お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用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等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
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、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。
5. 輸出に際しては、「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。本資料に記載されている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の目的で使用しないでください。また、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に使用することができません。
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。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されてい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生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   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、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当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、当社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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、社会的損害など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、機器またはシステムとしての出荷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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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。

アプリケーション・ノート

78K シリーズ開発ツール

チュートリアル・ガイド

対象ツール

SP78K0　Ver.2.00
SP78K0S　Ver.2.00
SP78K4　Ver.2.00

資料番号　U17047JJ1V0AN00（第 1 版）
発行年月　January 2004 NS　CP(K)

［メ　モ］

目次要約

- 本資料に記載されている内容は2004年1月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品の使用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等に関し，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，半導体製品の動作例，応用例を説明するものです。お客様の機器の設計において，回路，ソフトウエアおよびこれらに関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，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「特定水準」に分類しております。また，各品質水準は，以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認ください。

  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

  特別水準：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，生命維持を目的として設計されていない医療機器

  特定水準：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

  当社製品のデータ・シート，データ・ブック等の資料で特に品質水準の表示がない場合は，標準水準製品であることを表します。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNEC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う。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E　02.11

はじめに

―― NEC エレクトロニクス製開発環境を初めてご使用になる方へ ――

NEC エレクトロニクス製開発環境の世界へようこそ。

この資料ではサンプル・プログラムを用いて 78K シリーズの開発環境の操作をわかりやすく紹介しています。

―― すでに NEC エレクトロニクス製開発環境をお使いの方へ ――

シミュレータを用いた疑似画面出力プログラム，スロット・プログラムなどのサンプル・プログラムを用意しました。

開発環境の再確認のためにどうぞご使用ください。

対 象 者　この資料は，78K シリーズ開発ツールを初めて使用されるお客様を対象としています。
なお，使用するにあたってマイクロコンピュータ，C 言語，アセンブラの一般知識と Windows®の操作方法に関する基礎知識を必要とします。

目　　的　この資料は，78K シリーズ開発ツールの基本的な操作方法をお客様に理解していただくことを目的としています。
資料を読みながら実際にツールを使用することにより，お客様の理解をより深めていただけます。
なお，この資料では78K0の例を使って説明します。78K0Sおよび78K4で異なる部分がある場合に限り違いについての説明をしています。

構　　成　この資料は，次の内容で構成しています。

第 1 章　準備編

→ この資料で使用する 78K シリーズ開発ツールの概要とサンプル・プログラムのインストール方法について解説します。

第 2 章　体験編

→ PM plus，システム・シュミレータの基本的な操作方法を，サンプル・プログラムを使用しながら体験できます。78K0 と 78K4 が対象になります。
この章に関連するマニュアルは，関連資料の No.⑤，⑦，⑧です。

第 3 章　入門編

→ システム・シミュレータの基本的なディバグ操作を，サンプル・プログラムを使用しながら体験できます。78K0，78K0S，78K4 が対象になります。
この章に関連するマニュアルは，関連資料の No. ⑤，⑦，⑧，⑨です。

第 4 章　プログラミング編

→ 78K シリーズ CPU 用の機種依存処理を C 言語で記述するための方法について，サンプル・プログラムを使用しながら説明します。78K0，78K0S，78K4 が対象になります。
この章に関連するマニュアルは，関連資料の No. ⑤，⑦，⑧，⑨です。

関連資料　**この資料を使用する際，次の関連資料もあわせてご覧ください。**

関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

開発ツールに関する資料につきましては，SP78Kxx**のインストール**CD**に**PDF**ファイルが添付されています。**

開発ツールに関する資料（ユーザーズ・マニュアル）

資　料　名		資料番号	No.
CC78K0 Ver.3.50 **以上　C コンパイラ・パッケージ**	**操作編**	U16613J	①
	言語編	U14298J	②
RA78K0 Ver.3.60 **以上　アセンブラ・パッケージ**	**操作編**	U16629J	③
	言語編	U14446J	④
PM plus Ver.5.10		U16569J	⑤
ID78K0-NS Ver.2.52　**統合ディバッガ**	**操作編**	U16488J	⑥
SM78K **シリーズ** Ver.2.52　**システム・シミュレータ**	**操作編**	U16768J	⑦
SM78K **シリーズ** Ver.2.30 **以上　システム・シミュレータ**	**外部部品ユーザ・オープン・インタフェース仕様編**	U15802J	⑧

資　料　名		資料番号	No.
CC78K0S Ver.1.50 **以上　C コンパイラ・パッケージ**	**操作編**	U16654J	①
	言語編	U14872J	②
RA78K0S Ver.1.40 **以上　アセンブラ・パッケージ**	**操作編**	U16656J	③
	言語編	U14877J	④
PM plus Ver.5.10		U16569J	⑤
ID78K0S-NS Ver.2.52　**統合ディバッガ**	**操作編**	U16584J	⑥
SM78K **シリーズ** Ver.2.52　**システム・シミュレータ**	**操作編**	U16768J	⑦
SM78K **シリーズ** Ver.2.30 **以上　システム・シミュレータ**	**外部部品ユーザ・オープン・インタフェース仕様編**	U15802J	⑧

資　料　名		資料番号	No.
CC78K4 Ver.2.40 **以上　C コンパイラ・パッケージ**	**操作編**	U16707J	①
	言語編	U15556J	②
RA78K4 Ver.1.60 **以上　アセンブラ・パッケージ**	**操作編**	U16708J	③
	言語編	U15255J	④
PM plus Ver.5.10		U16569J	⑤
ID78K4-NS Ver.2.52　**統合ディバッガ**	**操作編**	U16632J	⑥
SM78K **シリーズ** Ver.2.52　**システム・シミュレータ**	**操作編**	U16768J	⑦
SM78K **シリーズ** Ver.2.30 **以上　システム・シミュレータ**	**外部部品ユーザ・オープン・インタフェース仕様編**	U15802J	⑧

デバイスに関する資料（ユーザーズ・マニュアル）

資　料　名	資料番号	No.
μPD780024A, 780034A, 780024AY, 780034AY **サブシリーズ**	U14046J	⑨

資　料　名	資料番号	No.
μPD789046 **サブシリーズ**	U13600J	⑨

資　料　名	資料番号	No.
μPD784038, 784038Y **サブシリーズ　ハードウエア編**	U11316J	⑨

本サンプル・プログラムおよびプログラム動作環境は 2004 **年** 1 **月現在の資料にもとづいて作成したもので，**
今後，予告なしに変更することがあります。
製品のご採用検討に当たっては事前に販売状況等を当社販売員にご確認の上，最新ドキュメントをご参照願います。

目　　次

第１章　準備編

この章では，この資料で使用する開発ツールの概要とサンプル・プログラムについて説明します。

なお，この資料では SP78Kxx に含まれる開発ツールのみで，サンプル・プログラムが動作できるようになっています。

使用するツール

ここでは，この資料で使用する開発ツールの概要について説明します。
開発ツールの名称とその主な機能は，次のとおりです。

● デバイス・ファイル

デバイス固有の情報は，デバイス・ファイルに入っているため，ツールを使用するには，デバイス・ファイルが必要となります。
この資料で使用するサンプルは，78K0 は DF780034，78K0S は DF789046，78K4 は DF784035 を使用してます。

● CC78Kxx 78K シリーズ C コンパイラ

78K シリーズ用の組み込み制御用プログラムを C 言語で記述するために開発された，汎用性，移植性の高い C コンパイラです。

● RA78Kxx 78K シリーズ アセンブラ・パッケージ

アセンブラ・ソース・プログラムから 78K シリーズで実行することができる実行コードを生成します。

● PM plus

Windows 上での統合開発環境です。
エディタ，コンパイラ，ディバッガなどの開発ツールを連携して効率的な開発が行えます。

● SM78Kxx 78K シリーズ システム・シミュレータ

ホスト PC 上で，78K シリーズ用に作られた実行コードをシミュレート実行します。

この資料のサンプル・プログラムを動作させる場合は，上記開発ツールをインストールしていただく必要があります。
なお，開発ツールのインストール方法については，SP78Kxx に添付されている“78K シリーズ ソフトウエア・パッケージ SP78Kxx 使用上の留意点”をご覧ください。
この資料では，スタート・メニューへの登録は，ディフォールトの“NEC Tools32”という名称で行ったと仮定して解説します。

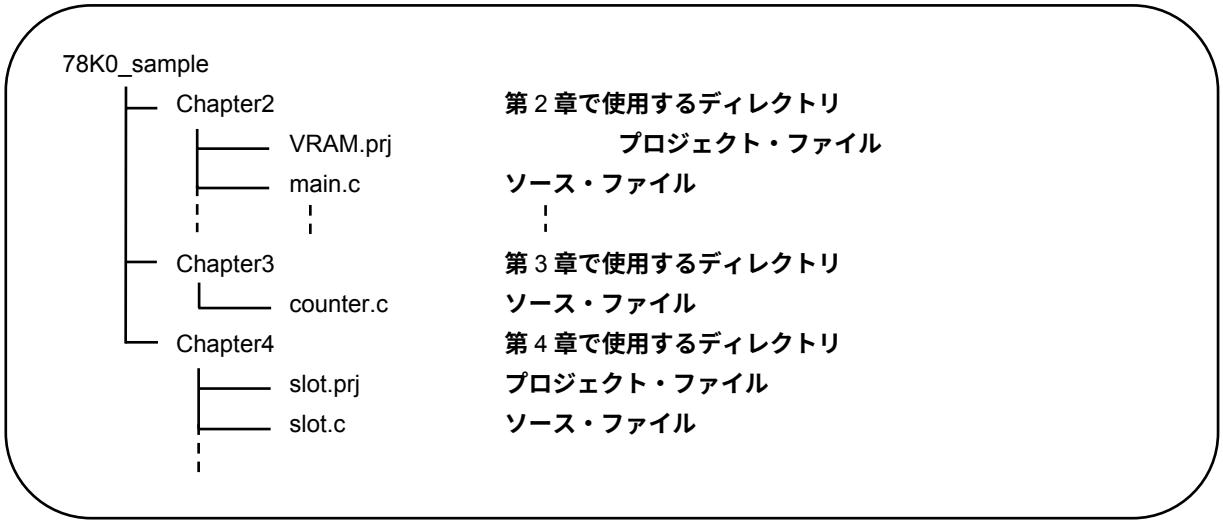
サンプル環境

ここでは，この資料で使用するサンプル・プログラムの構造について説明します。

● サンプル・プログラム本体のディレクトリ構造

サンプル・プログラム本体をインストールすると，お客様が指定したディレクトリに次のファイルが置かれます。なお，Chapter2，Chapter3，Chapter4 ディレクトリに置かれるファイルについては各章で説明します。

次に 78K0 の場合の例を示します。



78K0S の場合，Chapter2 はありません。

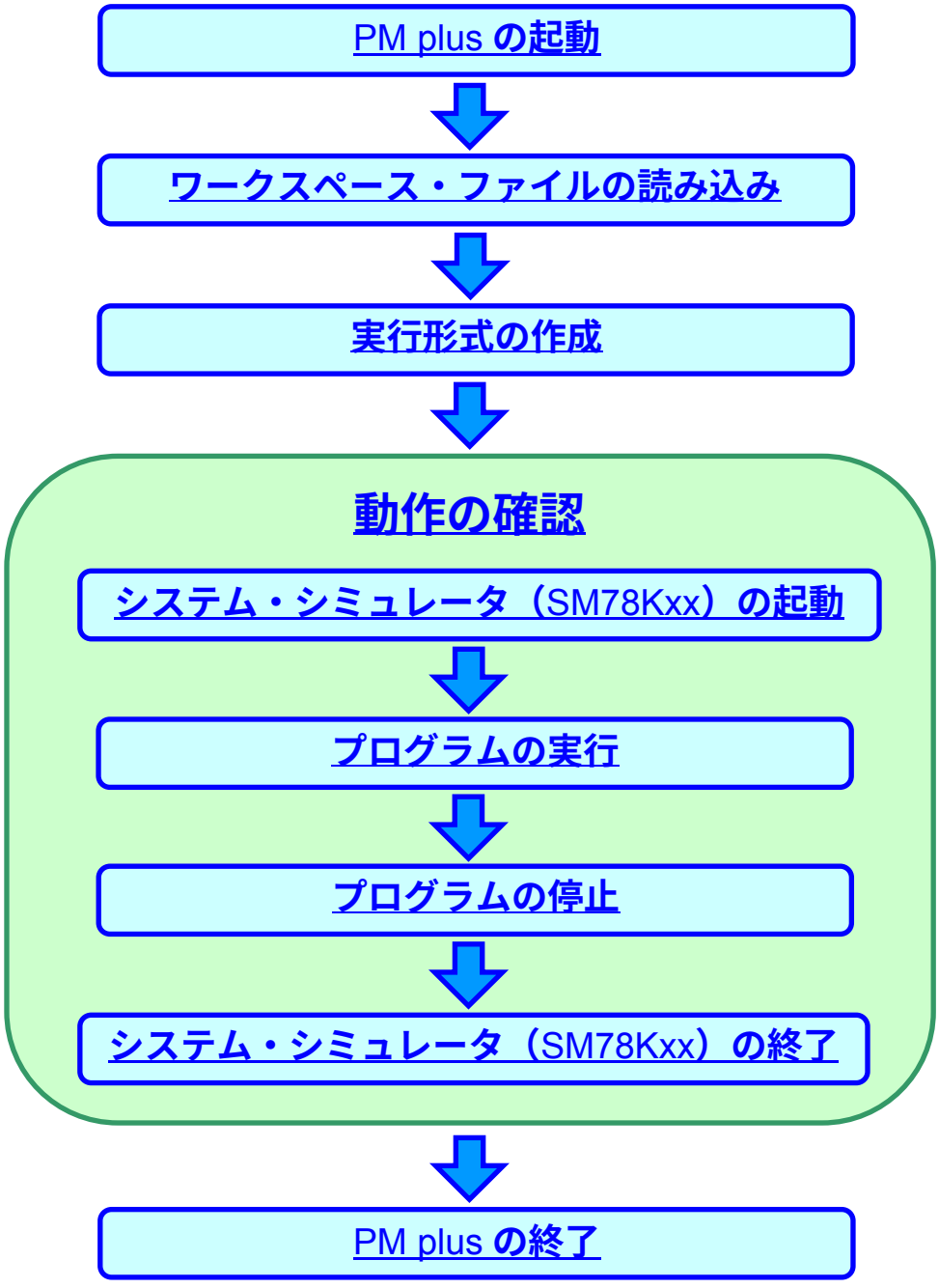
第 2 章　体験編

この章では，完成された 78K シリーズ用プログラムをシステム・シミュレータ（SM78Kxx [注]）で操作することを体験します。（注　以降では 78K シリーズの K0，K0S，K4 をまとめて表現するときに Kxx と表します。）

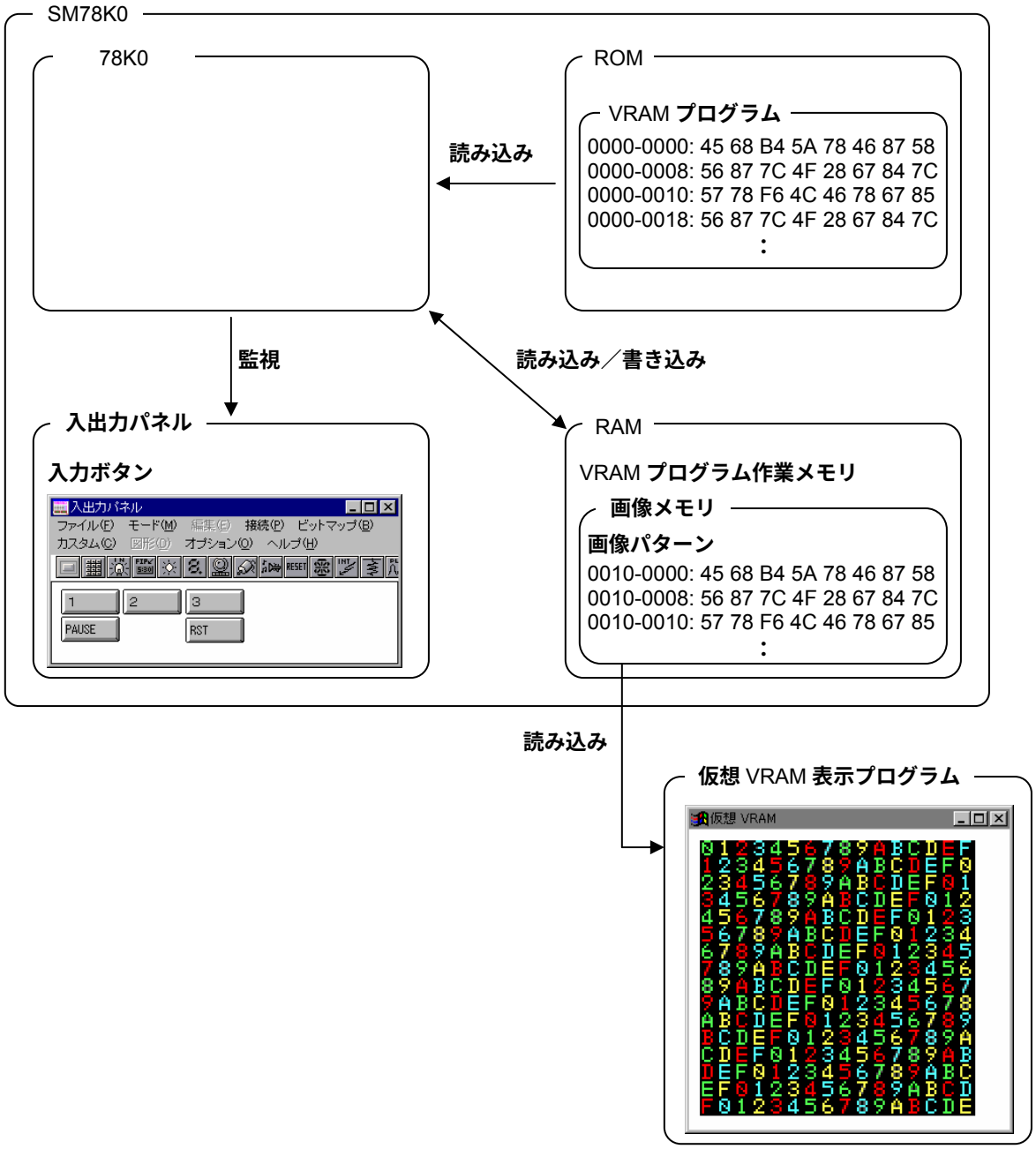
この章の例では外部 RAM を使用しますので 78K0S は対応していません。78K0 または 78K4 をご利用ください。ここでは，78K シリーズ用プログラムとして，画像メモリ（Visual RAM）にパターン画像を書き込むプログラム（以下，VRAM プログラム）を使用します。

VRAM プログラムをビルドし，SM78Kxx で操作することを通して，ツール（PM plus，SM）の基本的な操作方法と，アプリケーション・プログラムの作成時に必要なプロジェクト・ファイルの概念を理解できます。

全体の流れを，次に示します。



この章では，次の環境で VRAM プログラムを動作させます。



SM78Kxx ：78Kxx と，78Kxx に接続される ROM，RAM および入力ボタンをシミュレートします。

仮想 VRAM 表示プログラム

：SM78Kxx 用のユーザ・カスタマイズ外部部品です。

SM78Kxx で画像 RAM の内容をディスプレイ上に疑似的に表示します。

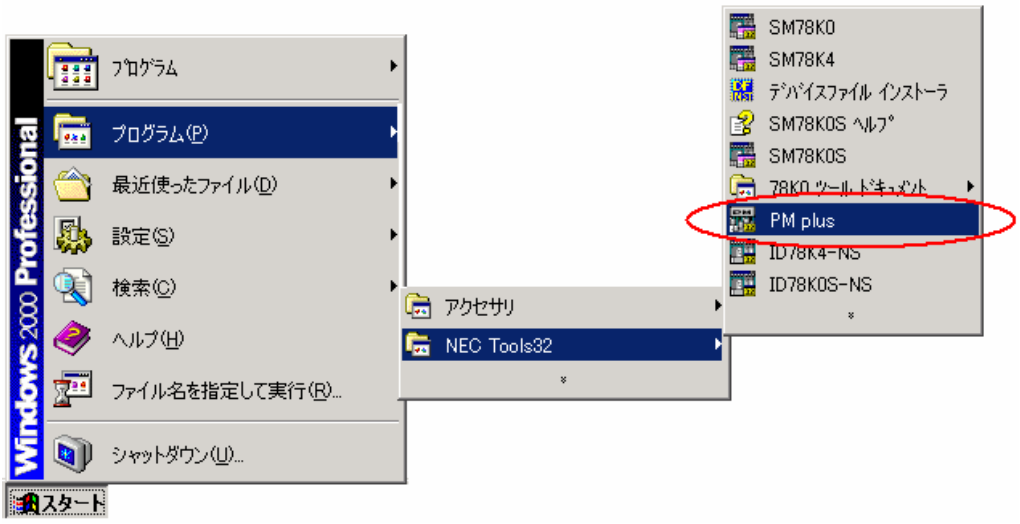
（このユーザ・カスタマイズ外部部品は，外部部品ユーザ・オープン・インタフェース機能を使用してこの章のために作成しています。外部部品ユーザ・オープン・インタフェース機能の詳細については，SM78K シリーズ システム・シミュレータ Ver.2.30 以上 ユーザーズ・マニュアル 外部部品ユーザ・オープン・インタフェース仕様編（U15802J）を参照してください。）

PM plus の起動

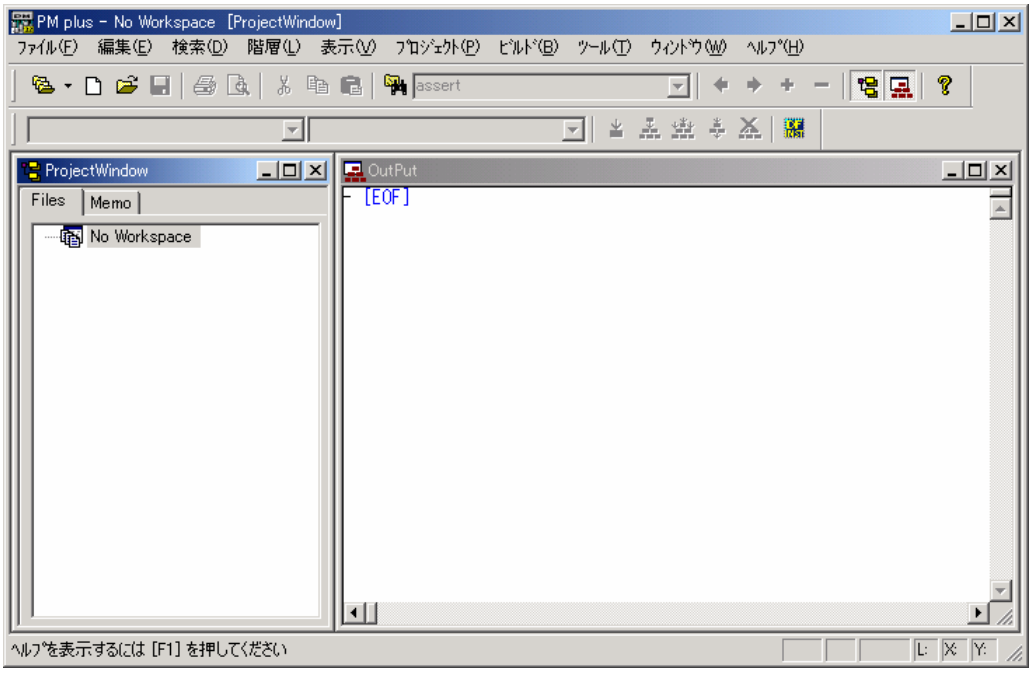
それでは，実際に各ツールを使用してみましょう。

まず，PM plus **を起動します。**

Windows **スタート・メニューの［プログラム**(P)**］→［**NEC Tools32**］→［**PM plus**］を選択してください。**

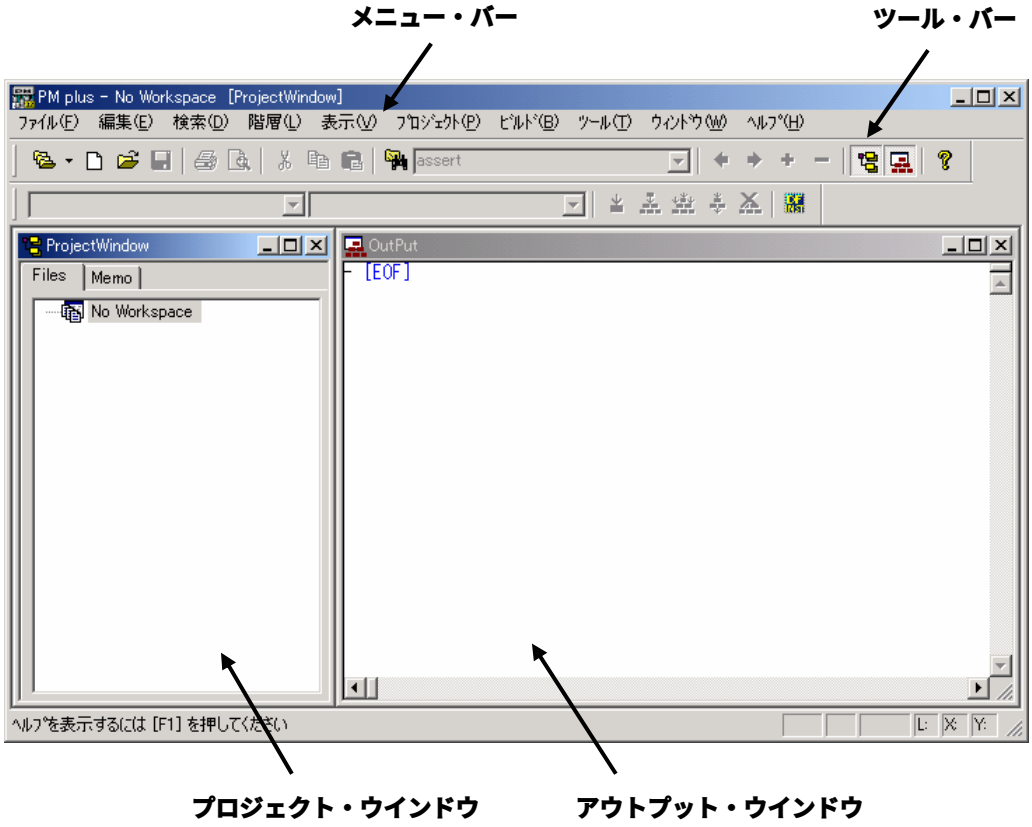


PM plus **が**
起動します。



PM plus の紹介

PM plus では，アプリケーション・プログラムや環境設定を一つのプロジェクトとして扱い，エディタでのプログラム作成，ソース管理，ビルド，ディバグといった一連の作業を管理します。



