan.renesas.com/inquiry

改訂記録	78K0 ファミリから RL78 ファミリへの置き換えガイド （CcnvCA78K0）

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.09.30	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部 ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 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、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれら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。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため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起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。 たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。 なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品は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。また、当社製品は耐放射線設計については行っておりません。当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、社会的損害等を生じさせないよう、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設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。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、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当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に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当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禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/