プロジェクト・ウインドウ ：プロジェクト名やそのソース・ファイル，インクルード・ファイルがツリー構造で表示されるウインドウです。

アウトプット・ウインドウ ：ビルドの実行過程が表示されるウインドウです。

メニュー・バー，およびツール・バーの詳細については，PM plus Ver.5.10 ユーザーズ・マニュアル（U16569J）を参照してください。

ワークスペースとは？

複数のプロジェクト・ファイルのファイル名を管理する単位です。

ワークスペース・ファイルとは？

複数のプロジェクト・ファイルのファイル名などの情報が保存されるファイルです。

ファイル名は“△△△△.prw”です。

プロジェクトとは？

PM plus **のもとで開発されるアプリケーション・システムを指します。**

PM plus **は環境情報をまとめて“プロジェクト・ファイル”に保存し，参照します。**

プロジェクト・ファイルとは？

プロジェクトで使用するツールやソース・ファイルなど環境情報が保存されるファイルです。

ファイル名は“△△△△.prj **”です。**

プロジェクト・ファイルは，プロジェクトの新規作成時に設定するプロジェクト・ディレクトリに作成されます。

ワークスペース・ファイルの読み込み

PM plus は，アプリケーション・プログラムの環境（使用するディレクトリやツール，オプション情報など）をプロジェクト・ファイルに保存します。

また，プロジェクト・ファイルの情報をワークスペース・ファイルに保存します。

この章では，あらかじめ作成されているワークスペース・ファイルとプロジェクト・ファイルを使用します。

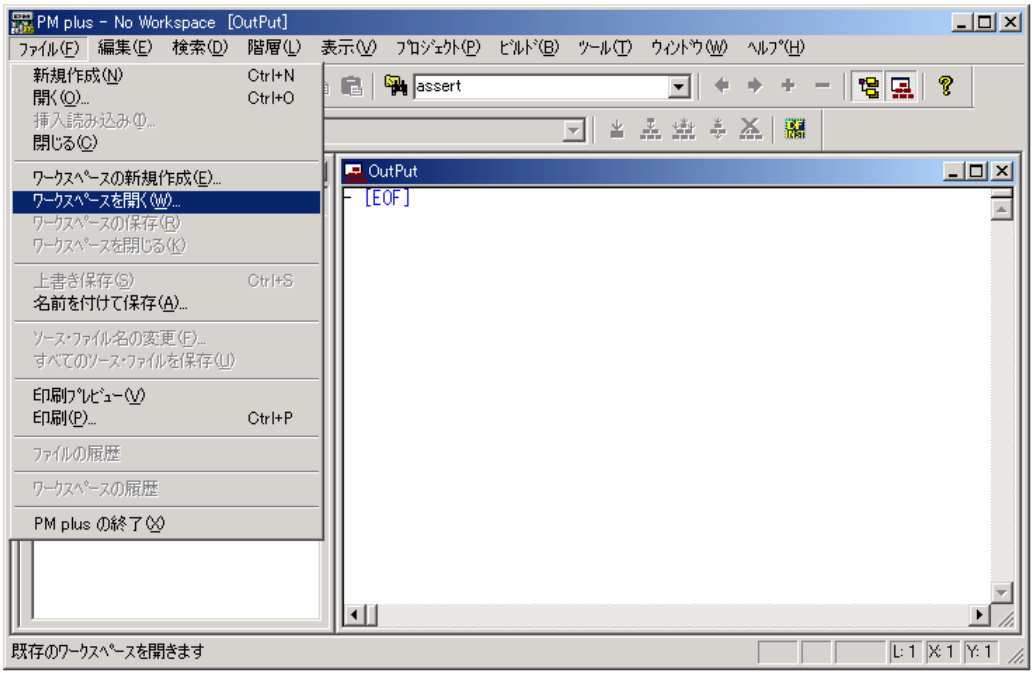
ワークスペース・ファイルとプロジェクト・ファイルの作成方法は，「第 3 章 入門編」で説明します。

この章で使用するプロジェクト・ファイルには，完成した VRAM プログラムのソース・ファイル名と SM78Kxx でシミュレートする 78Kxx，ROM，RAM および入力ボタンの設定が保存されています。

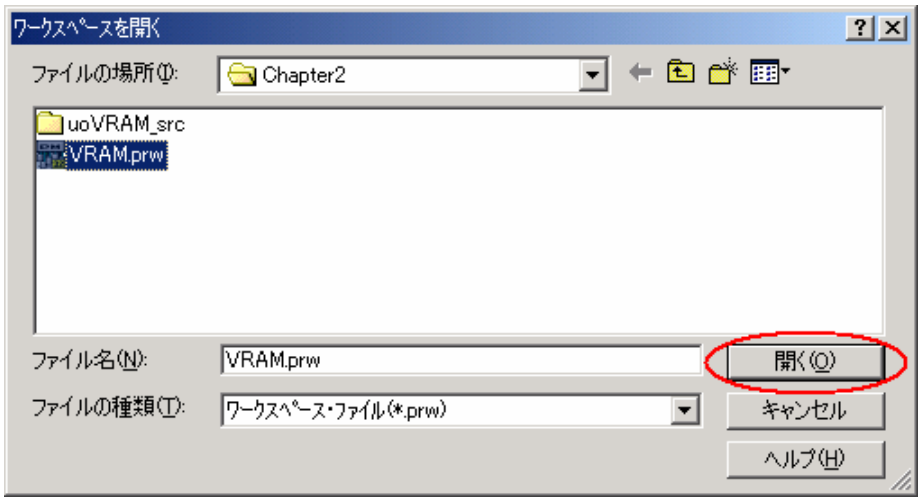
仮想 VRAM 表示プログラムは，体験編の一連の作業の中で，プロジェクト・ファイルに追加登録します。

PM plus のメニューの［ファイル(F)］→［ワークスペースを開く(W)...］を選択し，“ VRAM.prw ”を指定してください。

環境を設定していない方は「サンプル環境」をご覧ください。

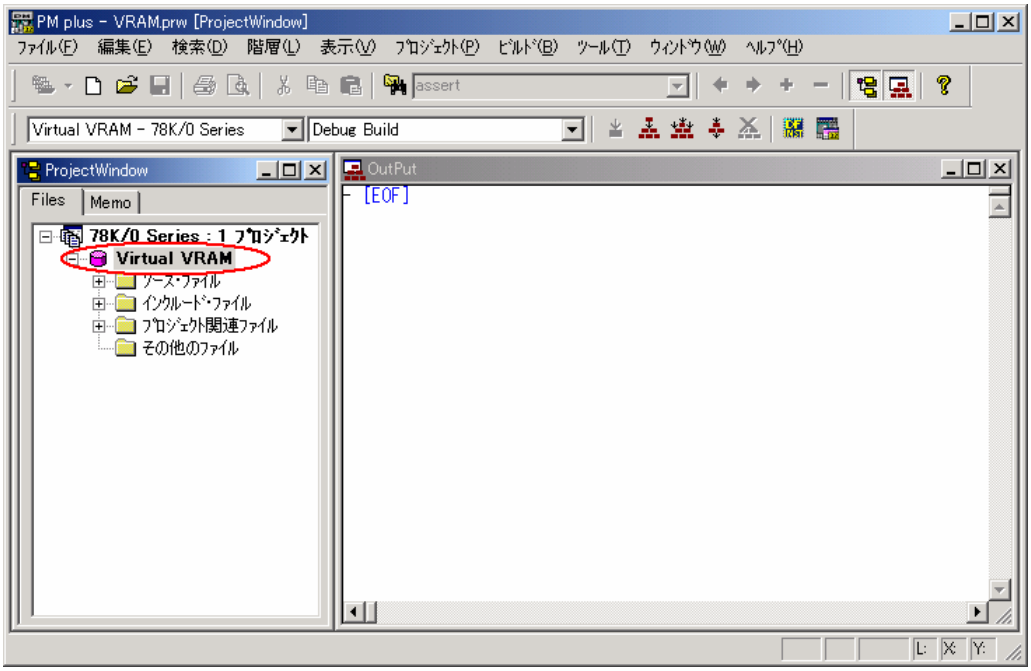


Chapter2 ディレクトリを
開いてください。



“ VRAM.prw ”を指定して 開く(O) ボタンを押してください。

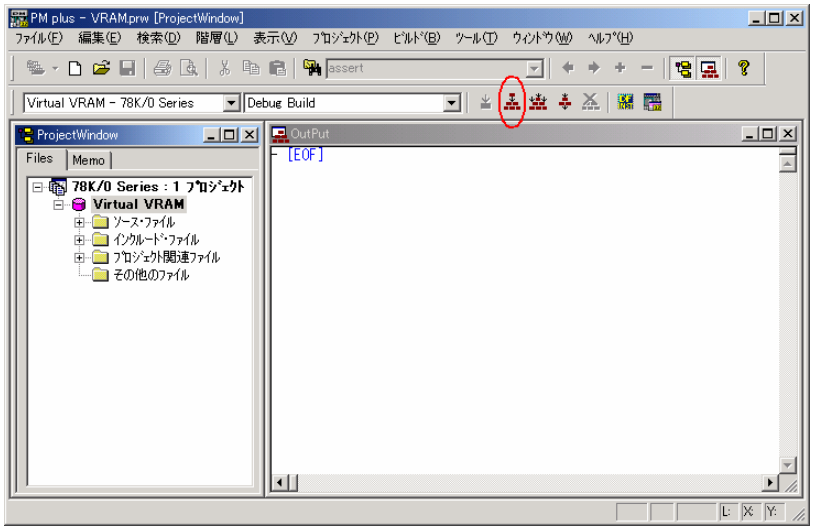
プロジェクト・ファイル
“ VRAM.prw ”を読み込みます。



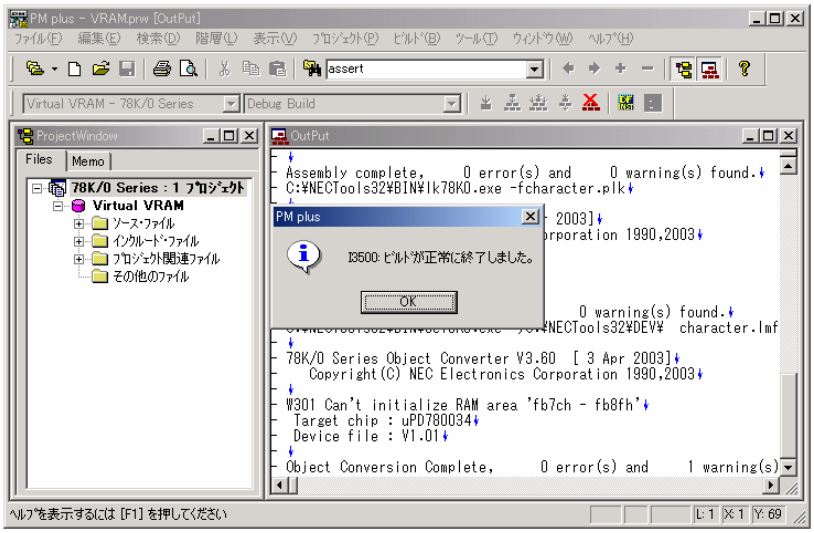
実行形式の作成

プロジェクトの実行形式を作成します。この作業をビルドと呼びます。

PM plus のビルド・ボタン  ，またはメニューの［ビルド(B)］→［ビルド(B)］を選択してください。



ビルド処理を実行します。



ビルド処理を正常に終了しました[注]。

注 ここで，アウトプット・ウインドウに“Can't initialize RAM area”というメッセージが表示されますが，VRAM プログラムの動作には問題ありません。
メッセージの詳細については，RA78Kxx アセンブラ・パッケージのユーザーズ・マニュアルを参照してください。

ビルドとは？

プロジェクトに登録されているソース・ファイルから実行形式ファイルなどを作成する機能です。
PM plus がコンパイル，アセンブル，リンクなどを自動的に実行します。
また，PM plus は，2 度目以降のビルドでは，前回のビルドから更新されたファイルを自動的に検出し，該当するファイルのみをコンパイル，アセンブルすることにより，ビルドにかかる時間を短縮しています。

リビルドとは？

ビルドは，前回から更新されたソース・ファイルのみをコンパイル，アセンブルしますが，リビルドではすべてのソース・ファイルをコンパイル，アセンブルします。
コンパイラ・オプション等，各種設定を変更したときは，ビルドではなくリビルドを選択する必要があります。
また，ビルドで更新されたファイルの検出に失敗する可能性がある次の場合にも，リビルドを選択する必要があります。

- 修正したソースを修正前の古いソース・ファイルにファイルコピーなどで置き換えたとき
- 前回のビルドの後，ホスト・マシンの時刻を訂正したとき
- 時刻がずれているホスト・マシン間でプロジェクト環境を移動したとき

動作の確認

NEC エレクトロニクスではユーザ・アプリケーションの実行環境として，統合ディバッガ，およびシステム・シミュレータを提供しています。

この資料では，システム・シミュレータ（SM78Kxx）を起動し，動作を確認します。

統合ディバッガ（ID78Kxx）とは？

統合ディバッガは，インサーキット・エミュレータとターゲット・システムを接続した開発環境でディバグを行う Windows ベースのソフトウエア・ツールです。
C ソース・レベル，またはアセンブラ・レベルでのディバグが可能です。
インサーキット・エミュレータの持つイベント設定機能を利用して，リアルタイムに実行，検証できます。

システム・シミュレータ（SM78Kxx）とは？

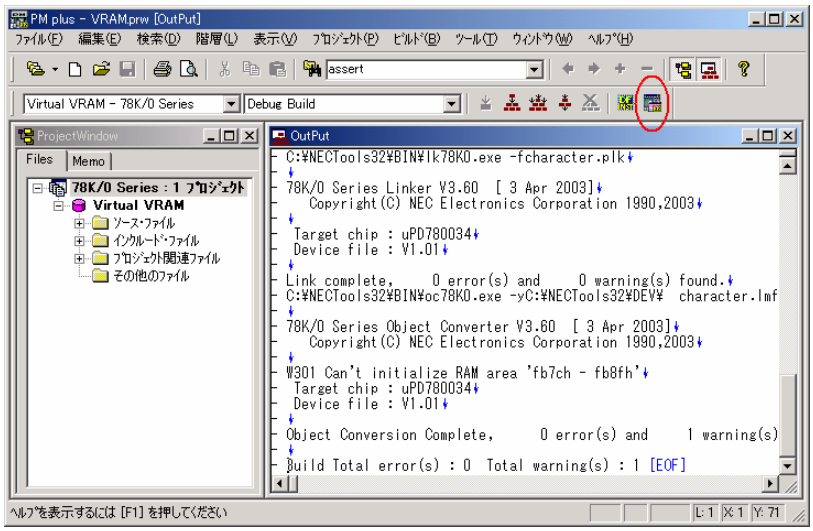
システム・シミュレータは，ホスト・マシン上でターゲット・システムの動作をシミュレーションしながら， ディバグを行う Windows ベースのソフトウエア・ツールです。
C ソース・レベル，またはアセンブラ・レベルでのディバグが可能です。
アプリケーションの論理検証をハードウエア開発から独立して行えます。

システム・シミュレータ（SM78Kxx）の起動

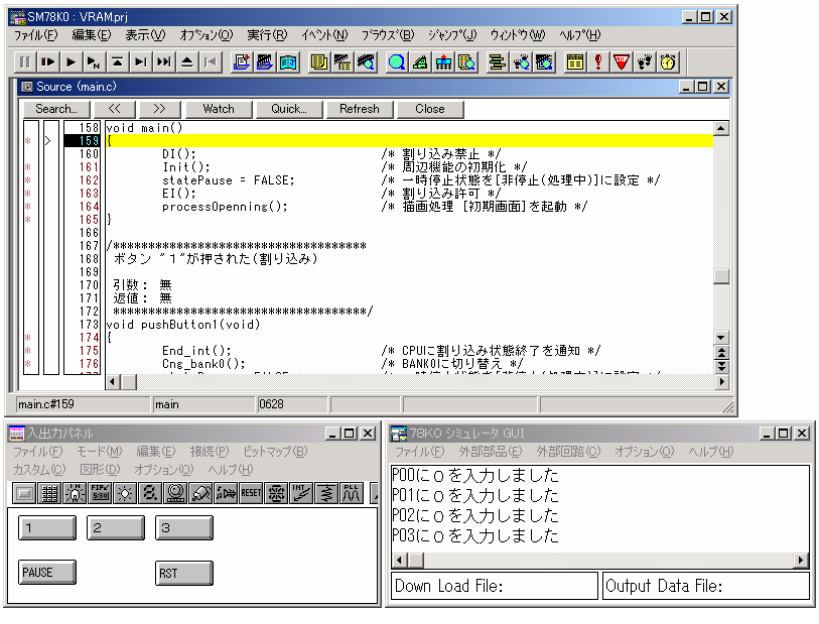
SM78Kxx を起動します。

PM plus のディバグ・ボタン  ，またはメニューの［ビルド(B)］→［ディバグ(D)］を選択してください。

ここで，ディバグ・ボタンが表示されていない場合は，メニューの［ツール(T)］→［ディバッガの選択(D)...］で“ SM78Kxx システム・シミュレータ ”を選択してください。
詳細については，「第3章 入門編」をご覧ください。

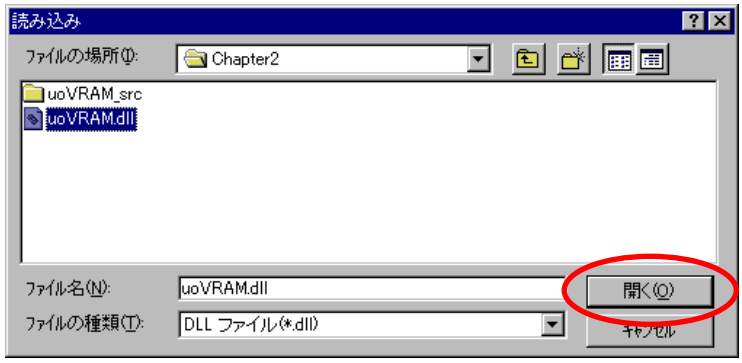


SM78Kxx が起動します。



次に，入出力パネル・ウインドウのメニューの［カスタム(C)］→［ロード(L)］を選択し，“ uoVRAM.dll ”を指定してください。

“ uoVRAM.dll ”を指定し，開く(O) ボタンを押します。



ここで，“ uoVRAM.dll ”ファイルが表示されない場合は，“ ファイル名(N): ”の欄に“ uoVRAM.dll ”と直接入力するか，または，エクスプローラで DLL ファイルを表示できるように設定してください。

仮想 VRAM ウインドウが開きます。

ここで扱う“ uoVRAM.dll ”は，VRAM プログラム用に作成されたものですので，使用するにあたって，設定変更などを行う必要はありません。

“ uoVRAM.dll ”のソースからの作成方法については，「付録 uoVRAM.dll の作成方法」をご覧ください。

また，詳細については，SM78K シリーズ システム・シミュレータ Ver.2.30 以上 ユーザーズ・マニュアル 外部部品ユーザ・オープン・インタフェース仕様編（U15802J）を参照してください。

システム・シミュレータ（SM78Kxx）の紹介

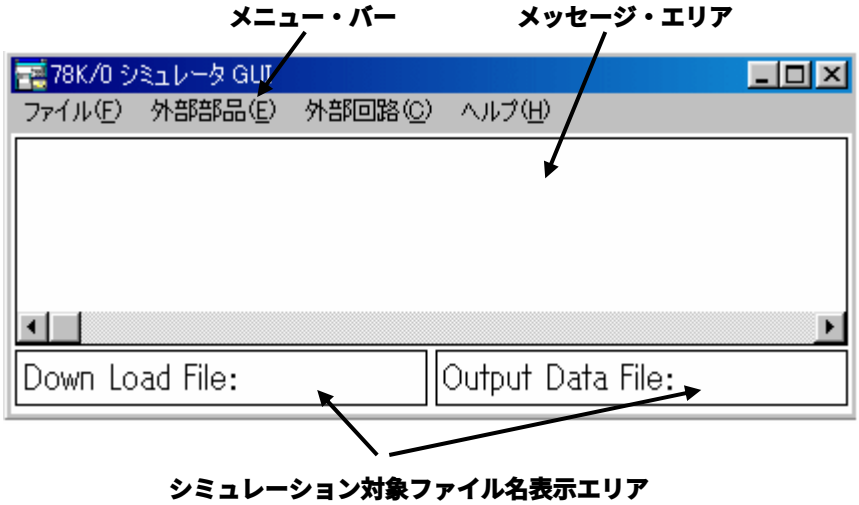
SM78Kxx は，メイン・ウインドウとシミュレータ GUI ウインドウから構成されています。

メイン・ウインドウ　　：CPU コア内部のステータスの表示，およびシミュレータ実行の制御を行います。
シミュレータ GUI ウインドウ　：外部部品の制御を行います。

SM78Kxx の初期画面は次のとおりです。

＜メイン・ウインドウ＞

＜シミュレータ GUI ウインドウ＞



各エリア，メニュー・バー，およびツール・バーの詳細については，SM78K シリーズ システム・シミュレータ Ver.2.52 ユーザーズ・マニュアル 操作編（U16768J）を参照してください。

入出力パネル・ウインドウの紹介

SM78Kxx は，疑似的なターゲット・システムを構築するために，ボタンや LED などの標準的な部品を提供しています。

入出力パネル・ウインドウは，それらの部品を設定，および使用するためのウインドウです。

78Kxx シミュレータ GUI ウインドウのメニューの［外部部品(E)］→［入出力パネル(P)...］を選択し，開くことができます。

- 入出力パネルの設定方法は，「第 3 章 入門編」で説明します。
- メニュー・バー，およびツール・バーの詳細については，SM78K シリーズ システム・シミュレータ Ver.2.52 ユーザーズ・マニュアル 操作編（U16768J）を参照してください。

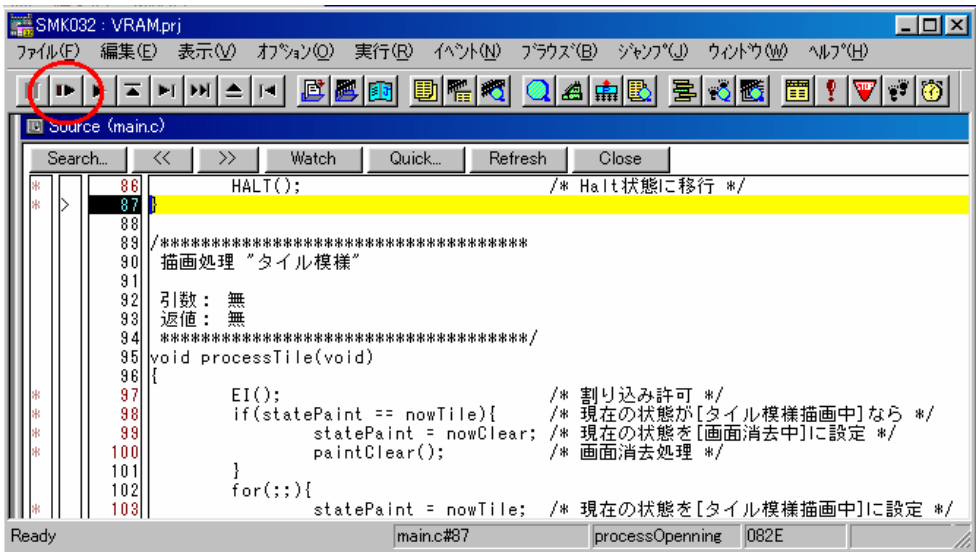
ここで用意されていない部品は，“ uoVRAM.dll ”のように「Microsoft™ Visual C++™」を使用して，自由に作成することができます。

- ユーザ作成の外部部品の詳細については，SM78K シリーズ システム・シミュレータ Ver.2.30 以上 ユーザーズ・マニュアル 外部部品ユーザ・オープン・インタフェース仕様編（U15802J）を参照してください。

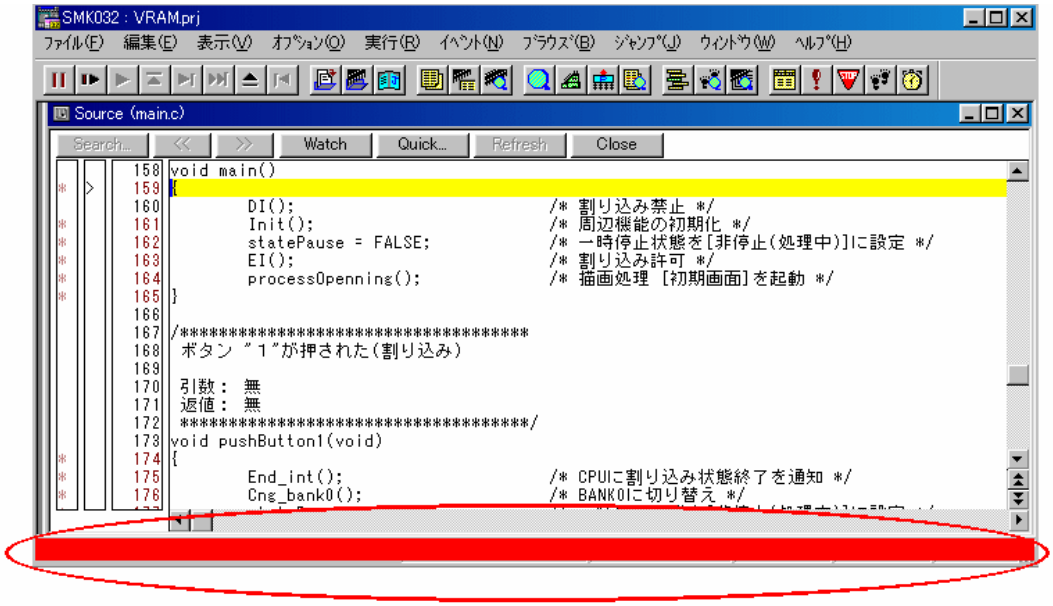
プログラムの実行

次に，プログラムを実行します。

SM78Kxx のリスタート・ボタン  ，またはメニューの［実行(R)］→［リスタート(R)］を選択してください。VRAM プログラムを実行します。



プログラムを実行します。

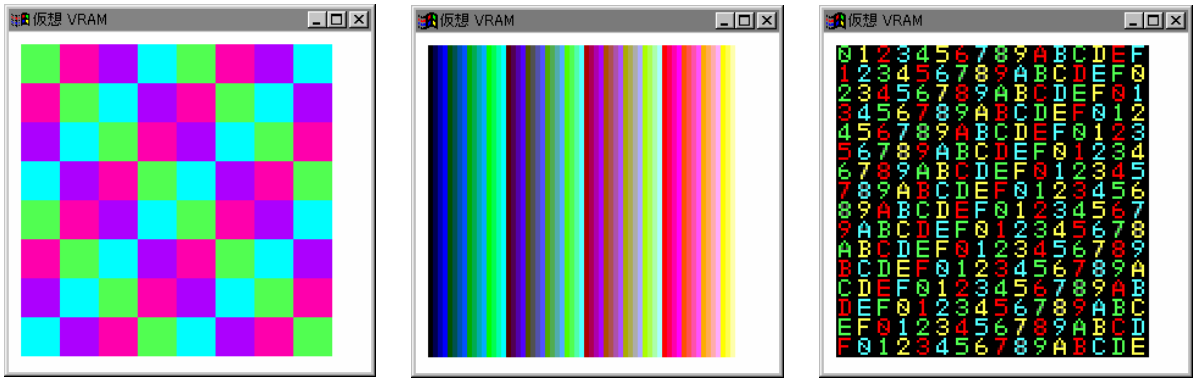


プログラムの実行中はステータス表示エリアが赤く変化します。

次に，実行中の VRAM プログラムを操作します。
入出力パネルの各ボタンを操作して，画面の変化を確認してください。

1 ボタンを押すと＜画面 ①＞を描画するプログラムに切り替わります。
2 ボタンを押すと＜画面 ②＞を描画するプログラムに切り替わります。
3 ボタンを押すと＜画面 ③＞を描画するプログラムに切り替わります。
RST ボタンを押すと，シミュレーションしているターゲット CPU をリセットします。
画面描画中に PAUSE ボタンを押すと，画面描画を一時停止します。
この描画停止中にもう一度 PAUSE ボタンを押すと描画を再開します。

＜画面 ①＞ ＜画面 ②＞ ＜画面 ③＞

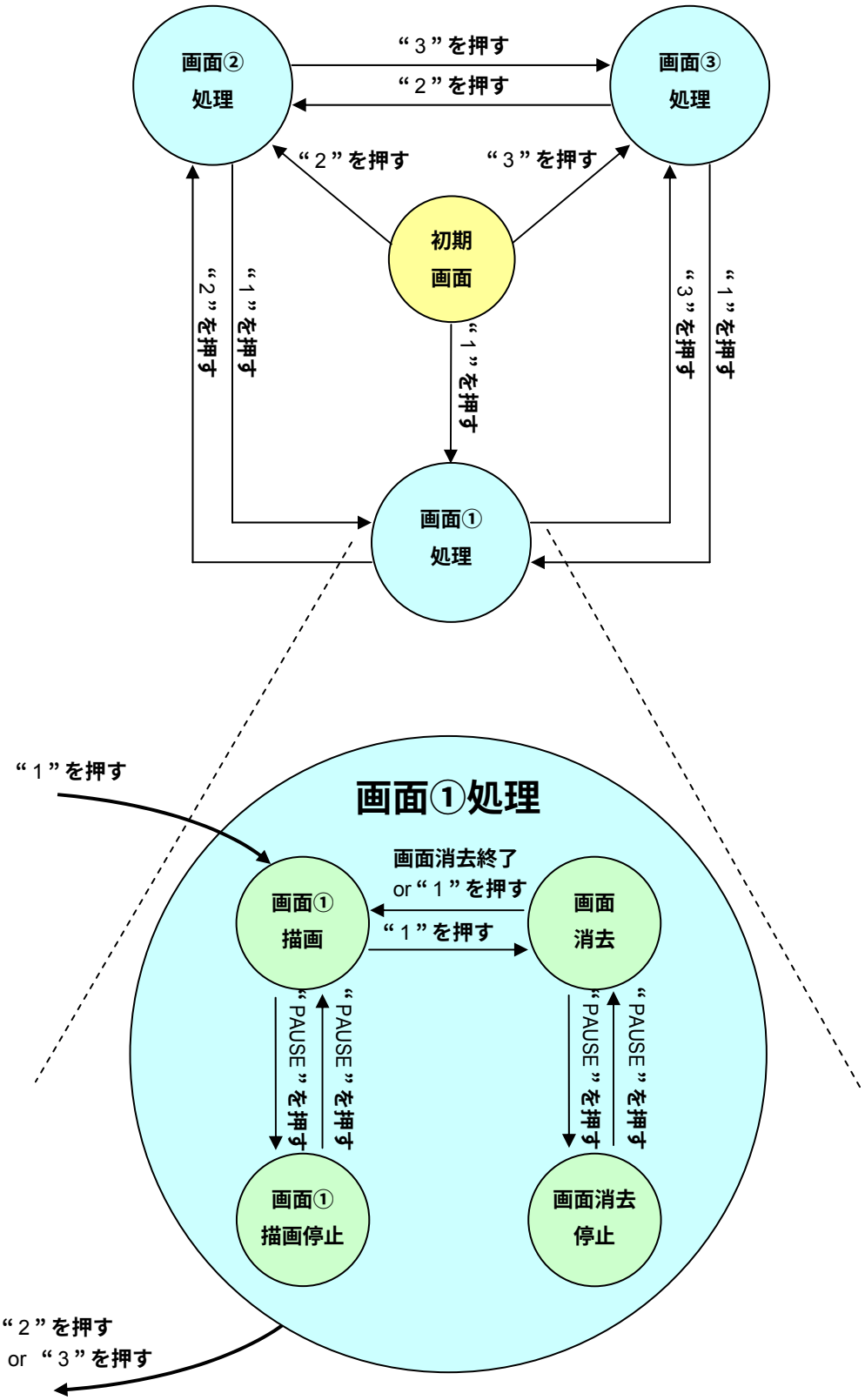


これで VRAM プログラムが正常に動作していることが確認できました。

ターゲット CPU のリセットとは？

ここでいうターゲット CPU とは，SM78K0 がシミュレートしている仮想のμPD780034 です。78K4 の時はμPD784035 になります。
ターゲット CPU のリセットを行うと，仮想μPD780034 の $\overline{\text{RESET}}$ 端子に low 信号が入力された状態を SM78K0 がシミュレートします。
これにより，仮想μPD780034 上で動作している VRAM プログラムは初期状態に戻ります。
したがって，この操作は，SM78Kxx が動作しているパソコンをリセットすることではありません。

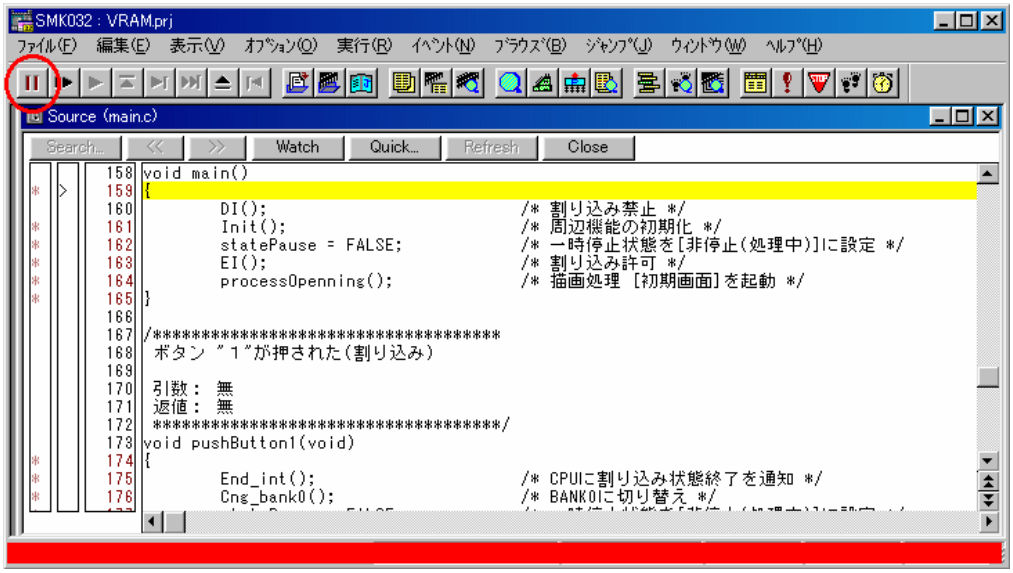
VRAM プログラムの画面処理の流れを次に示します。



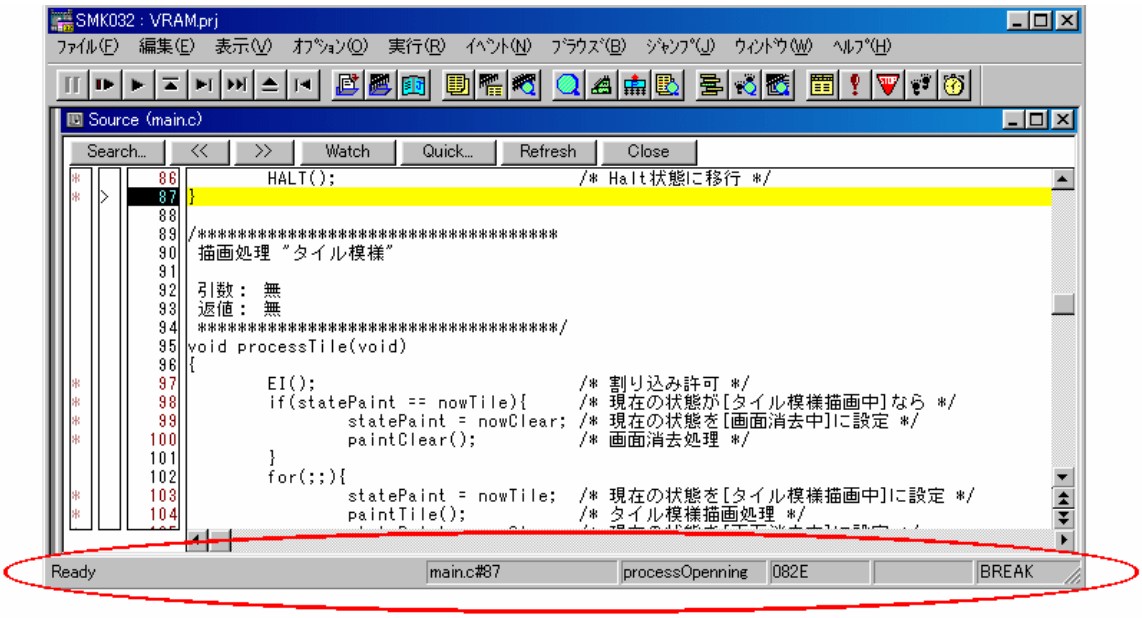
プログラムの停止

プログラムを停止します。

SM78Kxx **の停止ボタン**  **，またはメニューの［実行(R)］→［ストップ(S)］を選択してください。**



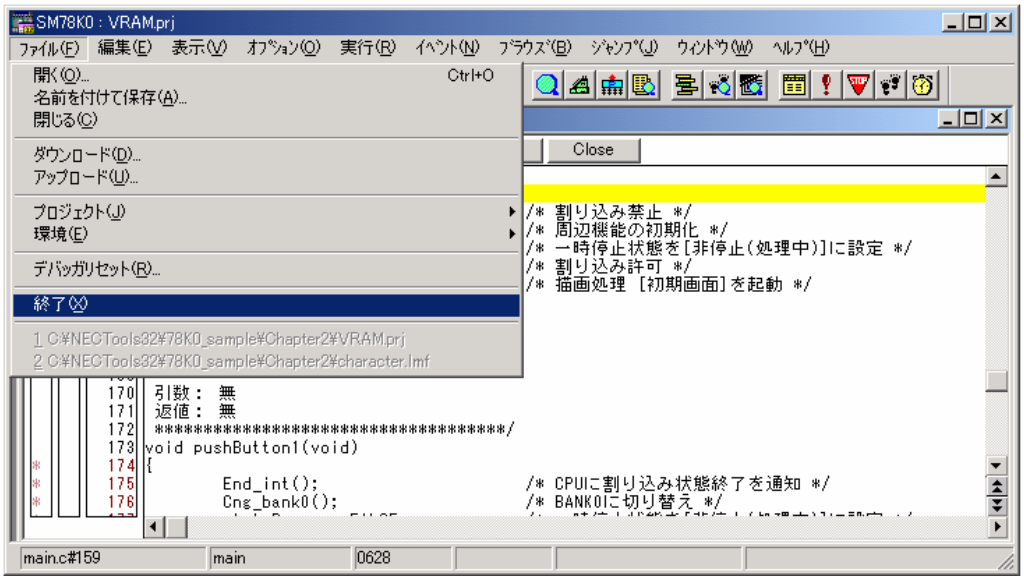
プログラムを停止します。



プログラムを停止すると，ステータス表示エリアの色が元に戻ります。

システム・シミュレータ（SM78Kxx）の終了

SM78Kxx **メニューの［ファイル(F)］→［終了(X)］を選択してください。**

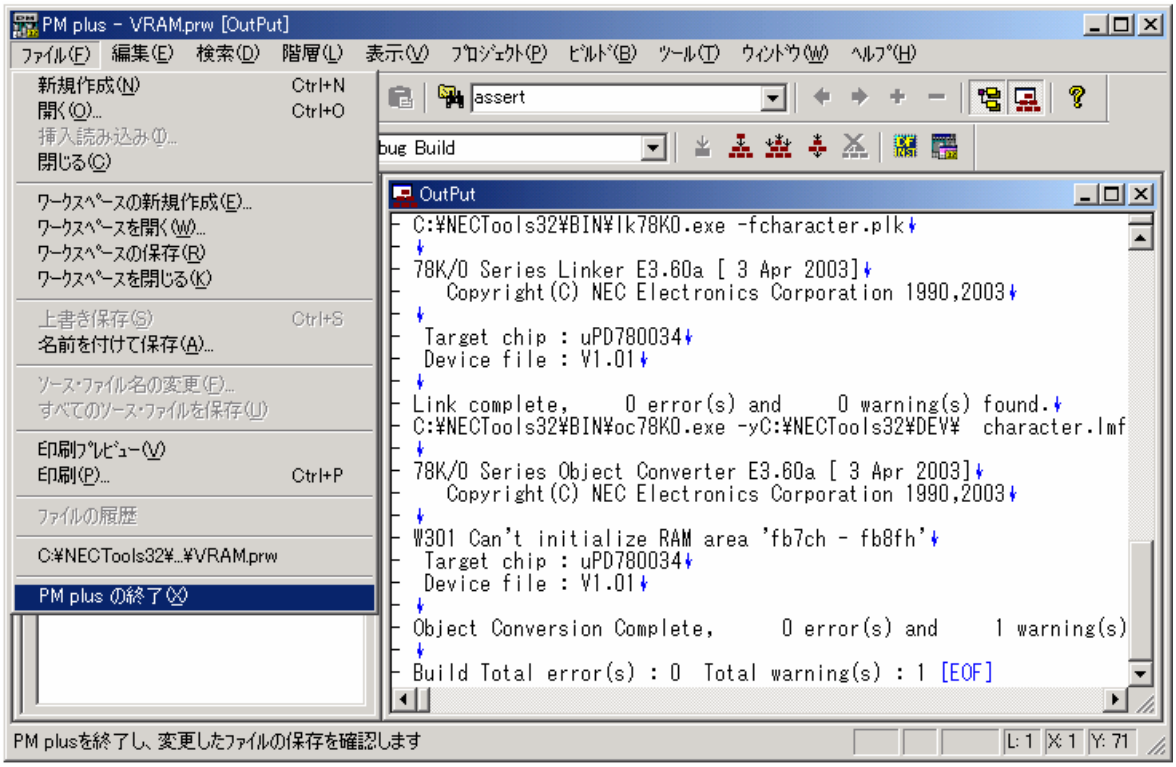


終了確認ダイアログが表示されます。

はい(Y) を押してください。SM78Kxx **が終了します。**

PM plus の終了

PM plus のメニューの［ファイル(F)］→［PM plus の終了(X)］を選択してください。



PM plus が終了します。

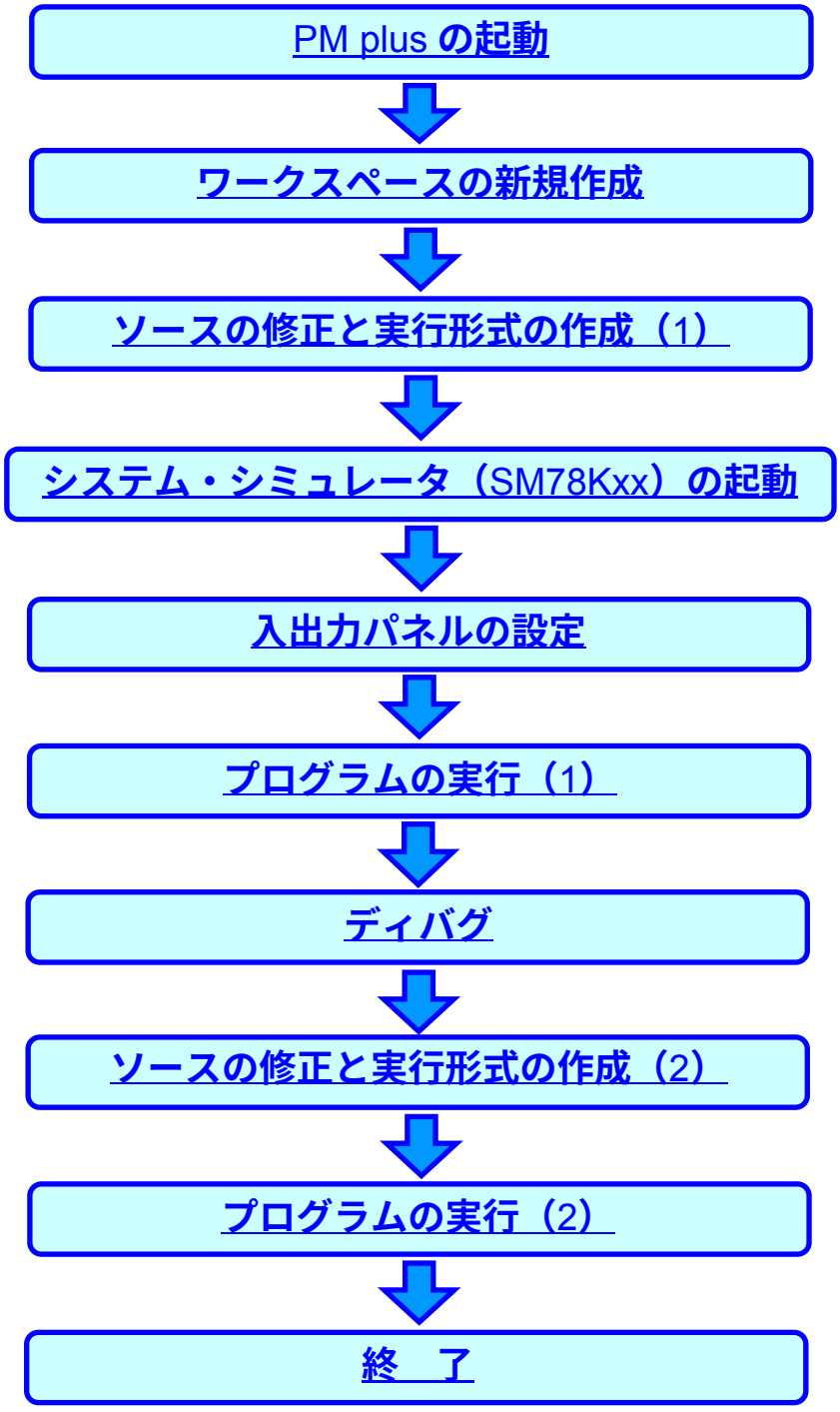
第 3 章　入門編

この章では，システム・シミュレータ（SM78Kxx）を用いた基本的なディバグ操作を，サンプル・プログラムを使用しながら解説します。

ここでは，サンプル・プログラムとしてカウンタ・プログラムを使用します。

カウンタ・プログラムには，あらかじめいくつかのバグが埋め込まれていますので，実際にこのバグを修正しながら操作を進めていきましょう。

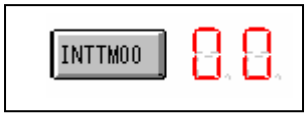
全体の流れを，次に示します。



カウンタ・プログラムの仕様

この章のディバグ操作を行う前に，カウンタ・プログラムの概要を理解していただく必要があります。
基本的な外部仕様は次のとおりです。

外部仕様
ある装置に，ボタンと 2 桁の 7 セグメント LED があり，ボタンを押すとカウンタがカウント・アップします。
78K0S の場合は，INTWT を使用します。78K4 の場合は，INTC00 を使用します。



この章では“ カウント・アップ機能 ”と“ LED 表示機能 ”を実装します。ボタン入力と初期化などの処理は，SM78Kxx のディバグ機能を利用した，ディバグ用メイン・ルーチンを使用します。

基本仕様（“ カウント・アップ機能 ”,“ LED 表示機能 ”）

- カウント・アップ機能
  - INTTM00 割り込みが発生したら，1 つずつカウント・アップする 2 桁の 10 進カウンタです。
  - 99 までカウントしたら次は 0 に戻します（ループ・カウンタにします）。
- LED 表示機能
  - 10 進カウンタの値を 7 セグメント LED に出力します。
  - 7 セグメント LED の制御には 78Kxx の入出力ポートを使用します。

ディバグ用メイン・ルーチン基本仕様

- カウンタの値を 0 で初期化します。
- 7 セグメント LED制御用の入出力ポートを初期化します。
- INTTM00 割り込みの発生は，SM78K0 で疑似的に内部割り込みを発生させる“ 内部割り込みボタン ”を利用し，疑似割り込みを受け取るのに必要な部分のみを実装します。
- カウント処理時以外は，CPU を HALT モードにします。

入出力ポートとは？
78K シリーズのほとんどのデバイスには，CPU 外部にある部品などの制御や CPU 外部から信号を取得する手段の 1 つとして，入出力ポートが装備されています。
入出力ポートの詳細については，各デバイスのユーザーズ・マニュアルを参照してください。

内部仕様は次のとおりです。

内部仕様

- カウンタの値はグローバル変数 count1, count10 に保持します。

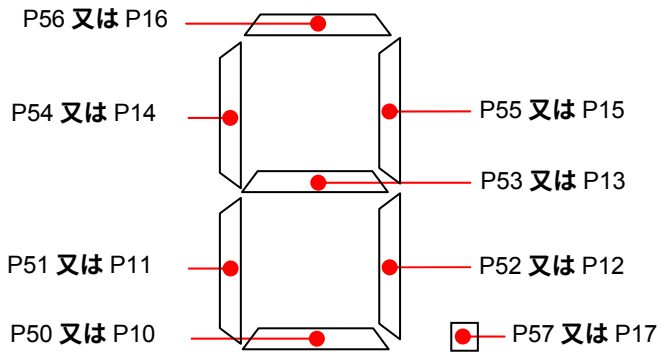
変数	内容
int count1	1 桁目（1 の位）の値を保持
int count10	2 桁目（10 の位）の値を保持

- プログラムは，LED の表示，カウント・アップ，ディバグ用メイン・ルーチンの 3 つの関数で構成します。

処理関数	処理内容
LED の表示 void putLED()	・カウンタの値を LED に表示
カウント・アップ void interrupt1()	・INTTM00 割り込みで起動 ・カウンタのカウント・アップと桁溢れを処理 ・カウント・アップ後，putLED()関数を起動
ディバグ用メイン・ルーチン void main()	・LED を制御する入出力ポートを初期化 ・INTTM00 割り込みを受け取り可能状態に設定 ・カウンタを 0 で初期化し，putLED()関数を起動し，初期値 0 を表示 ・CPU を HALT モードに移行

- 7 セグメント LED の制御には 78K0，78K4 は入出力ポート P4，P5 を，78K0S は P0,P1 を使用します。
  P4 は表示内容の設定，P5 は表示する桁を指定します。

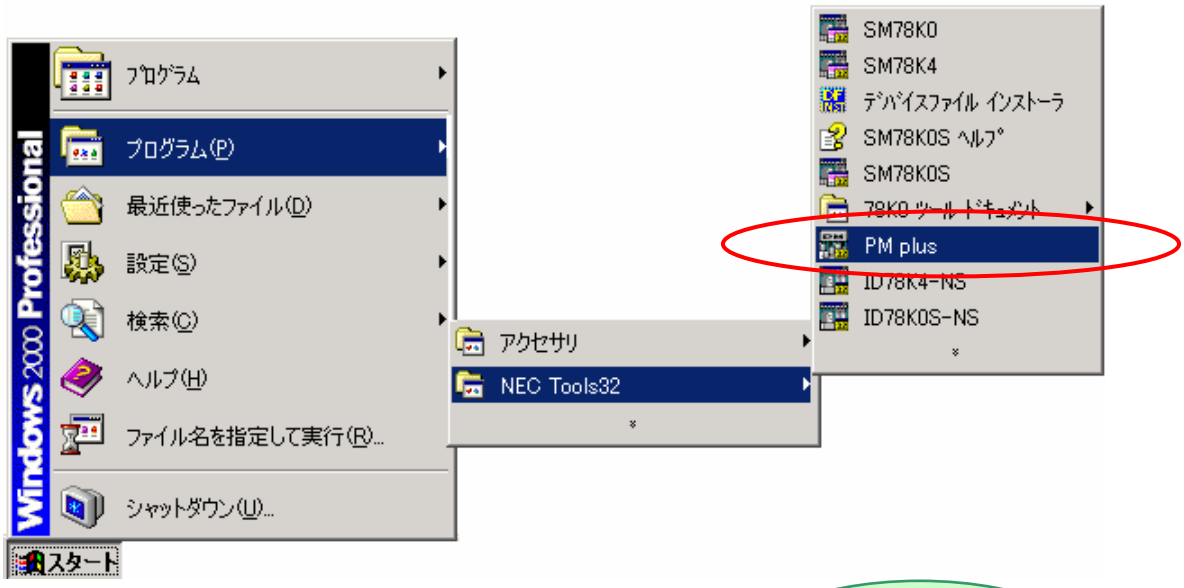
ポート	ビット位置	略号	ビットの役割
P4 又は P0	0	P40 又は P00	0→1 の時，P1 に出力している内容を 1 桁目に反映
	1	P41 又は P01	0→1 の時，P1 に出力している内容を 2 桁目に反映
P5 又は P1	0	P50 又は P10	下の横棒の点灯状態を設定　（1...点灯，0...無点灯）
	1	P51 又は P11	左下の縦棒の点灯状態を設定（1...点灯，0...無点灯）
	2	P52 又は P12	右下の縦棒の点灯状態を設定（1...点灯，0...無点灯）
	3	P53 又は P13	中の横棒の点灯状態を設定　（1...点灯，0...無点灯）
	4	P54 又は P14	左上の縦棒の点灯状態を設定（1...点灯，0...無点灯）
	5	P55 又は P15	右上の縦棒の点灯状態を設定（1...点灯，0...無点灯）
	6	P56 又は P16	上の横棒の点灯状態を設定　（1...点灯，0...無点灯）
	7	P57 又は P17	右下ドットの点灯状態を設定（1...点灯，0...無点灯）



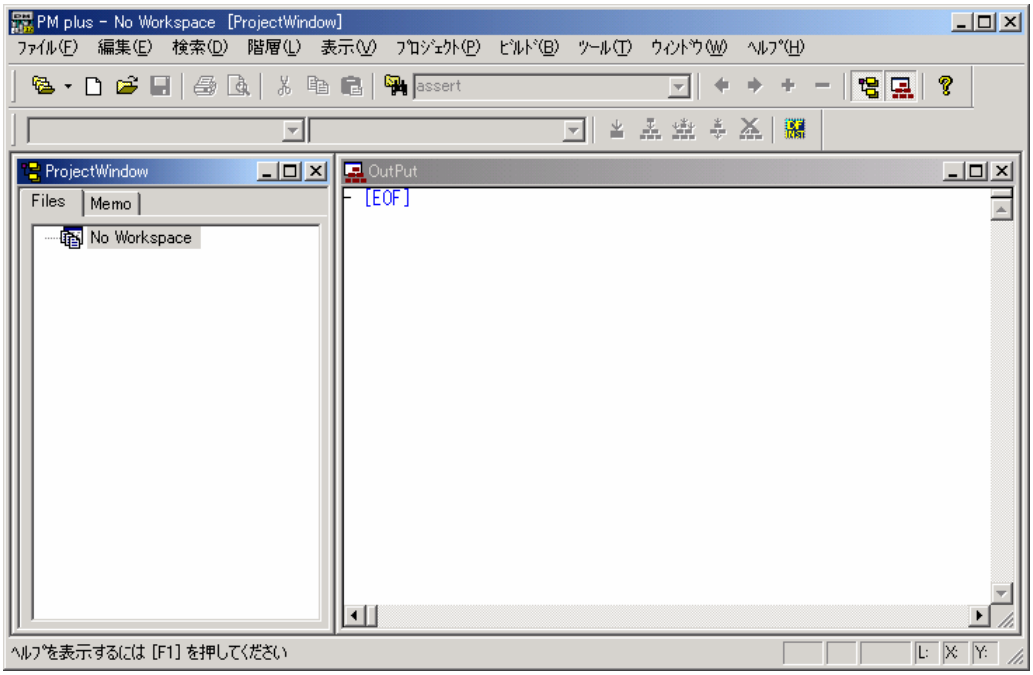
PM plus の起動

まず，PM plus を起動します。

Windows **スタート・メニューの［プログラム(P)］→［NEC Tools32］→［PM plus］を選択してください。**



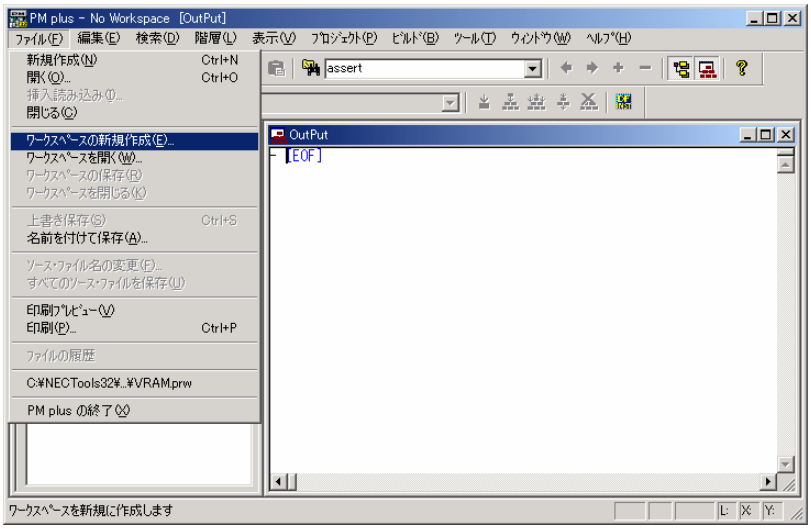
PM plus **が**
起動します。



ワークスペースの新規作成

ワークスペースを新規作成します。

PM plus のメニューの［ファイル(F)］→［ワークスペースの新規作成(E)...］を選択してください。



ワークスペースの新規作成
ダイアログが開きます。

＜各項目の説明＞

ワークスペース・ファイル名（W）

→ ワークスペース情報を保存するファイル名を指定します。

フォルダ位置（F）

→ ワークスペース・ファイル，およびプロジェクト・ファイルを保存するフォルダを指定します。

参照(R)... ボタンを押すと，参照ダイアログから選択できます。

プロジェクト・グループ名（G）

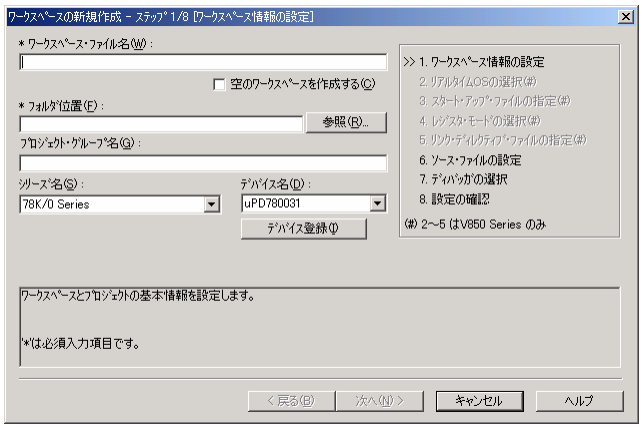
→ プロジェクト・ウインドウで表示するプロジェクト・グループ名を指定します。

シリーズ名（S）

→ 使用するデバイス・ファイルのシリーズ名を指定します。

デバイス名（D）

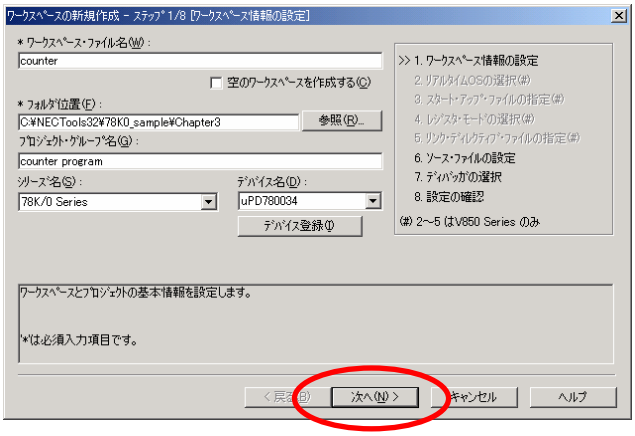
→ 使用するデバイス・ファイルのデバイス名を指定します。



次のページで，ここで設定する具体的な内容を記述しています。

ワークスペース情報を，次のように設定してください。

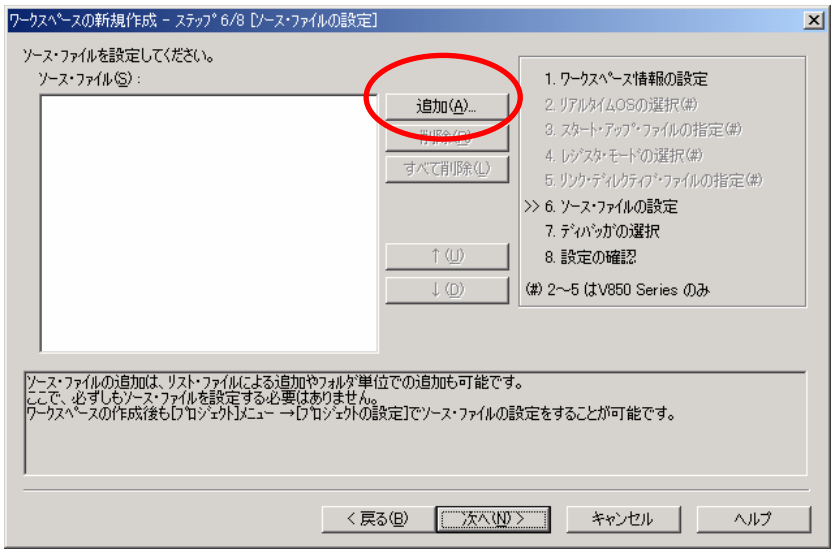
ワークスペース・ファイル名（W）
→ counter
フォルダ位置（F）
78K0 の場合，
→ ¥78K0_sample¥Chapter3
（ 参照(R)... ボタンでサンプルを展開したディレクトリを選択してください）
78K0S の場合，
→ ¥78K0S_sample¥Chapter3
78K4 の場合，
→ ¥78K4_sample¥Chapter3
プロジェクト・グループ名（G）
→ counter program
シリーズ名（S）
→ 78K/0 Series　(78K0 の場合)
→ 78K/0S Series (78K0S の場合)
→ 78K/4 Series　(78K4 の場合)
デバイス名（D）
→μPD780034　(78K0 の場合)
→μPD789046　(78K0S の場合)
→μPD784035　(78K4 の場合)



設定が完了したら，次へ(N)> ボタンを押してください。

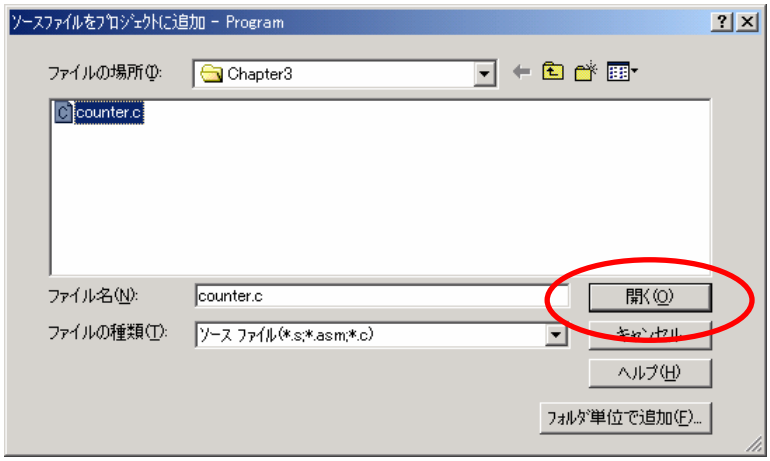
プロジェクトに登録するソース・ファイルを指定します。

追加(A)... ボタンを押してください。

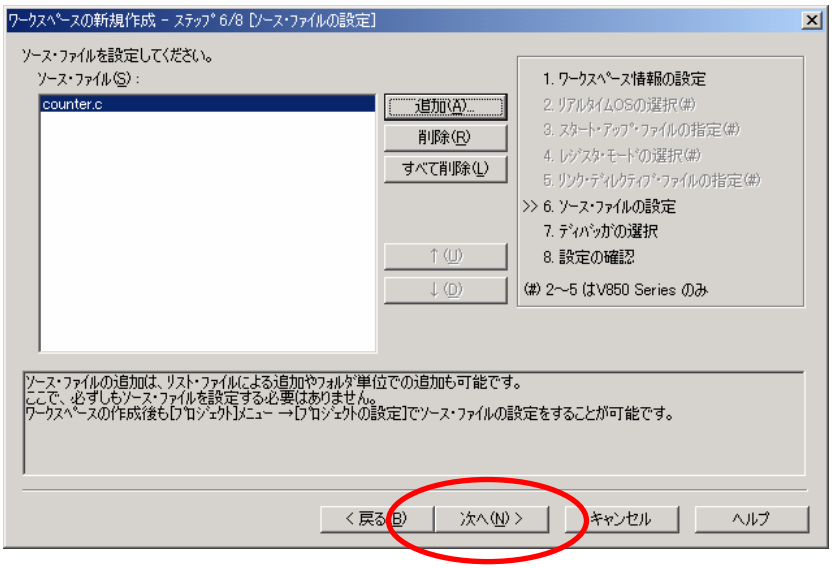


“ counter.c ”を選択し，

開く(O) ボタンを押します。

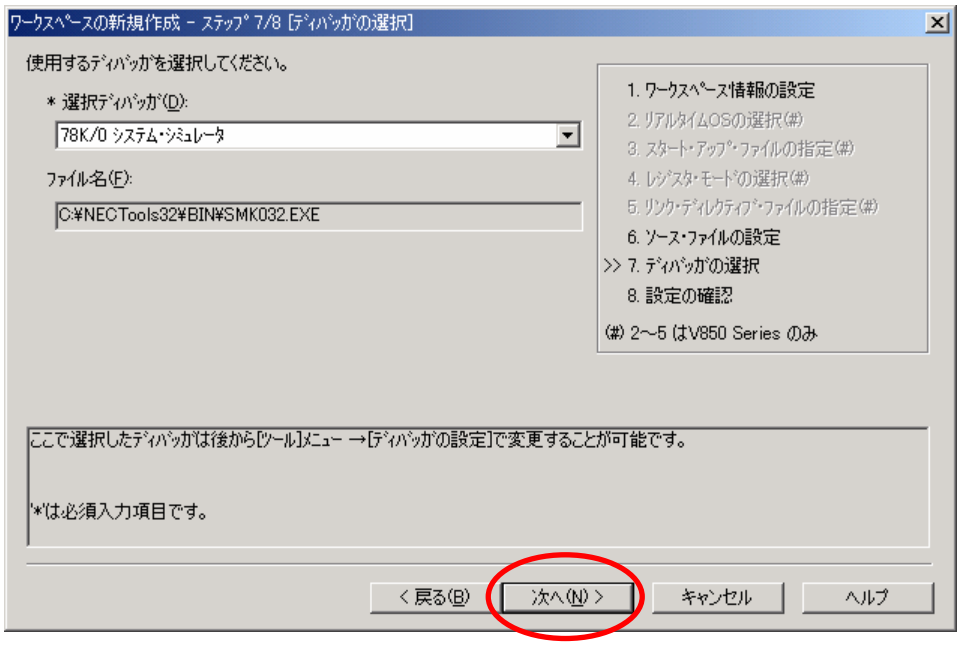


ここで，ソース・ファイル名が出てこない場合は，ワークスペース情報の設定において，
フォルダの位置の設定が正しく行われていません。
もう一度，設定をやり直してください。



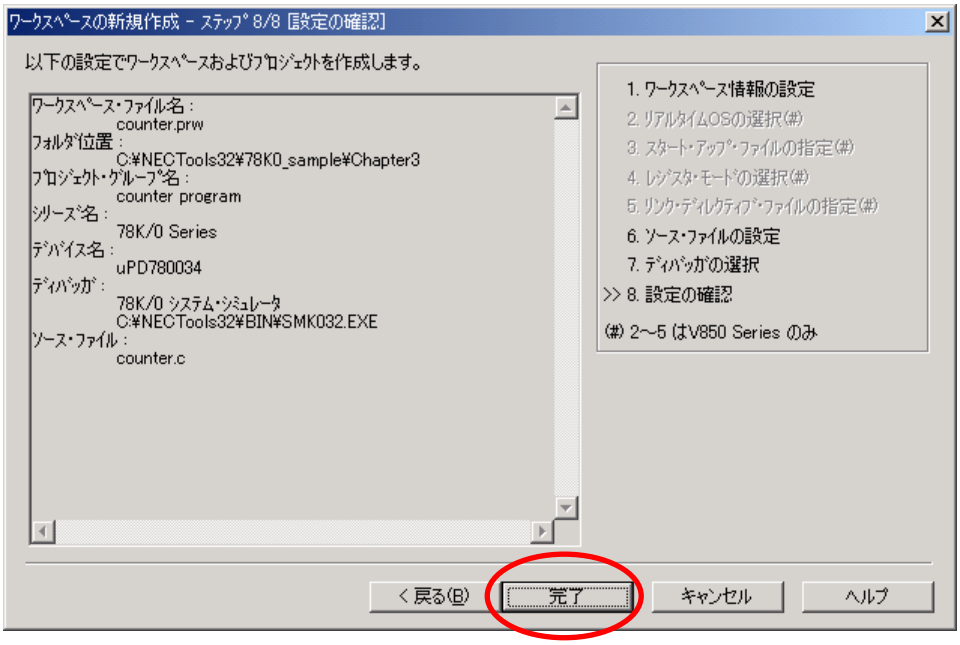
ソース・ファイル“ counter.c ”
が登録されます。

使用するディバッガを指定します。

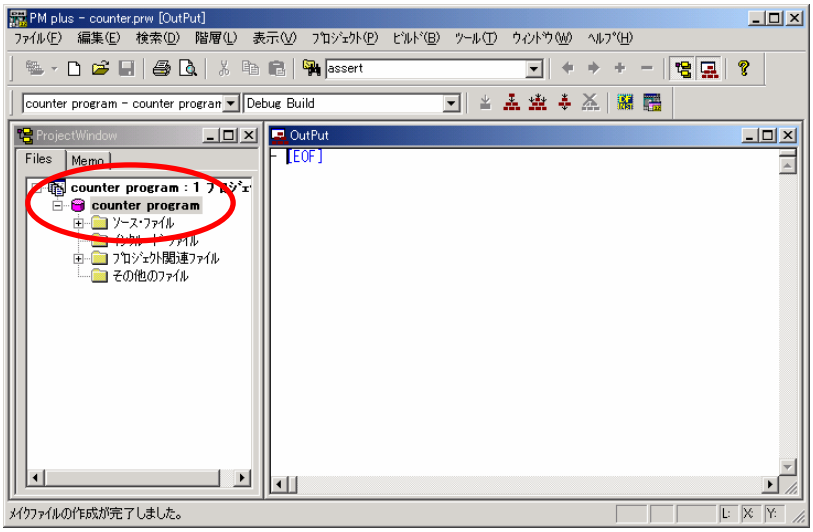


次へ(N)>　ボタンを押してください。

設定した内容を確認します。



設定した内容が間違っていなければ，完了　ボタンを押してください。



プロジェクト“counter program”が登録されました。

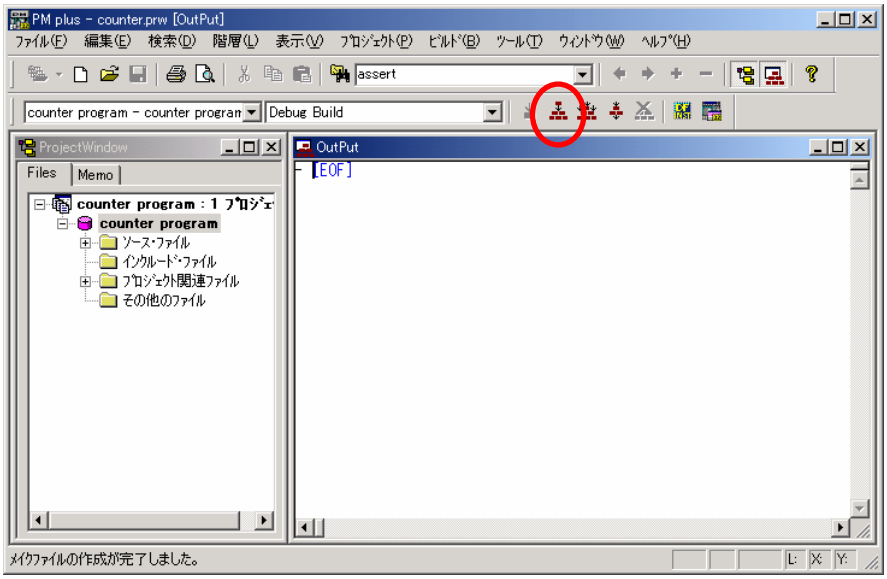
これで，ワークスペースの作成は完了です。

ソース・ファイルは，後から随時追加登録することができます。

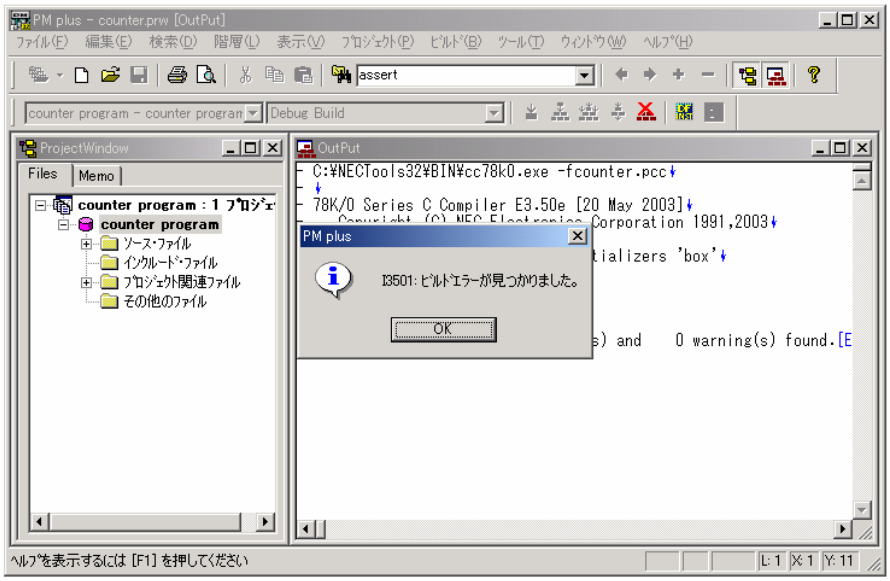
ソースの修正と実行形式の作成（1）

プロジェクトをビルドします。

PM plus のビルド・ボタン  **，またはメニューの［ビルド(B)］→［ビルド(B)］を選択してください。**



ビルド処理を実行します。



カウンタ・プログラムのソース中にエラーを検出したので，エラー・メッセージが表示されました。

OK ボタンを押してください。

では，エラーの修正を始めましょう。

ソースを修正します。

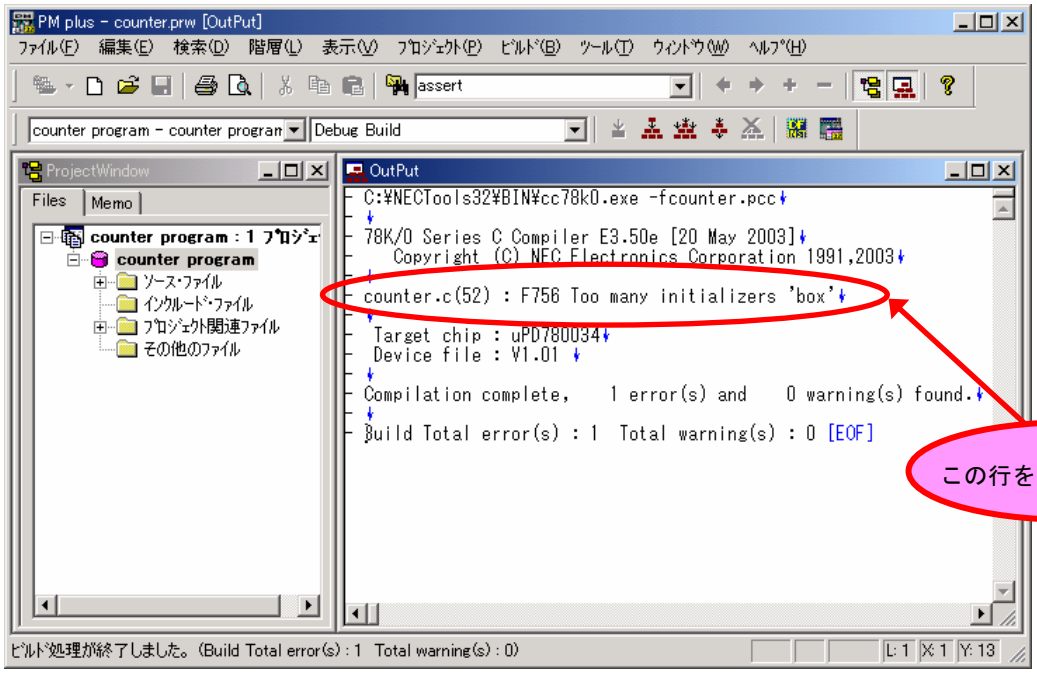
アウトプット・ウインドウに詳細なエラー・メッセージが表示されています。

エラーの表示行 “ counter.c(52):F756 Too many initializers’box **” をダブルクリックしてください。**

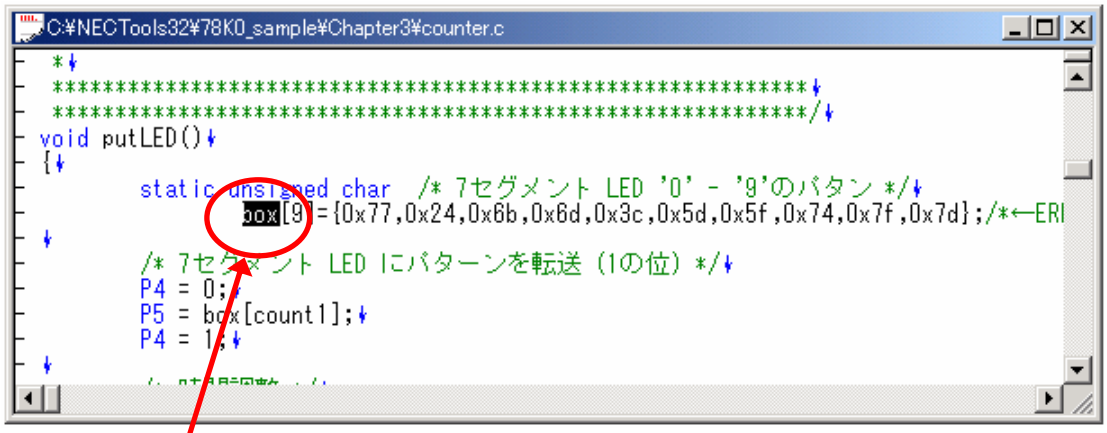
エラーの読み方について

“ counter.c (52):F756 Too many initializers’box ”

ソース・ファイル名　　行番号　　エラー・メッセージ（エラー要因）を示しています。



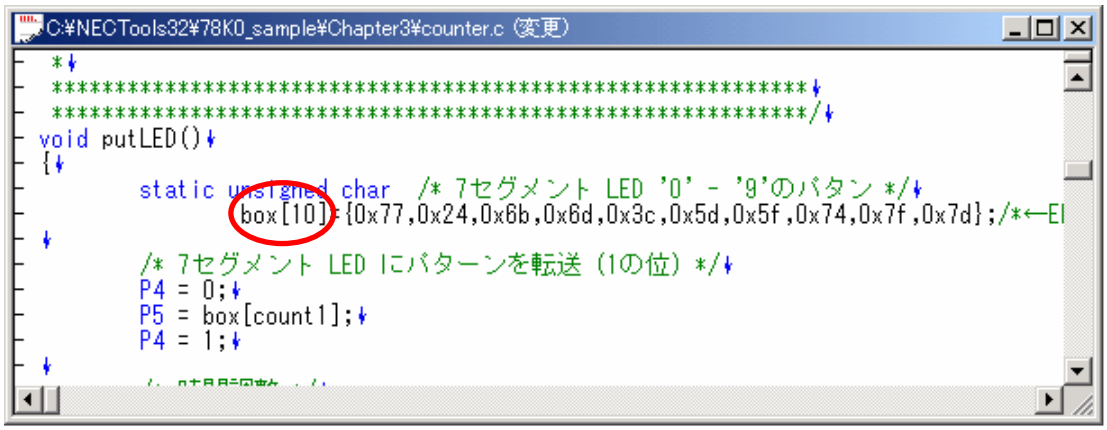
エディタが
開きます。



カーソルがエラー原因の
行に置かれています。

52 行目では，初期値として記述された｛｝内のデータの個数（10 個）に対して，box 配列の領域が不足しています。

“ box[9] ” を“ box[10] ” に修正してください。

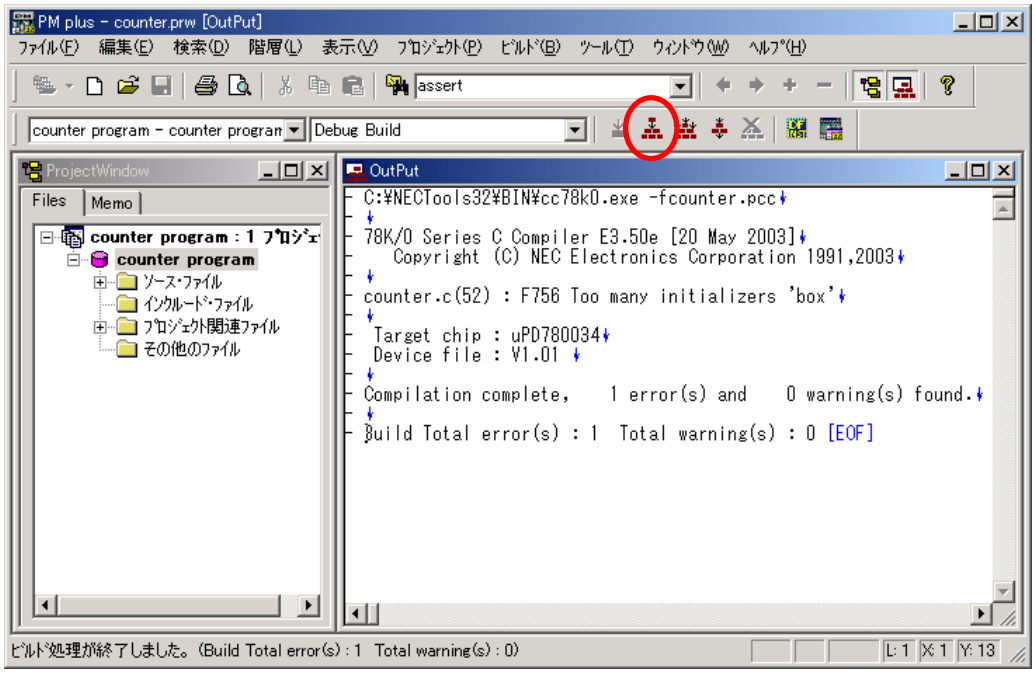


これで，ソースの修正が終了しました。

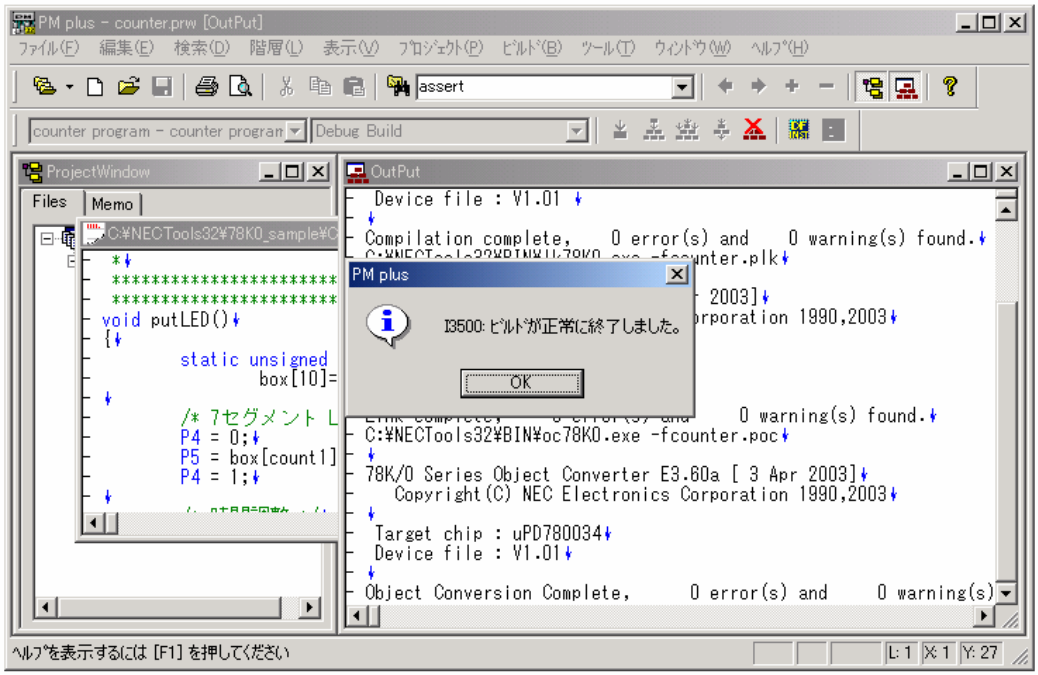
エラーの修正が終了したので，再びビルドします。

PM plus のビルド・ボタン  ，またはメニューの［ビルド(B)］→［ビルド(B)］を選択してください。

PM plus のエディタ機能の場合， ソースの修正内容はビルド時に自動的にセーブされます。



ビルドを行います。



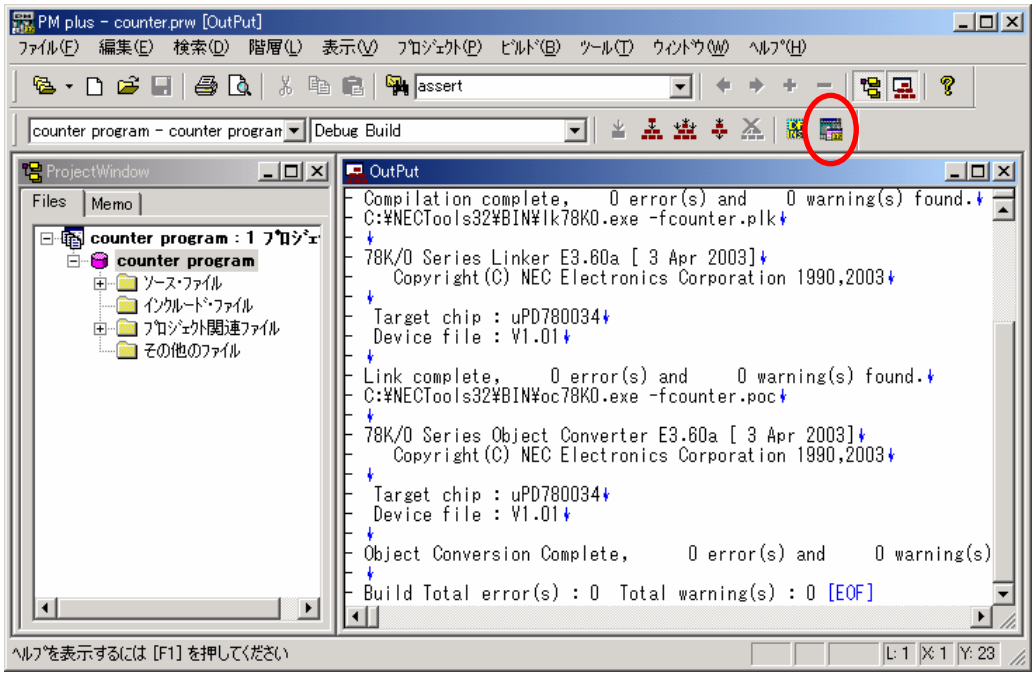
ビルド処理が正常に終了し，実行形式が作成されました。

実行形式のディフォールトのファイル名は，先頭に登録されたソース名.lmf です。

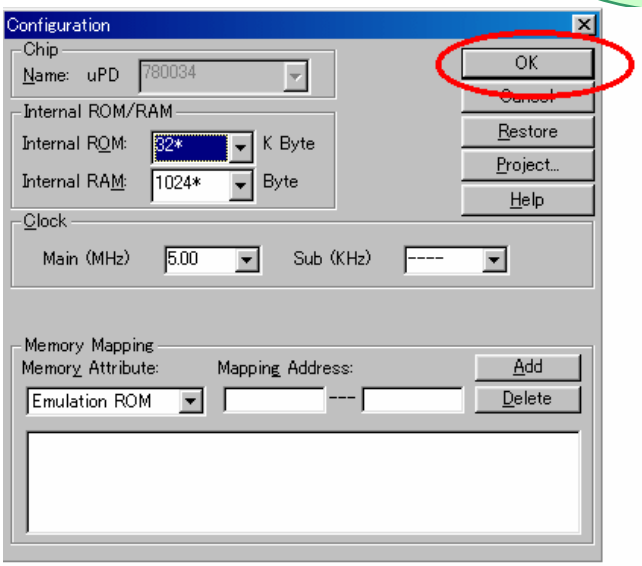
システム・シミュレータ（SM78Kxx）の起動

SM78K0 を起動します。

PM plus のディバグ・ボタン  ，またはメニューの［ビルド(B)］→［ディバグ(D)］を選択してください。



コンフィギュレーション
・ダイアログが開きます。

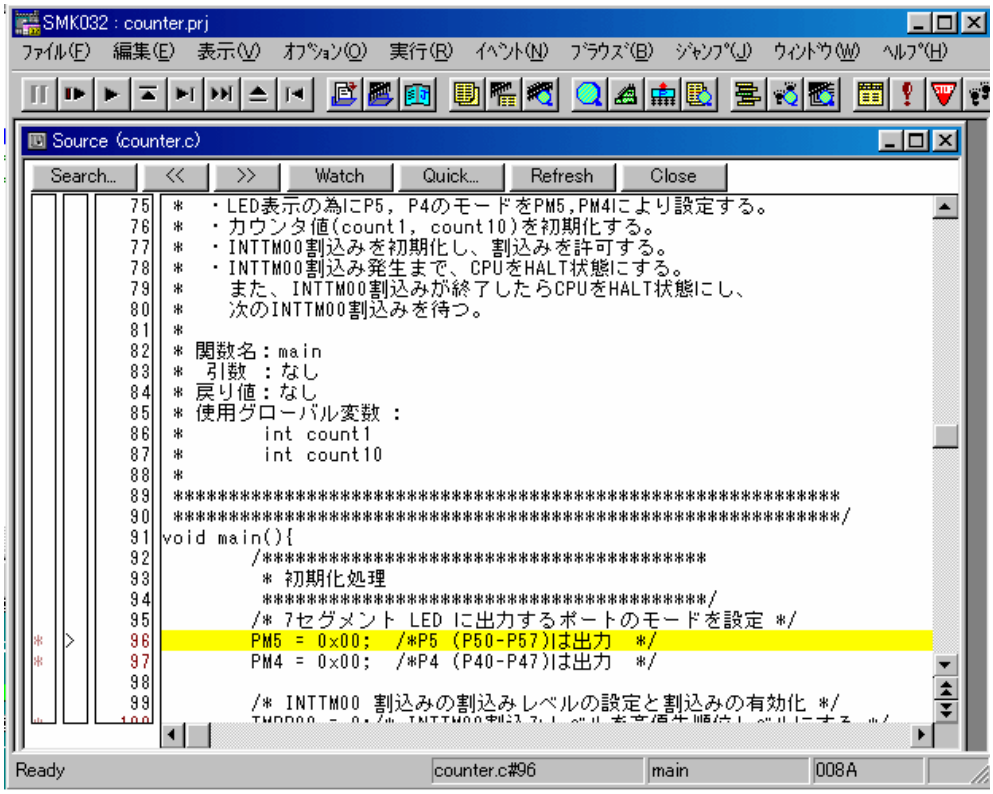


このダイアログでは，メモリ・マッピング，クロックなどの設定を行います。

ただし，この章で扱うカウンタ・プログラムは μPD78xxxx に内蔵されている ROM と RAM で動作するため，ここでは設定を変更する必要はありません。

OK ボタンを押してください。

SM78Kxx のメイン・ウインドウが開きます。



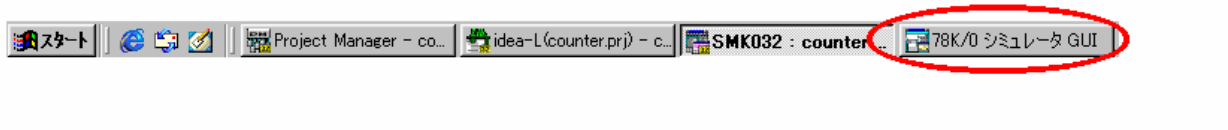
入出力パネルの設定

プログラムを実行する前に，カウンタ・プログラムで使用するボタンと 7 セグメント LED の設定を行います。

まず，入出力パネルでボタンの設定を行います。

入出力パネル・ウインドウは，78Kxx シミュレータ GUI ウインドウから開くことができます。

では，タスク・バーの“78Kxx シミュレータ GUI”を選択して，78Kxx シミュレータ GUI ウインドウを開いてください。



78Kxx シミュレータ GUI ウインドウが開きます。

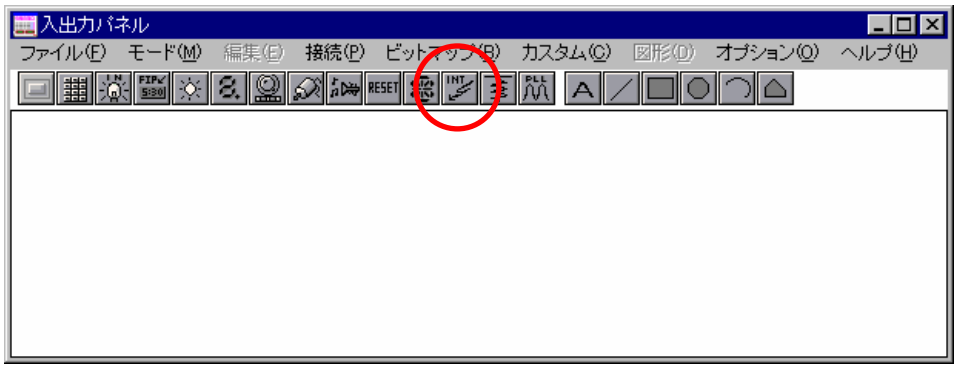
78Kx シミュレータ GUI ウインドウのメニューの［外部部品(E)］→［入出力パネル(P)...］を選択してください。

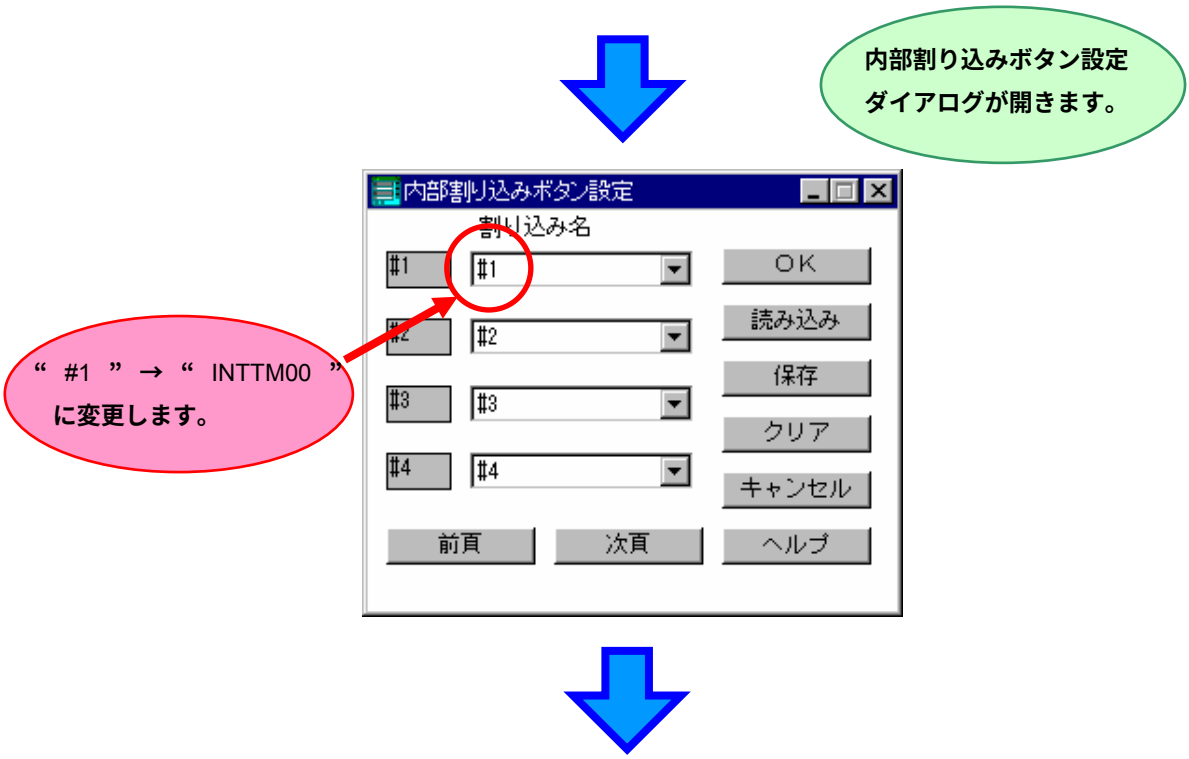
入出力パネル・ウインドウが開きます。

内部割り込みボタンの設定をします。

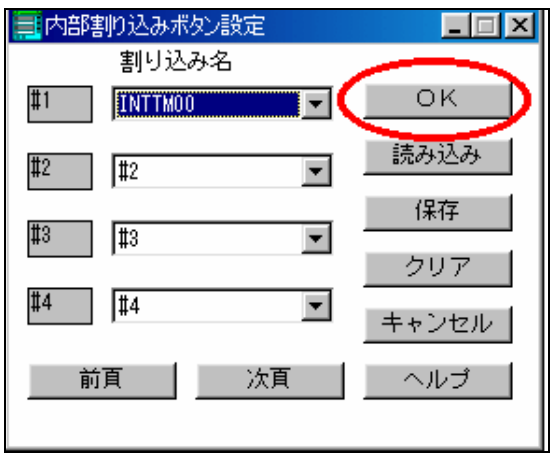
まず，入出力パネル・ウインドウの内部割り込みボタン  ，またはメニューの［接続(P)］→［内部割り込みボタン(I)...］を選択してください。

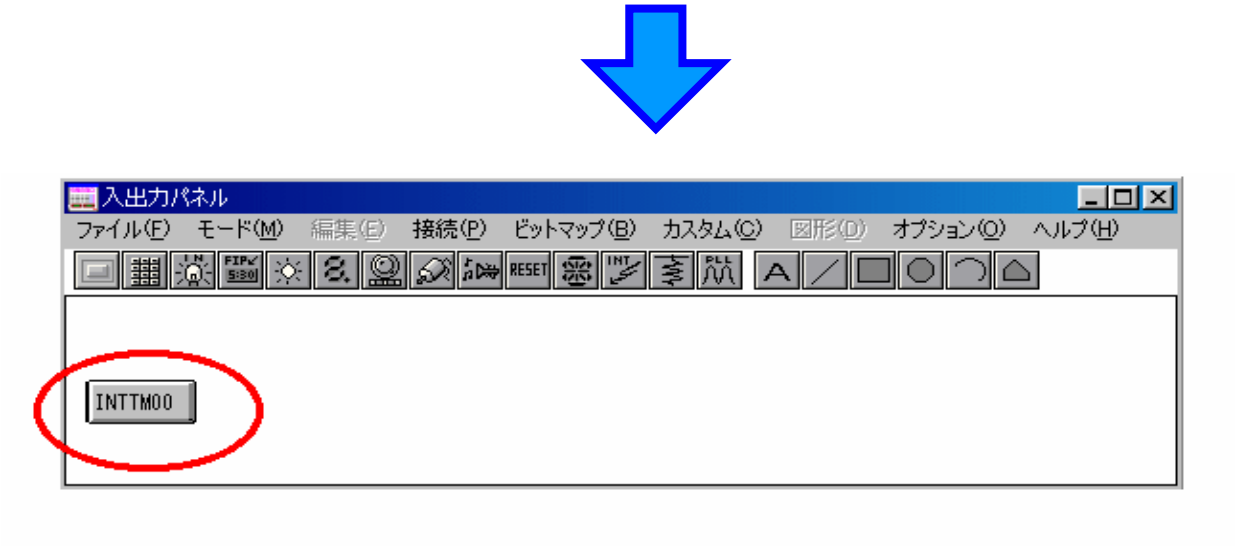
内部割り込みボタン設定ダイアログが開きますので，割り込み名の“ #1 ”を 78K0 は“ INTTM00 ”，78K0S は“ INTWT ”，78K4 は“ INTC00 “ に変更してください。





OK ボタンを押してください。





入出力パネル上に内部割り込みボタンが表示されます。

内部割り込みボタンとは？

SM78Kxx のディバグ機能の 1 つです。タイマなどの CPU 周辺機能が発生させる内部割り込みを，ユーザが任意にボタンを押すことにより，疑似的に発生させることができます。

内部割り込みが発生する条件を整えるのが難しい場合でも，この機能を使用すると，ディバグ時に容易に割り込みを発生させて割り込み処理の動作確認を行うことができます。

7 セグメント LED端子の設定を行います。

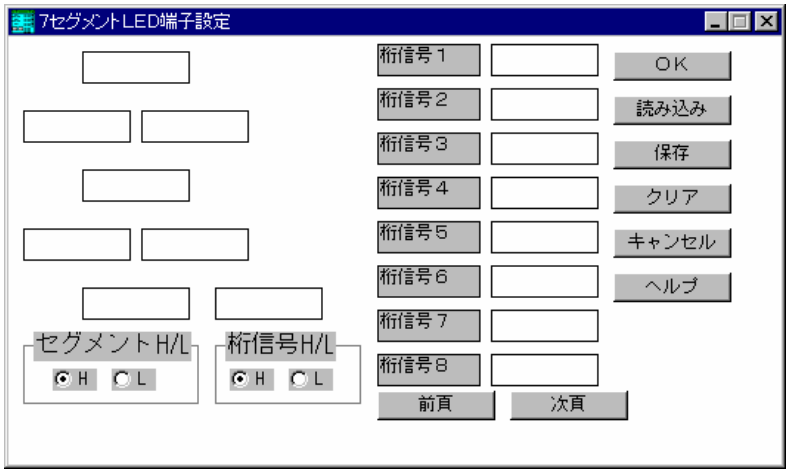
7 セグメント LED 端子に入出力ポート P1，P3 の各ビットを接続します。

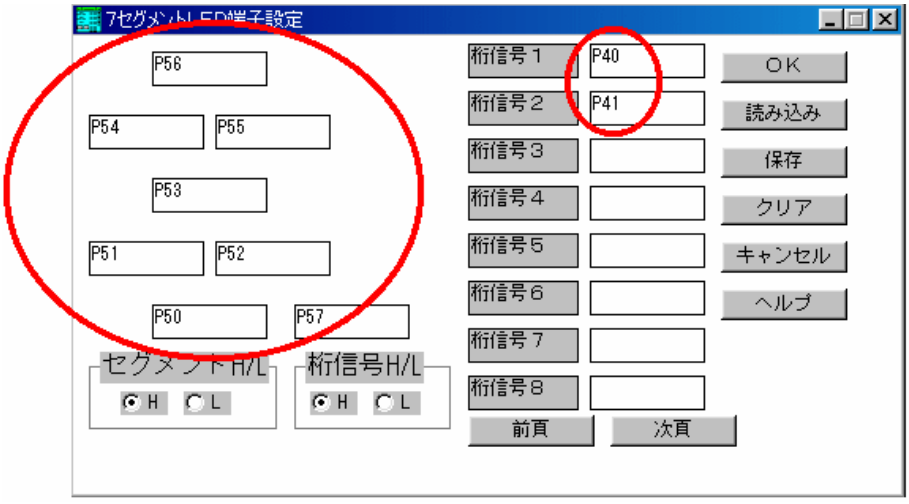
まず，入出力パネル・ウインドウの 7 セグメント LED 端子設定ボタン  ，またはメニューの［接続(P)］→［7 セグメント LED(S)...］を選択してください。

7 セグメント LED 端子設定ダイアログが開きますので，7 セグメント LED を下図のように設定してください。

7 セグメント LED 端子設定ダイアログが 開きます。

7 セグメント LED 端子を次のように設定します。

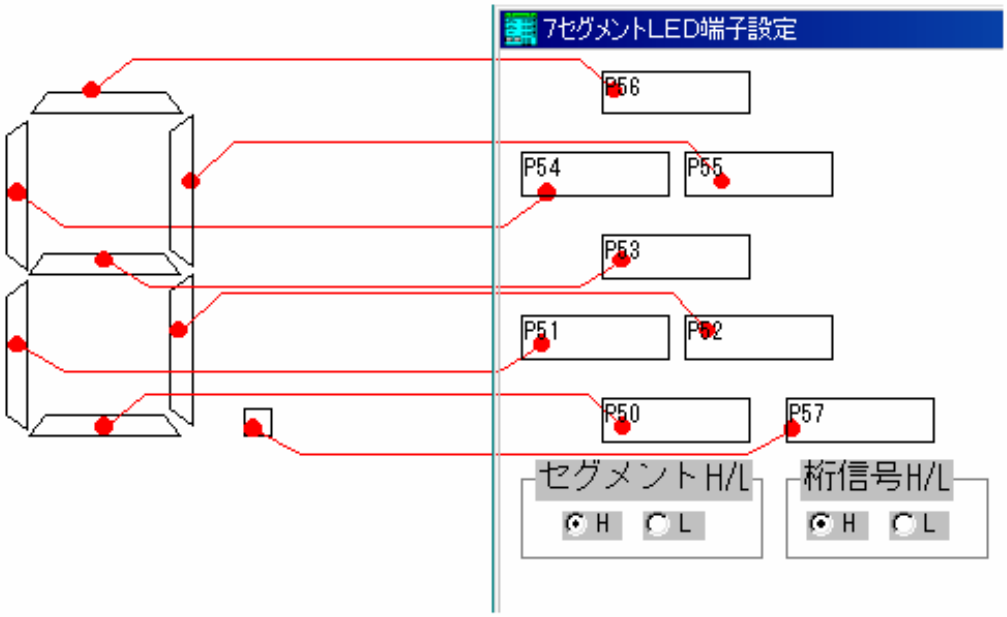




設定が終了したら，OK ボタンを押してください。

仮想 LED の表示と，7 セグメント LED 端子設定ダイアログのセグメント信号設定エリアの各設定枠は，次のように対応しています。78K0S の場合は P1X に対応します。

＜仮想 LED 表示＞　　＜7 セグメント LED 端子設定枠＞



入出力パネル上に仮想 LED が表示されます。

これで，ボタンと 7 セグメント LED の設定が終了しました。

7 セグメント LED とは？

疑似的なターゲット・システムを構築するために SM78Kxx が用意している外部部品の 1 つです。主に数値を表示するために，入出力ポートと接続して使用します。操作に使用する信号には，各桁共通の “ セグメント信号 ” と各桁独立の “ 桁信号 ” があります。“ セグメント信号 ” 端子に表示する値，“ 桁信号 ” 端子に表示する桁位置の順に入出力ポートから出力します。

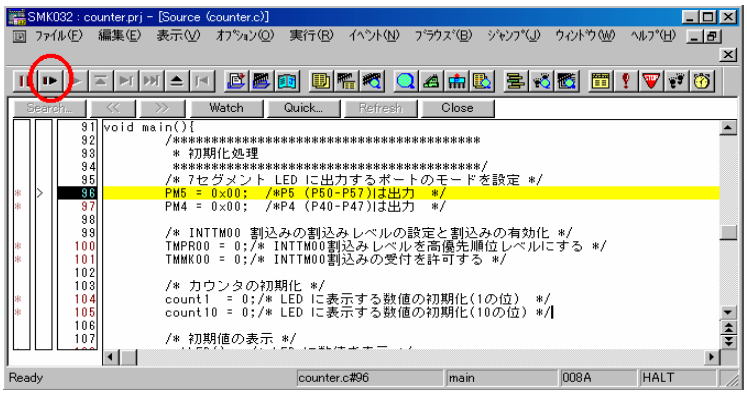
詳細については，SM78K シリーズ システム・シミュレータ Ver.2.52 ユーザーズ・マニュアル 操作編（U16768J）を参照してください。

プログラムの実行（1）

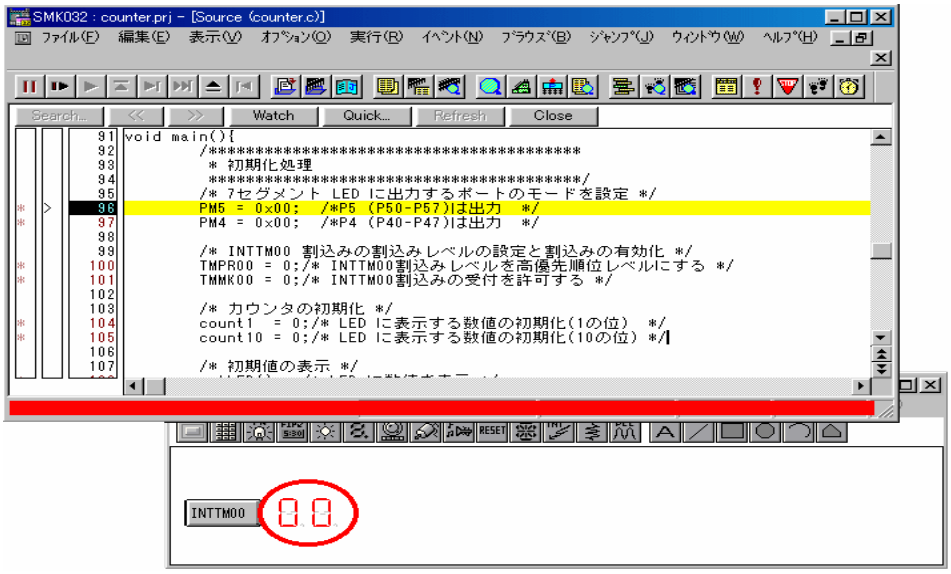
カウンタ・プログラムを実行します。

SM78Kxx のリスタート・ボタン  ，またはメニューの［実行(R)］→［リスタート(R)］を選択してください。

この操作により，エミュレーション CPU をリセットしてから，プログラム実行を行います。



プログラムを実行します。



プログラムの実行中は，ステータス表示エリアが赤く変化します。入出力パネルには“ 00 ”が表示され，INTTM00 ボタンによる入力待ち状態になっています。78K0S は INTWT ボタン，78K4 は INTC00 ボタンとなります。

ここで，上記の動作が起きない場合は，次の操作を行ってください。

① LED が点灯しない場合

→ 再度，リスタートしてください。

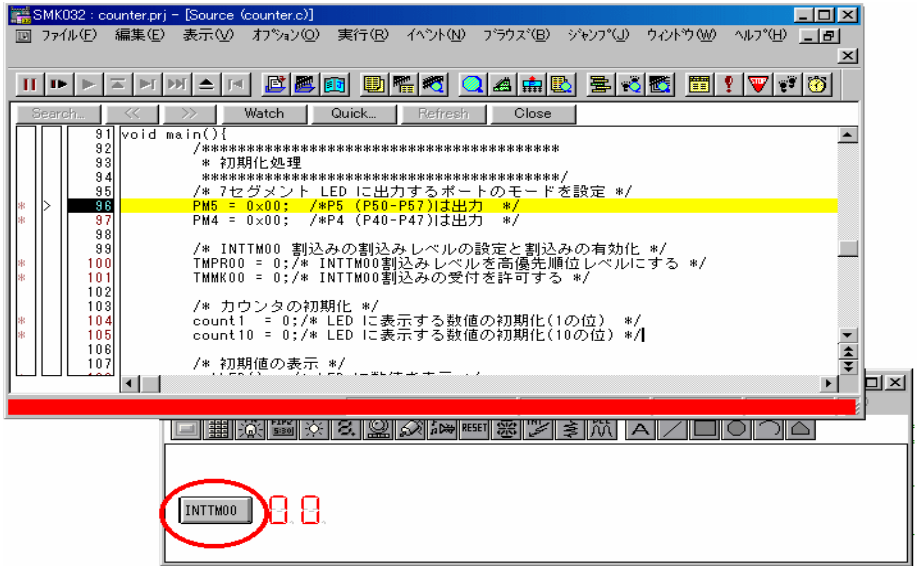
リスタートしても動作が改善されない場合は，7 セグメント LED 端子の設定をやり直してください。

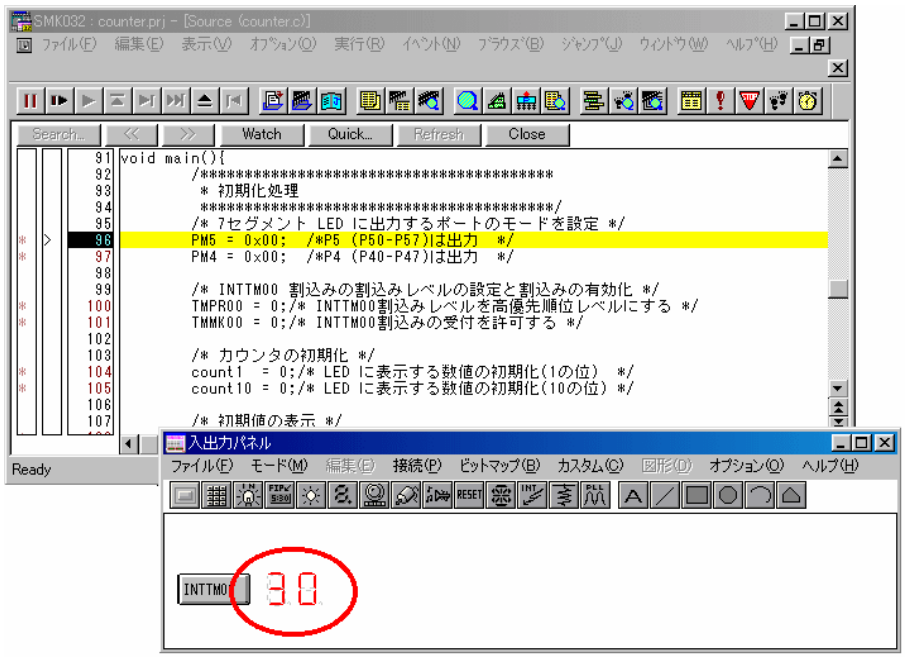
② LED が“ 00 ”以外の表示を行った場合

→ 7 セグメント LED 端子の設定をやり直してください。

INTTM00 ボタンを数回押してください。

INTTM00 ボタンを押すたびに，カウンタが 1 つずつカウント・アップする仕様です。





INTTM00 ボタンを押すと，LED の表示は 10 の位のみカウント・アップし，1 の位は変化しません。したがって，仕様に合った動作をしていないことがわかります。

ここで，上記の動作が起きない場合は，次の操作を行ってください。

① INTTM00 ボタンを押しても，何もアクションが起きない場合

→ 内部割り込みボタンの設定をやり直してください。

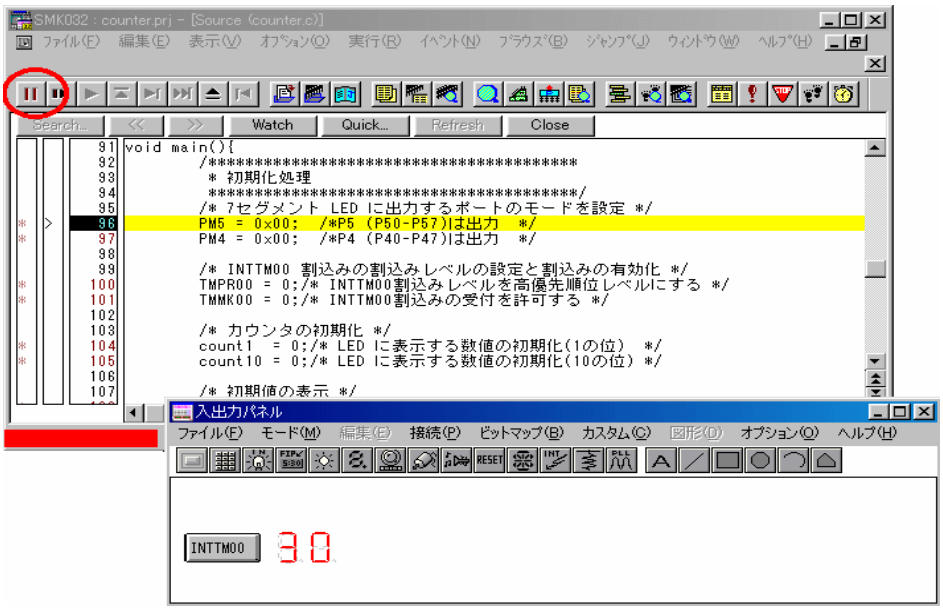
② LED が上記の動作（10 の位のみカウント・アップ）でない場合

→ 7 セグメント LED 端子の設定をやり直してください。

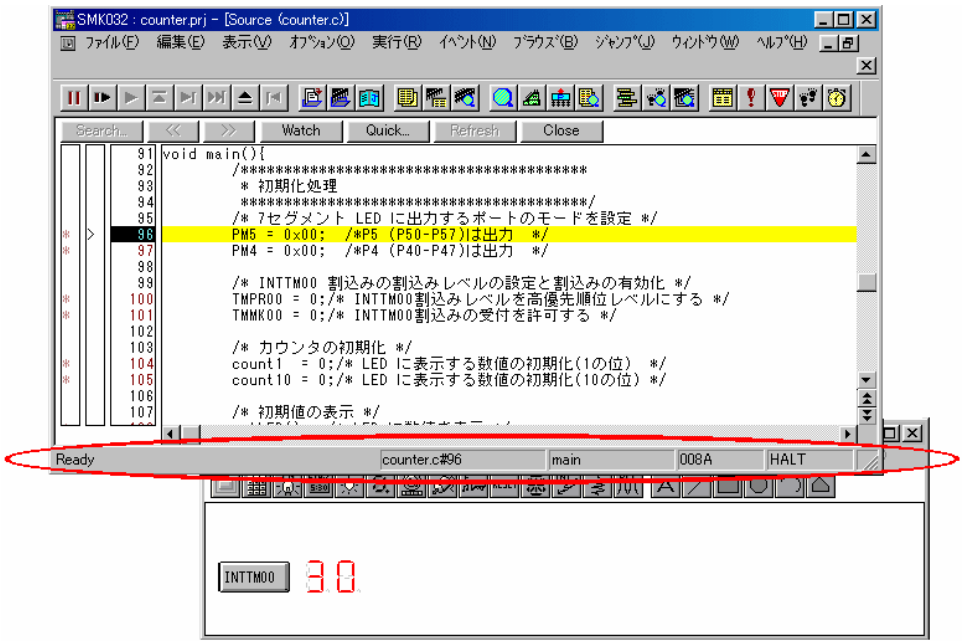
カウント・アップが正常に処理されていないので，ディバグを行いましょう。

プログラムを停止します。

SM78Kxx **の停止ボタン**  **，またはメニューの［実行(R)］→［ストップ(S)］を選択してください。**



プログラムを停止します。



プログラムを停止すると，ステータス表示エリアの色は元に戻ります。

ディバグ

count1 には LED の 1 の位の値が，count10 には LED の 10 の位の値が入っています。

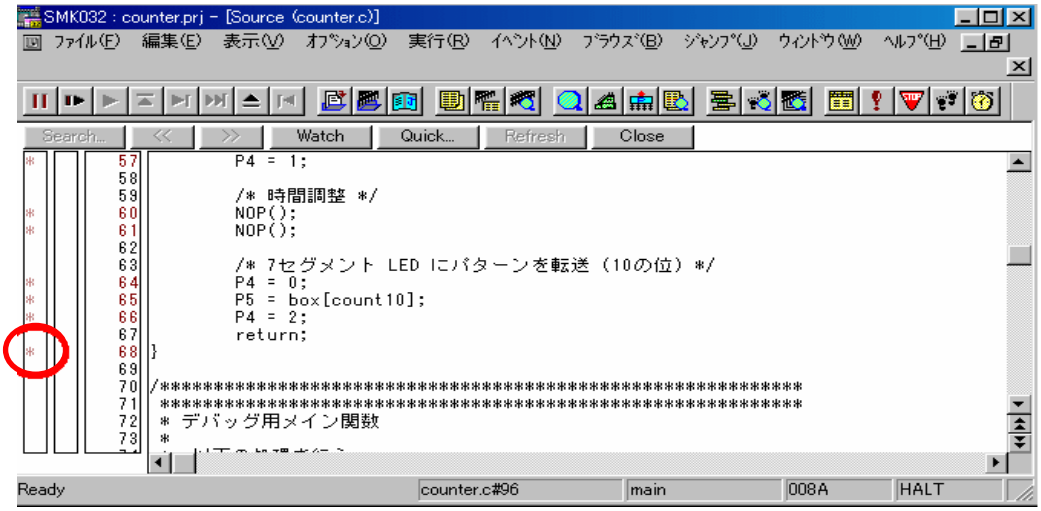
詳細については，「カウンタ・プログラムの仕様」をご覧ください。

まず，LED 表示ルーチン（putLED()関数）を実行したときに count1，count10 にどのような値が設定されているのか調べてみましょう。

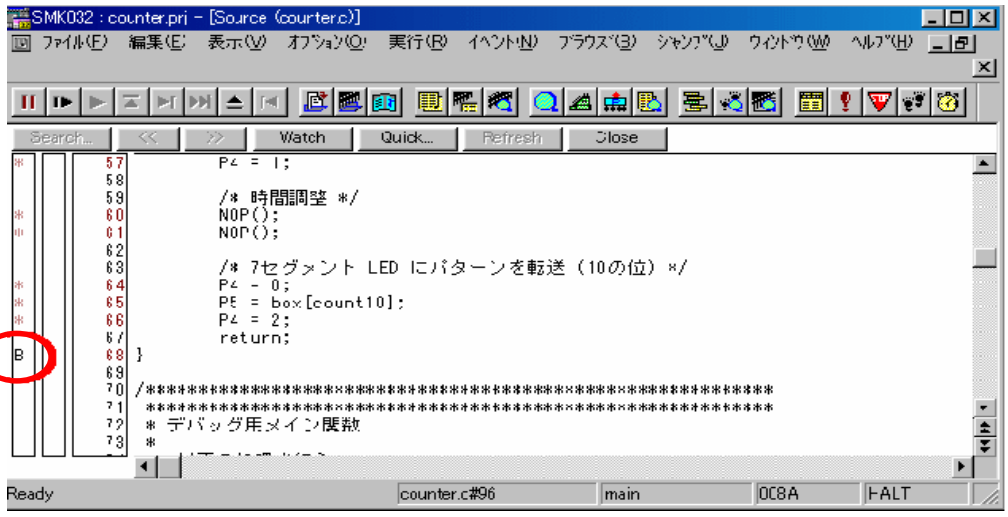
78K0 と 78K4 の場合は 68 行目に，78K0S の場合は 69 行目にブレークポイントを設定します。

ブレークポイントの設定できる行には“ * ”が表示されています。

78K0 と 78K4 の場合は 68 行目の“ * ”を，78K0S の場合は 69 行目をクリックしてください。



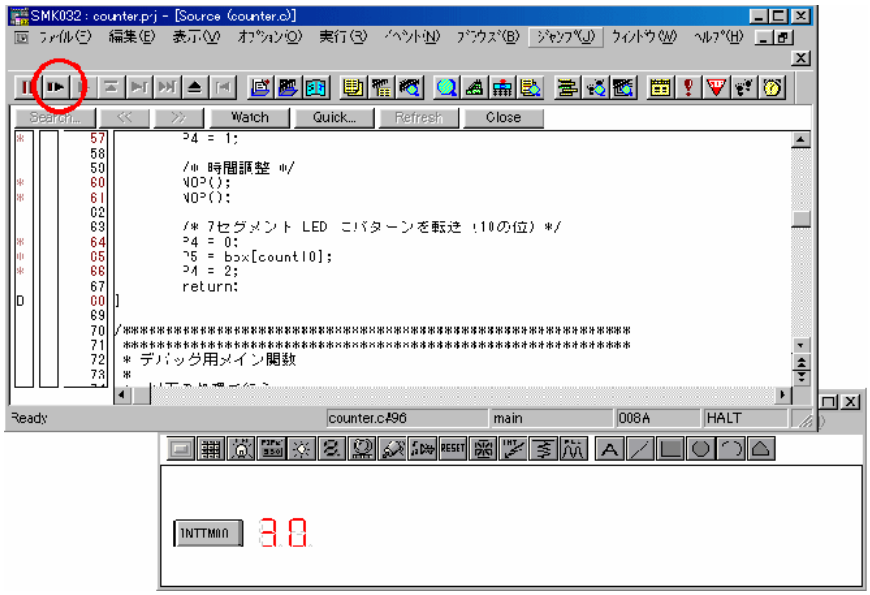
“ * ”が“ B ”に変化します。



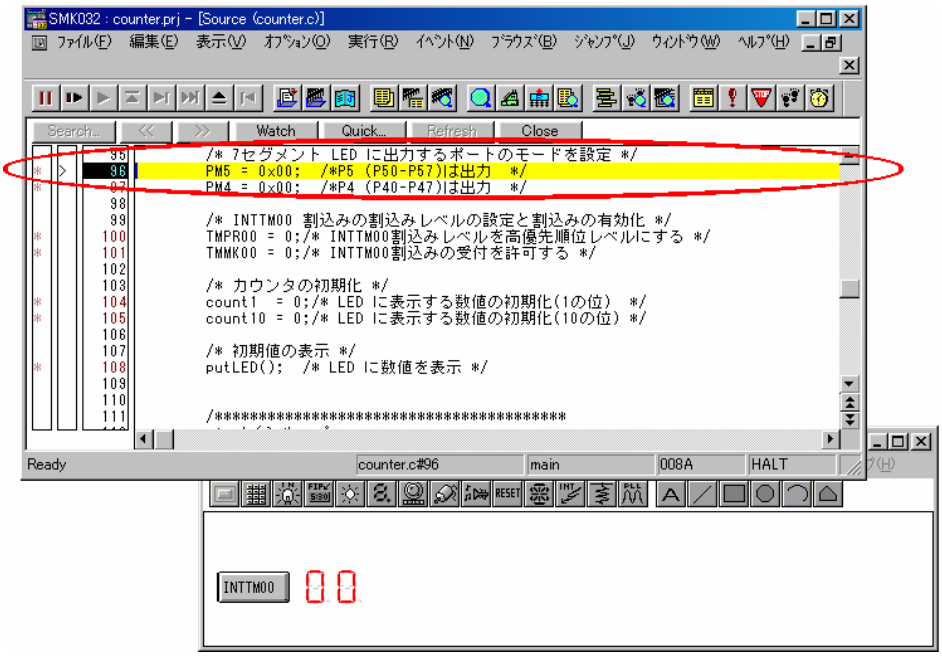
68 行目（78K0 と 78K4）または 69 行目（78K0S）にブレークポイントが設定されました。

それでは，プログラムを実行します。

SM78Kxx のリスタート・ボタン　  　，またはメニューの［実行(R)］→［リスタート(R)］を選択してください。



プログラムを実行します。



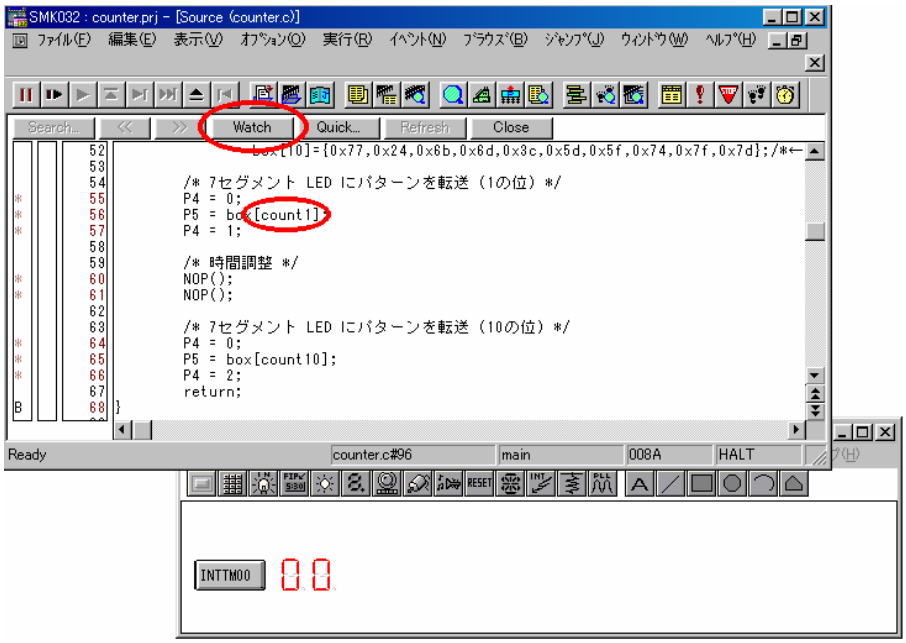
プログラムを実行するとすぐに，ブレークポイントでプログラムが停止します。

プログラムが停止した行には“ > ”が表示され，黄色に反転します。

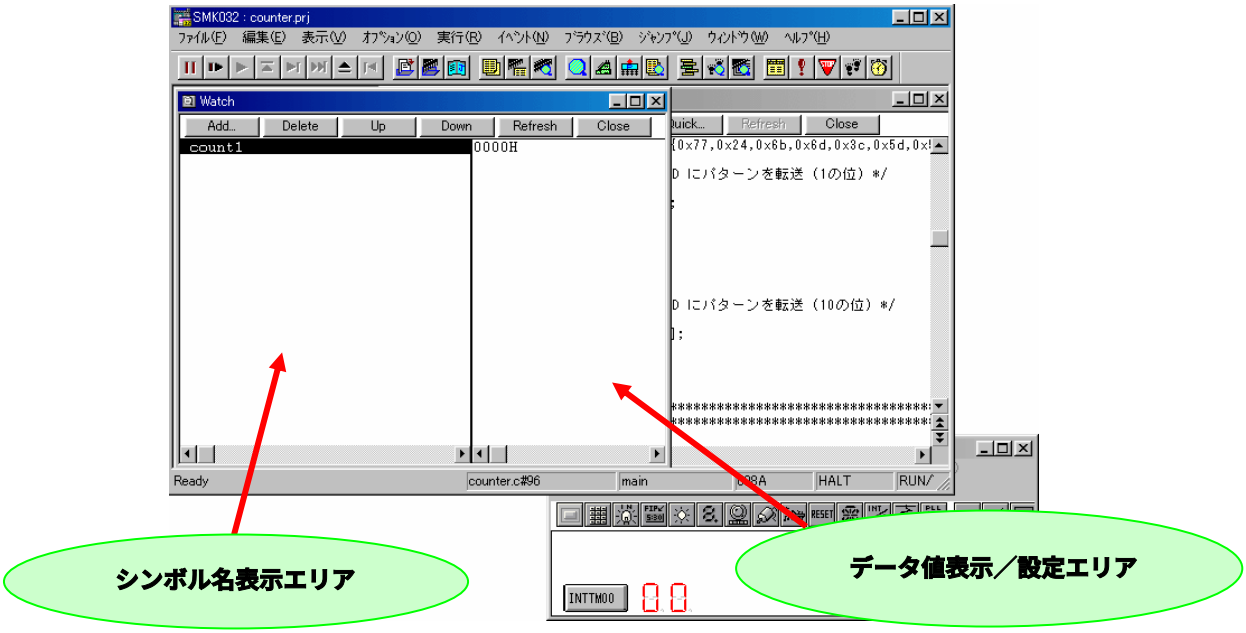
変数の値を見るには，ウォッチ・ウインドウを使用します。

ウォッチ・ウインドウを開き，このときの count1，count10 それぞれの値を確認しましょう。

56 行目の “ count1 ” をダブルクリックし反転表示させてから，Watch ボタンを押してください。



ウォッチ・ウインドウが開きます。

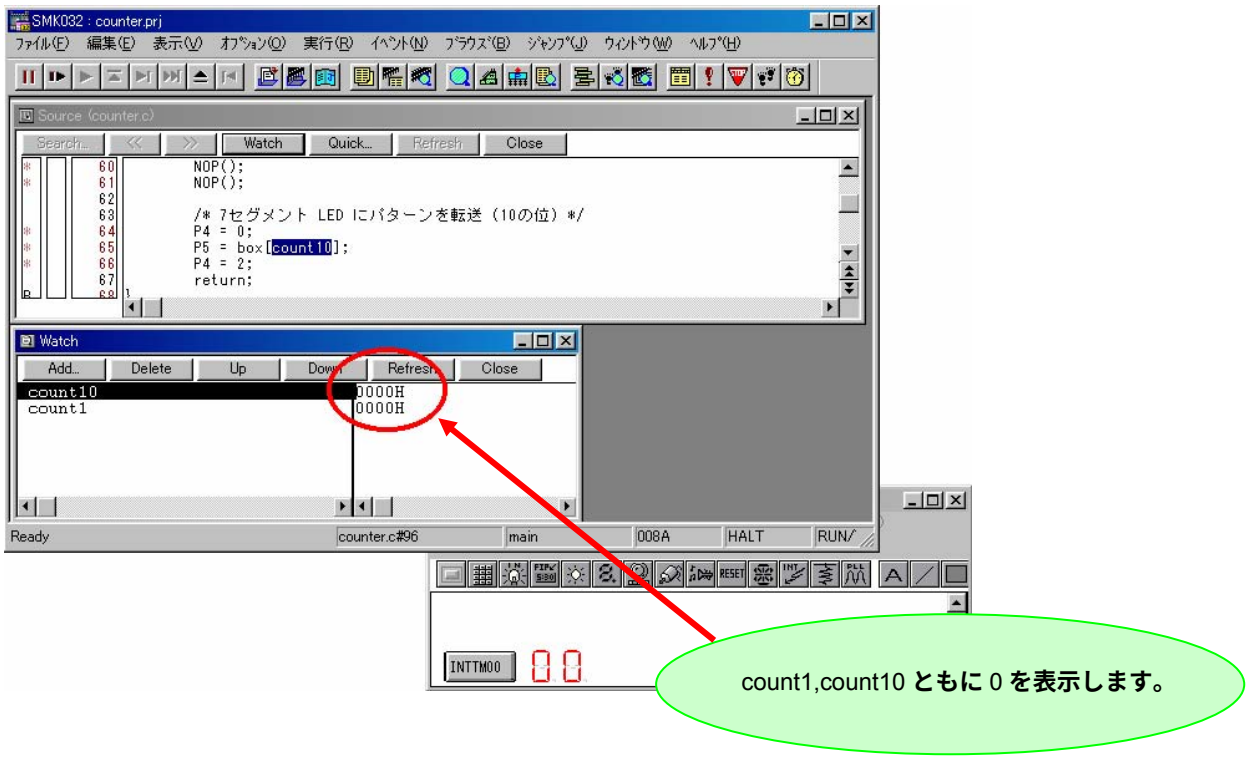


ウォッチ・ウインドウはシンボル名表示エリアとデータ値表示／設定エリアで構成されています。

詳細については，SM78K シリーズ システム・シミュレータ Ver.2.52 ユーザーズ・マニュアル 操作編（U16768J）を参照してください。

同様に，ソース・テキスト・ウインドウ（Source ウインドウ）の 65 行目の“ count10 ”をダブルクリックし，反転表示させてから，Watch ボタンを押してください。

ウォッチ・ウインドウに count10 が追加されます。



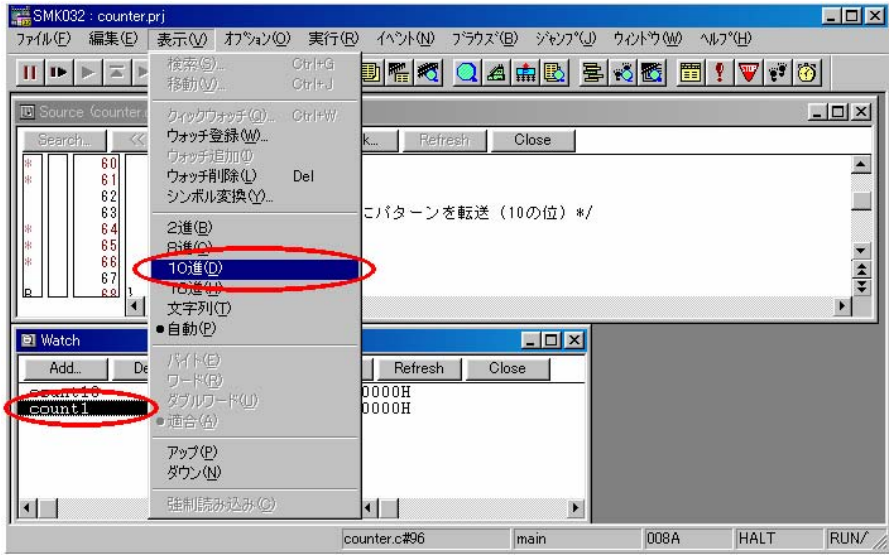
ウォッチ・ウインドウを見ると，count1，count10 ともに初期値が 0 であることが確認できます。

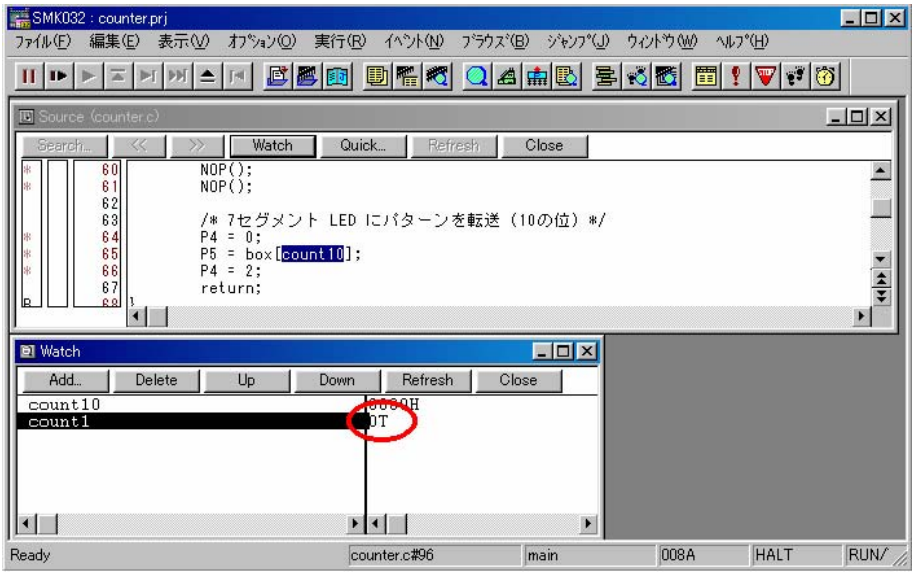
同様に，入出力パネルの LED も初期値 00 を表示していることが確認できます。

ウォッチ・ウインドウのデータ値表示／設定エリアは 16 進表示ですが，SM78Kxx の［表示(V)］メニューにより変更することができます。

では，count1 の値を 10 進表示に変更してみましょう。

ウォッチ・ウインドウの count1 をクリックして反転させたあとに，SM78Kxx のメニューの［表示(V)］→［10 進(D)］を選択してください。



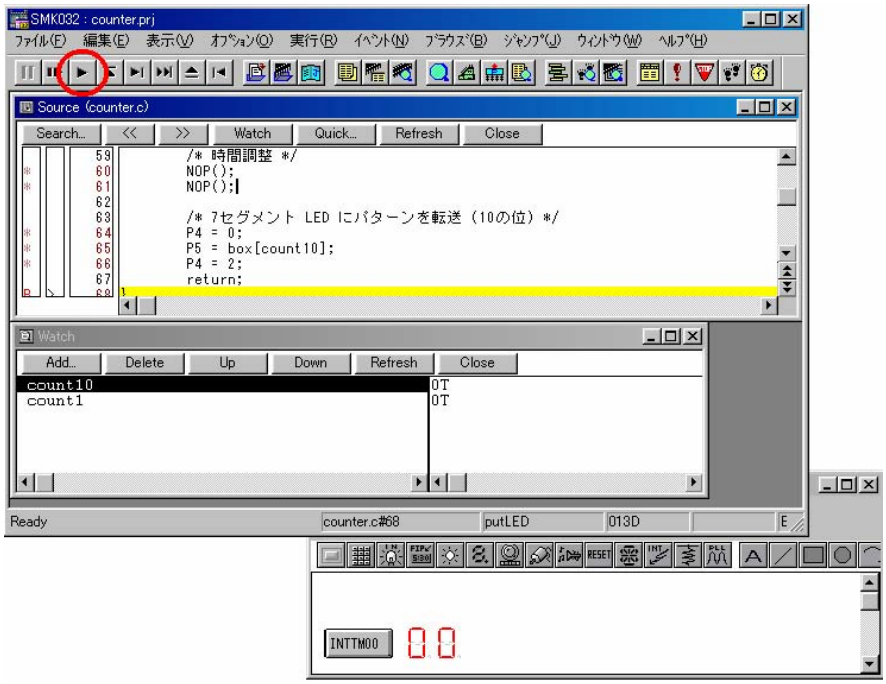


同様に count10 の値も 10 進表示に変更してください。

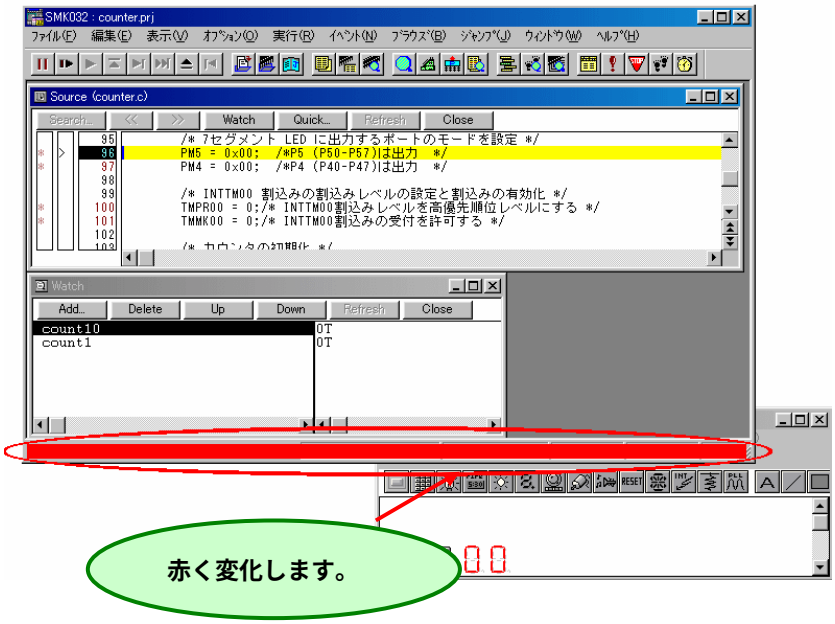
次に，INTTM00 ボタンを押したときに，LED に情報が転送され，正しく表示されているかどうかを確認します。

SM78Kxx のスタート・ボタン　　，またはメニューの［実行(R)］→［継続して実行(G)］を選択してください。

プログラム実行を停止行から継続して行うため，“ リスタート ” ではなく “ スタート ” を使用します。



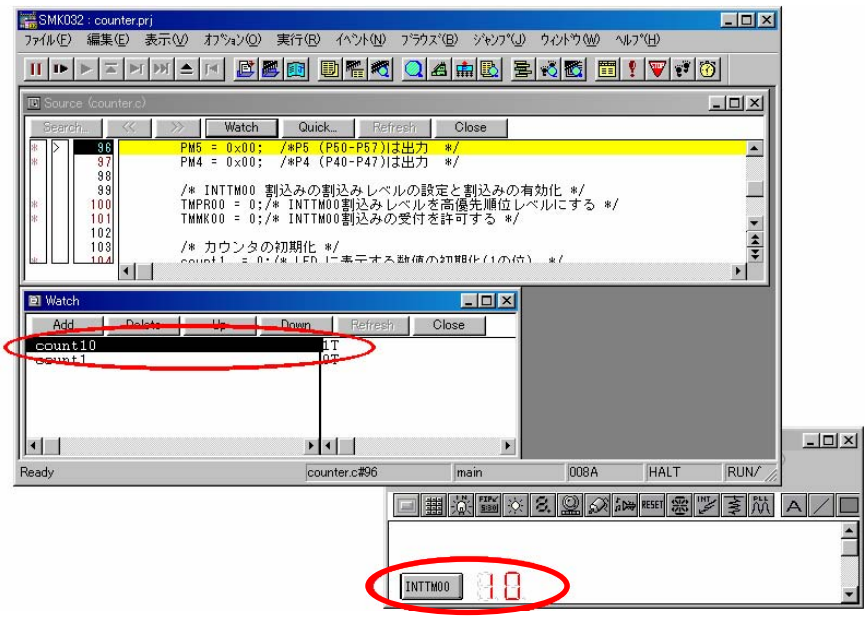
プログラムを実行します。



赤く変化します。

INTTM00 ボタンを押したときの，LED 表示ルーチン（putLED()関数）の動作を見てみましょう。

INTTM00 ボタンを押して，ウォッチ・ウインドウの表示を確認してください。



INTTM00 ボタンを押すと，ブレークポイントでプログラムが停止します。

ここで，カウント・アップが正常に動作していれば，count10 に 0，count1 に 1 の値が設定されます。

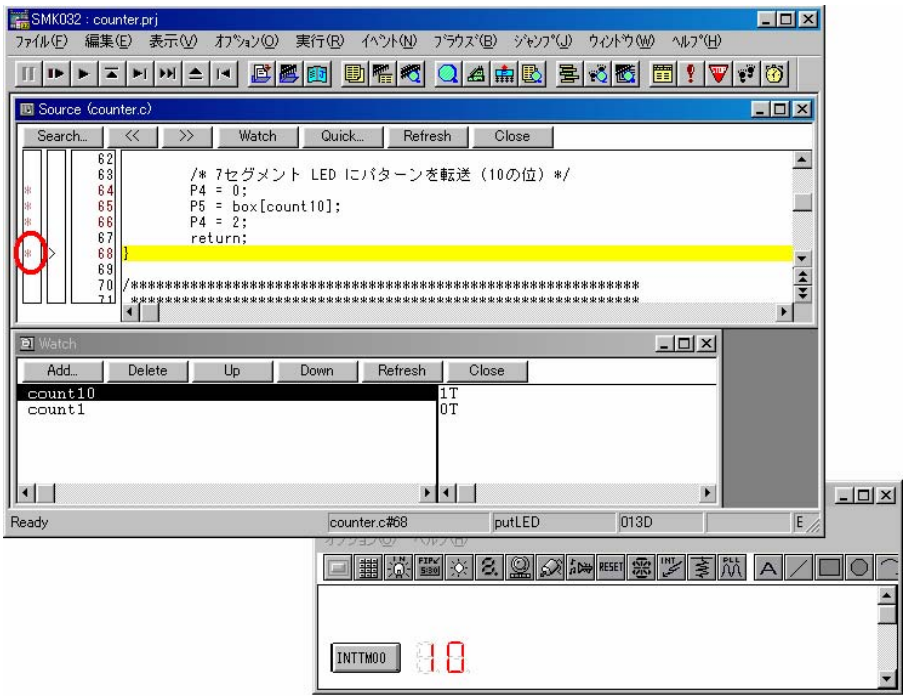
しかし，ウォッチ・ウインドウを見ると，count10 が 1，count1 が 0 となっており，変数に正しくデータが設定されていないことがわかります。

一方，入出力パネルの LED は 10 となっており，ウォッチ・ウインドウの変数の値が LED に反映されています。

これにより，count10，count1 の値を LED に表示する処理は，正しく行われていることが確認できます。

表示処理は正しいことが確認できましたので，次はカウント・アップ処理が正しく行われているかどうかを確認しましょう。

これから，LED 表示ルーチン（putLED()関数）とは別の箇所にあると考えられるエラーを探しますので，putLED()でプログラムが停止しないようにブレークを解除します。



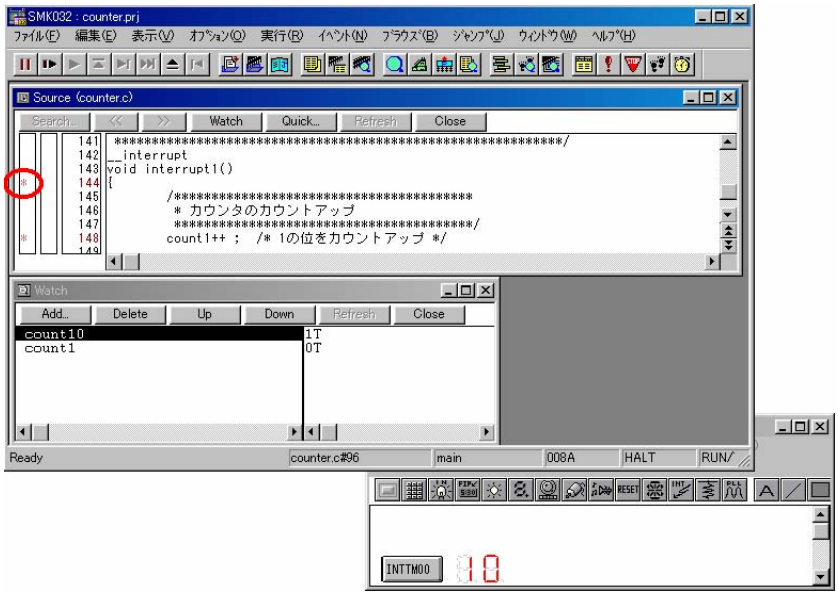
“ B ”をクリックすると“ * ”に戻り，ブレークポイントが解除されます。

次に，INTTM00 ボタンを押したときのカウント・アップ処理の動作を確認します。

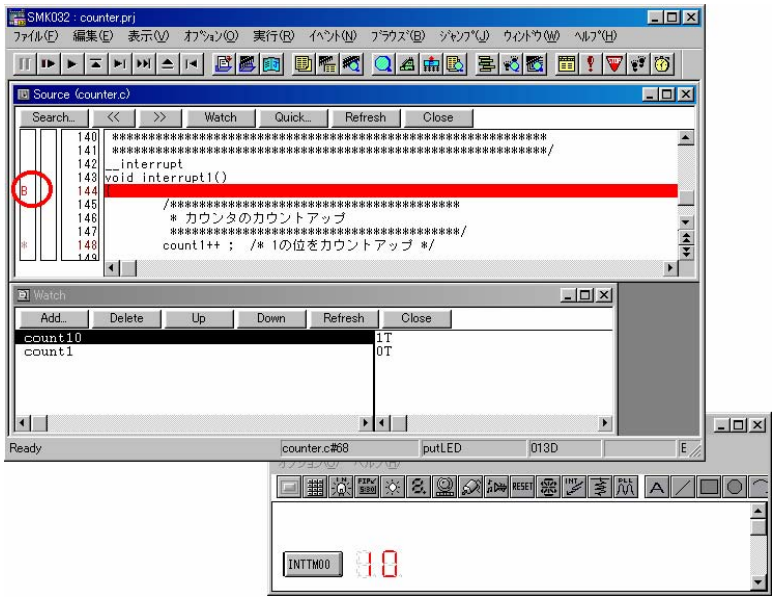
内部割り込みが発生したときに実行される関数（interrupt1()）の中で，count1，count10 の値がどのように変化するかを調べてみましょう。

144 行目にブレークポイントを設定します。

144 行目の“ * ”をクリックしてください。



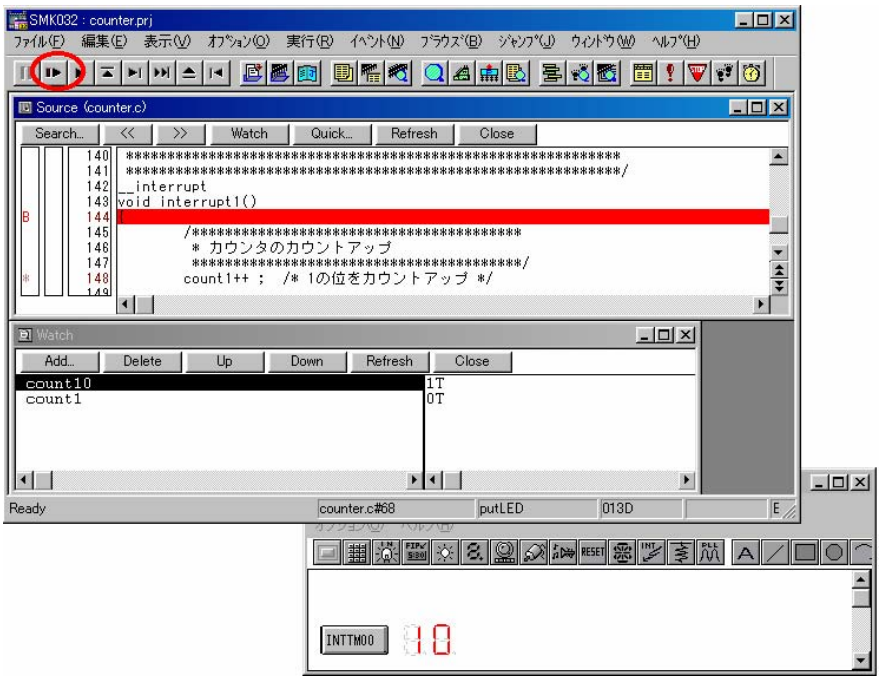
“ * ”が“ B ”に変化します。



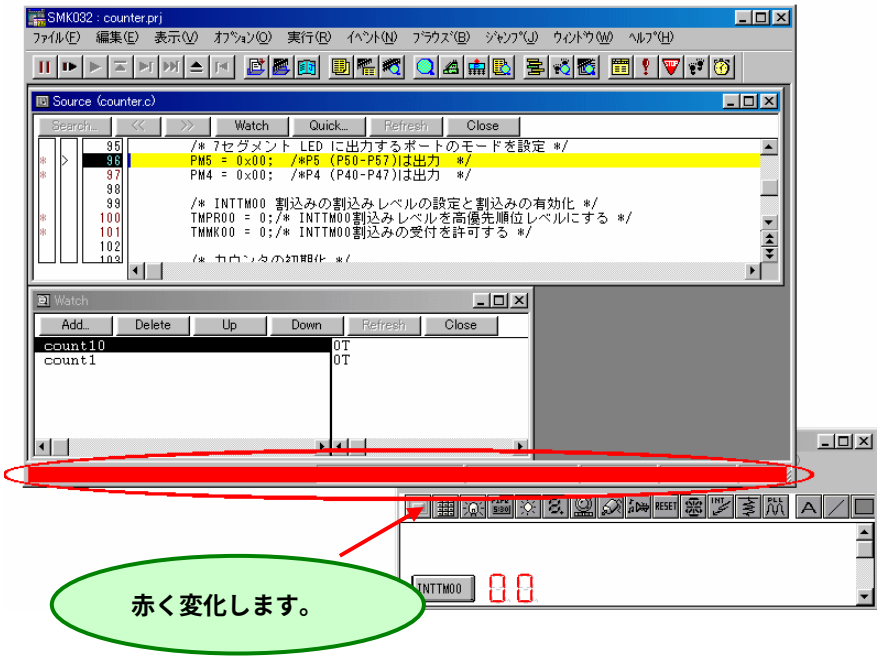
144 行目にブレークポイントが設定されました。

リスタートします。

SM78Kxx のリスタート・ボタン　，またはメニューの［実行(R)］→［リスタート(R)］を選択してください。

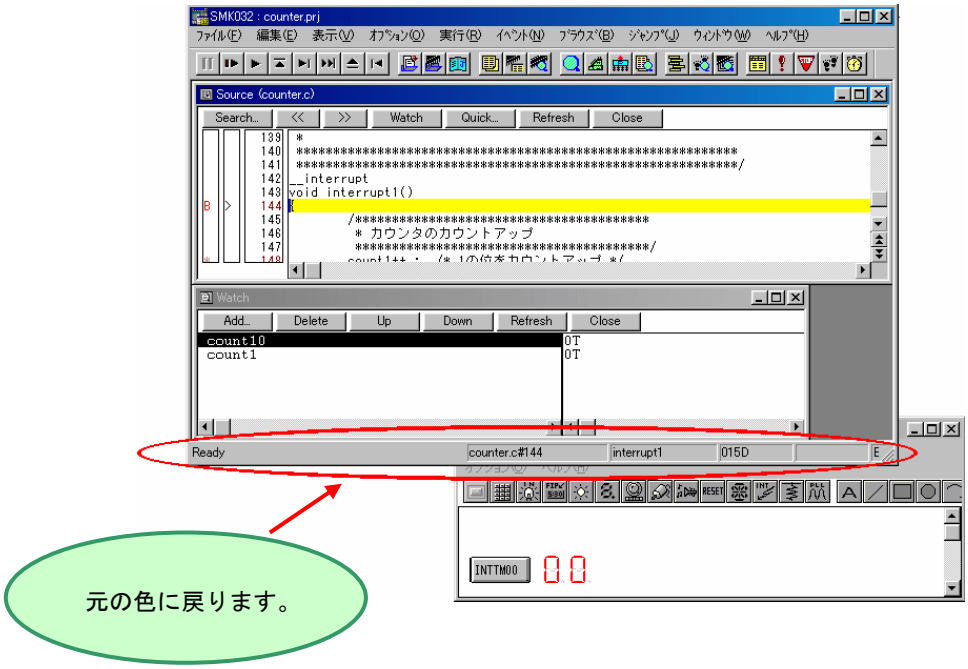


プログラムを実行します。



赤く変化します。

INTTM00 **ボタンを押してください。**

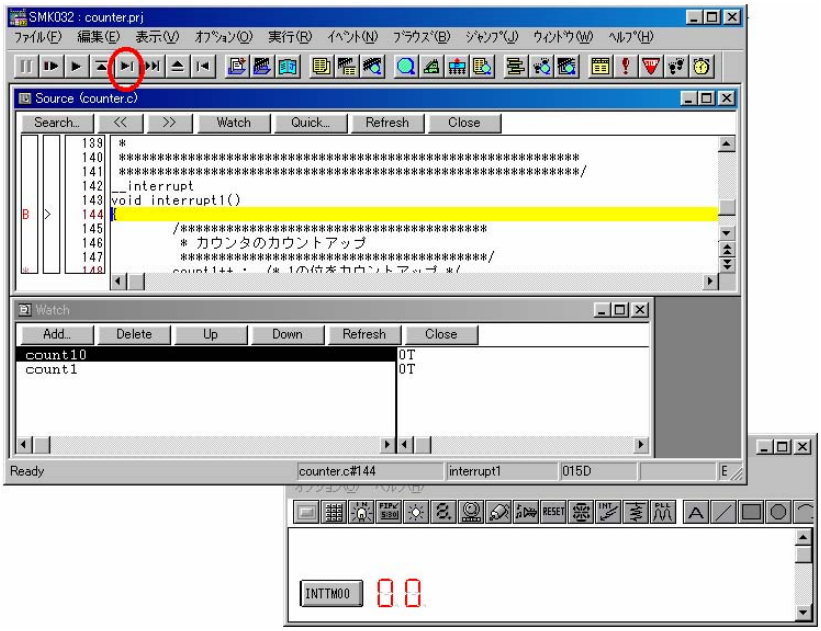


INTTM00 **ボタンを押すと，144 行目のブレークポイントでプログラムが停止します。**

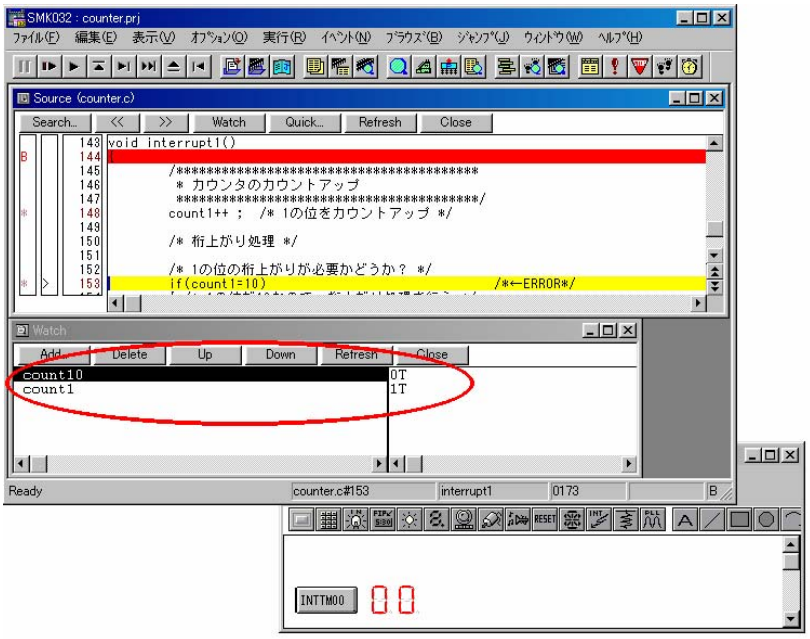
ここからはソース・プログラムを 1 行ずつ実行し，count1，count10 の値を確認してみましょう。

ステップ実行を行います。

SM78Kxx のステップイン・ボタン  ，またはメニューの［実行(R)］→［ステップイン(I)］を選択して 3 回選択してください。



プログラムを 3 回
ステップ実行します。

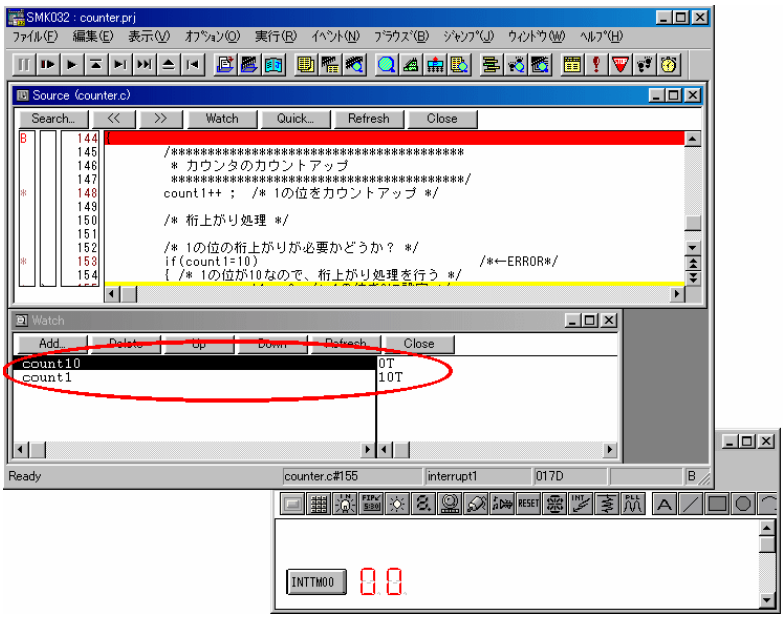


148 行目をステップ実行後，ウォッチ・ウインドウの count1 の値を確認してください。

count1 が 1 になっているので，正しく動作していることが確認できます。

続けて，153 行目をステップ実行します。

SM78Kxx のステップイン・ボタン  ，またはメニューの［実行(R)］→［ステップイン(T)］を選択してください。



153 行目をステップ実行すると，count1 の値が 1 から 10 に変化しました。

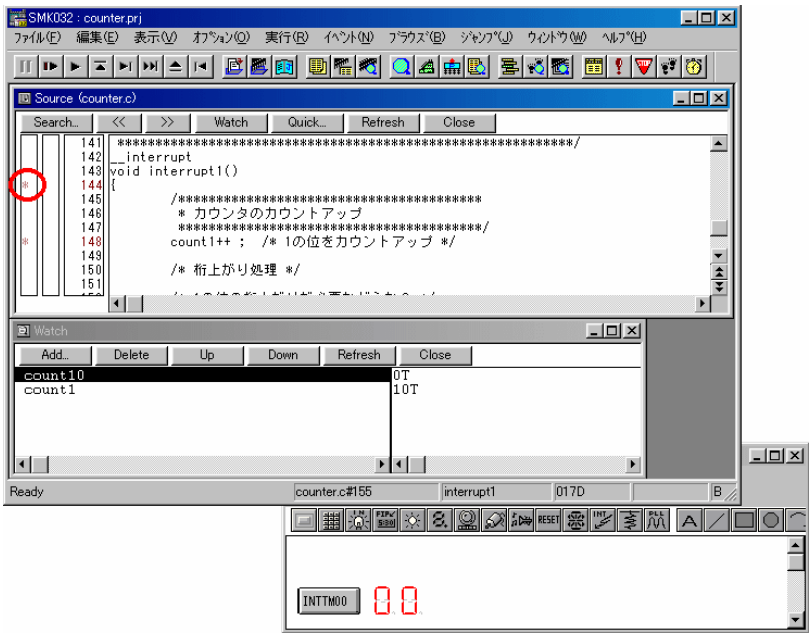
153 行目は，桁上がり処理が必要かどうかを判断する条件式であり，count1 の値の変化はないはずです。

これで，153 行目に問題があることがわかりました。

153 行目を確認すると，if 文の条件式で「count1 と 10 とを比較する式（count1==10）」を記述すべきところが，「count1 に 10 を代入する式（count1=10）」になっています。

エラー箇所が確認できましたので，ブレークを解除します。

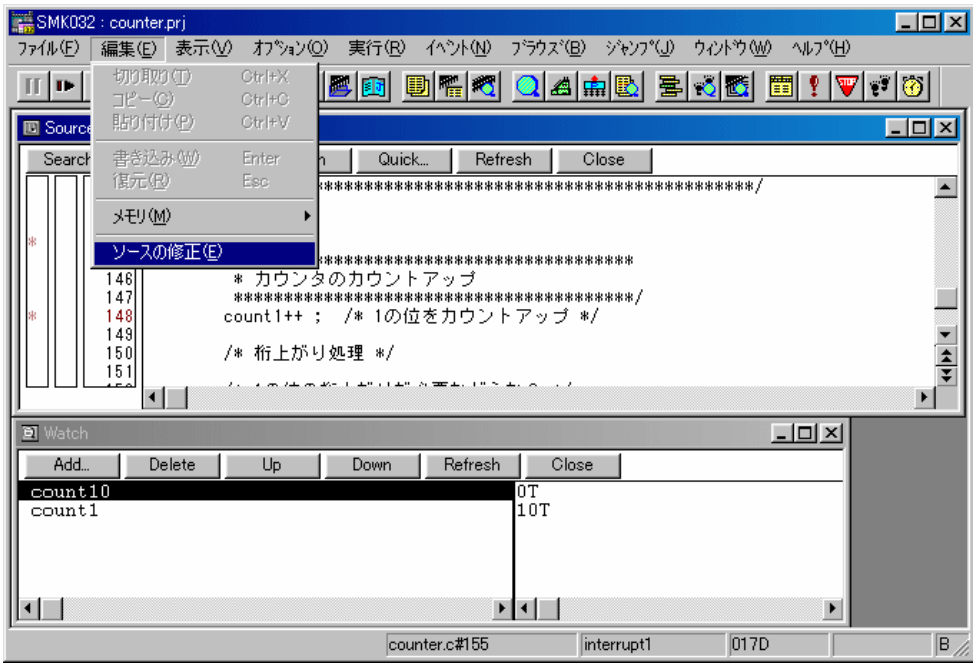
“ B ”をクリックすると“ * ”に戻り，ブレークポイントが解除されます。



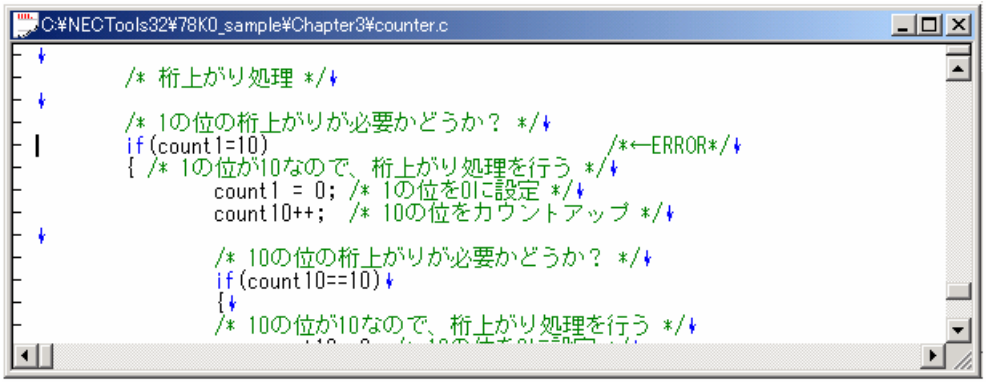
ソースの修正と実行形式の作成（2）

プログラムのエラー箇所を修正します。

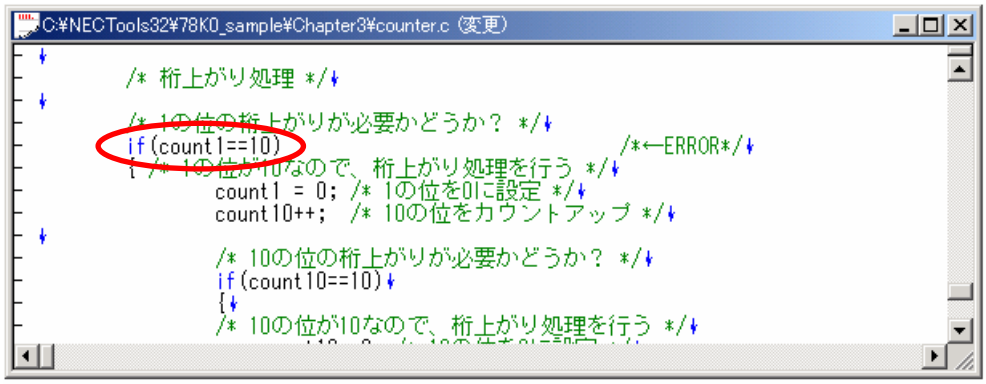
SM78Kxx **のメニューの［編集(E)］→［ソースの修正(E)］を選択してください。**



エディタが開きます。

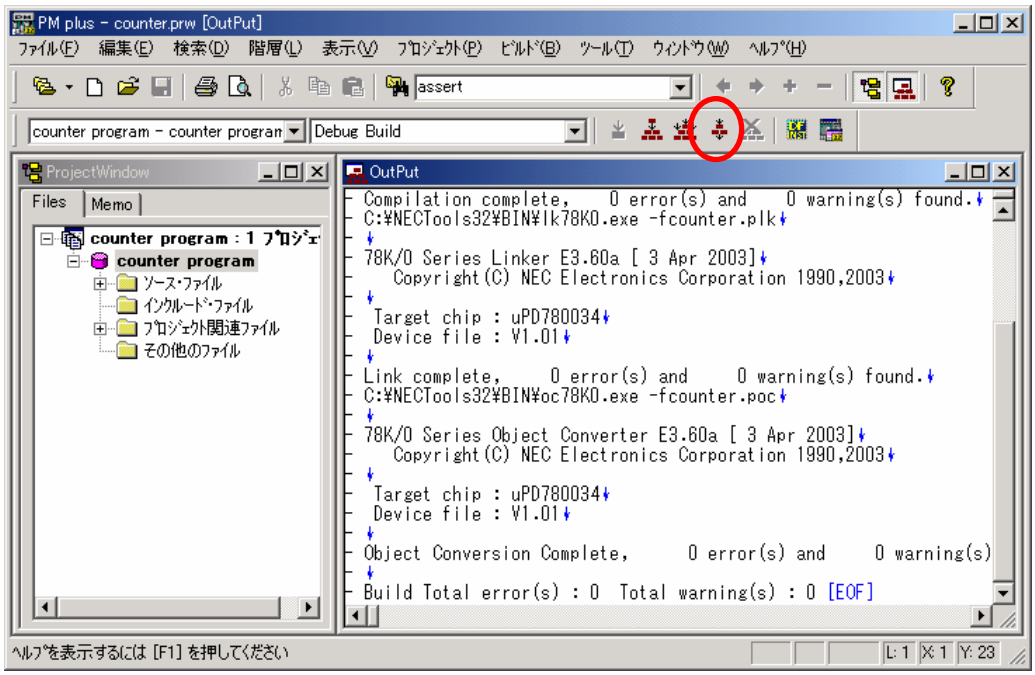


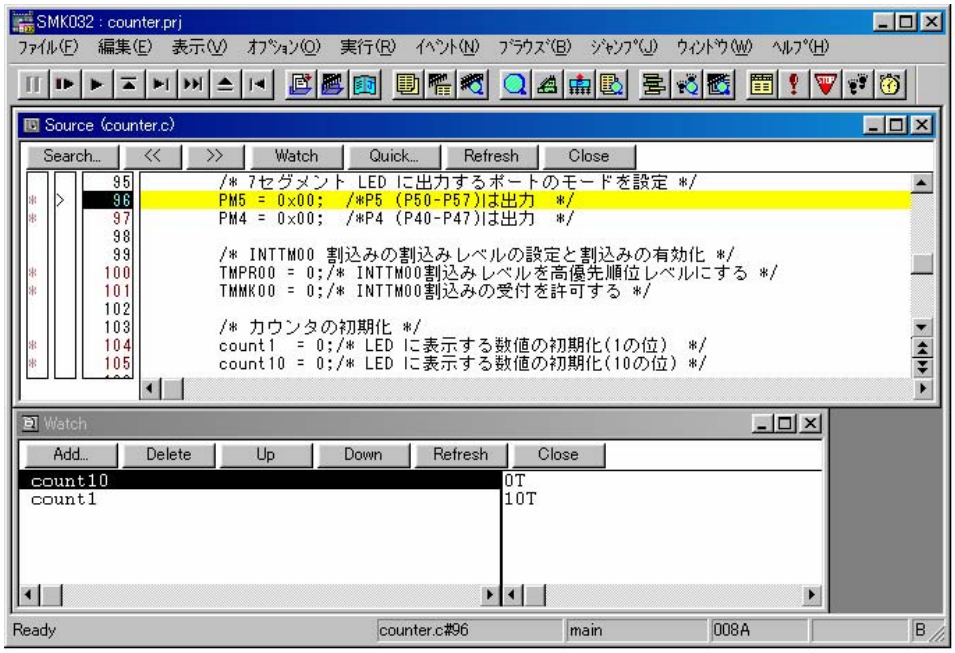
153 行目の if 文の条件式 “ count1=10 ” を “ count1==10 ” に修正してください。



修正が終了したら，PM plus のビルド->ディバグ・ボタン　  　，またはメニューの［ビルド(B)］→［ビルド->ディバグ(A)］を選択してください。

PM plus のエディタ機能の場合， ソースの修正内容はビルド時に自動的にセーブされます。



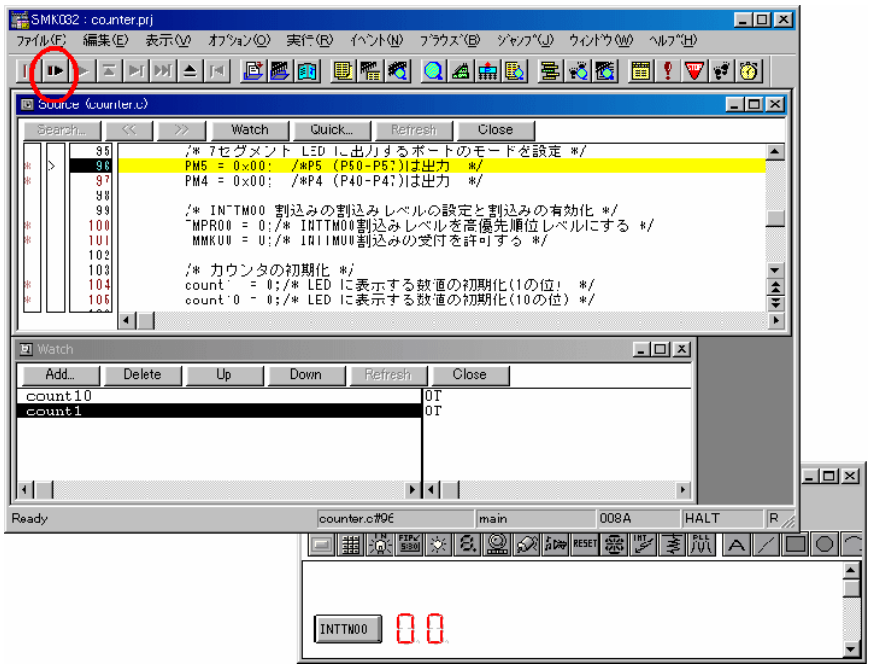


ビルドが終了すると，SM78Kxx **は自動的に実行形式ファイルのダウンロードを行います。**

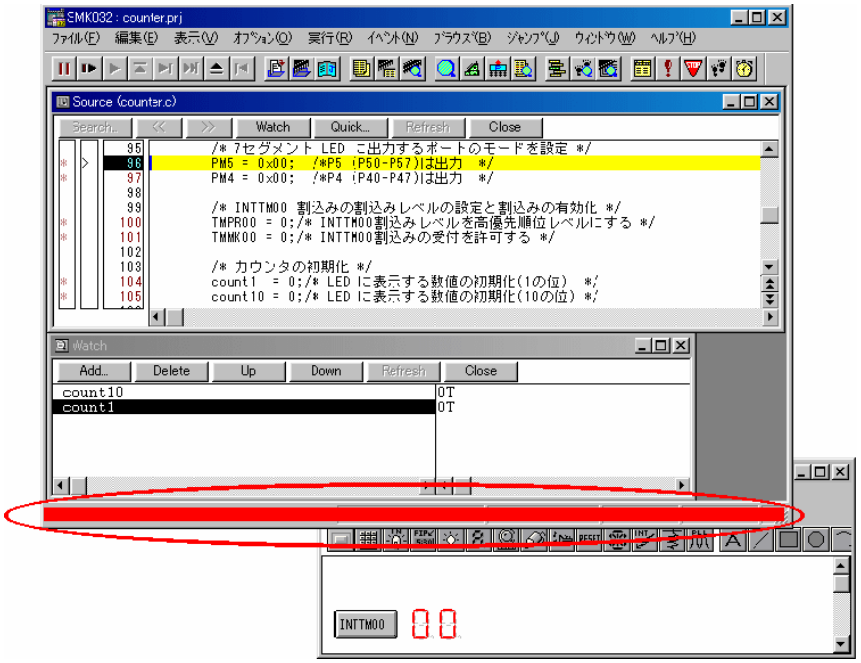
プログラムの実行（2）

リスタートします。

SM78Kxx **のリスタート・ボタン**  **，またはメニューの［実行(R)］→［リスタート(R)］を選択してください。**

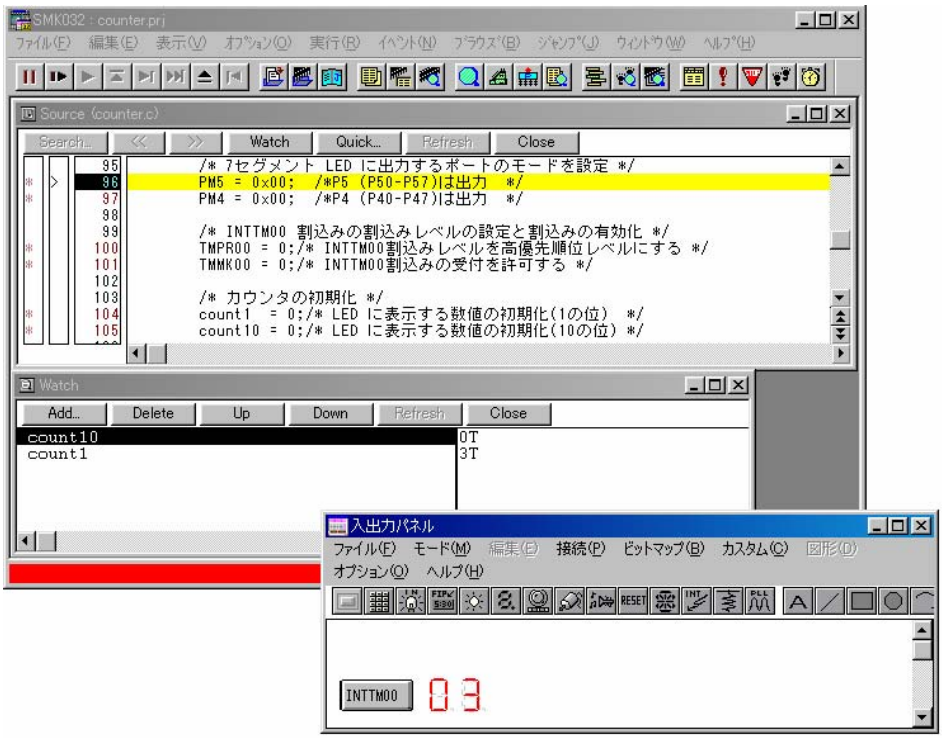


プログラムを実行します。



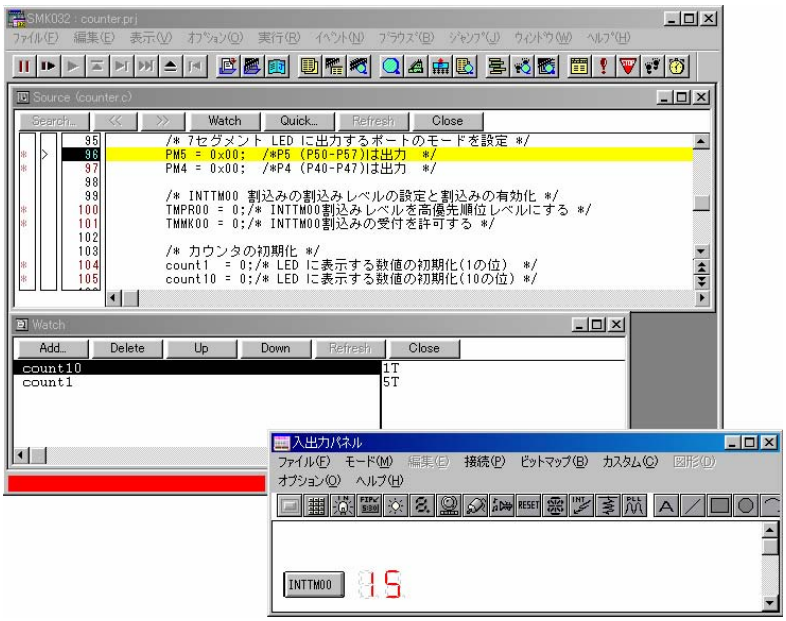
カウント・アップ動作が正常に行われるかを確認します。

まず，1 の位が正しくカウント・アップしているかを確認するためにボタンを数回押してください。



1 の位は，正常にカウント・アップ処理しています。

次に，桁上がり処理が正しく行われているかを確認するために，INTTM00 ボタンを 10 回以上押してください。



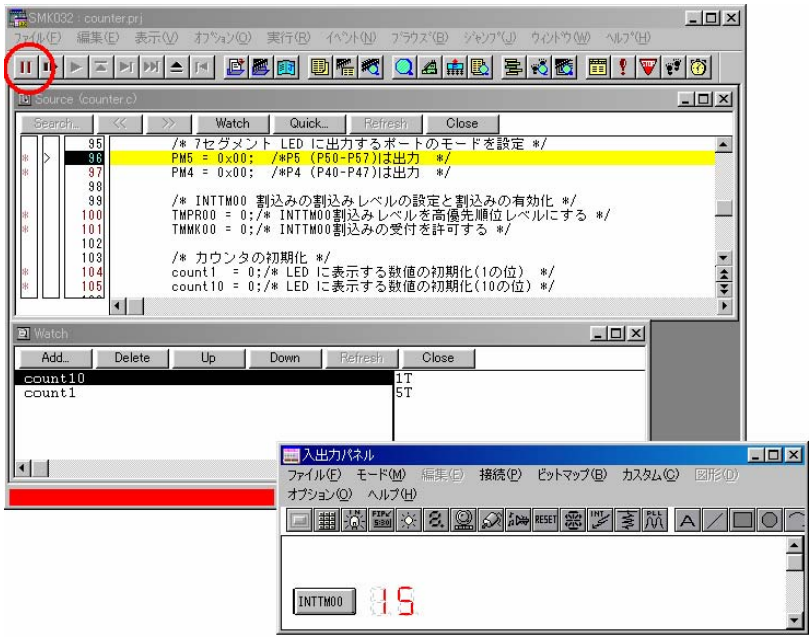
桁上がり処理は正常に行われています。

したがって，カウント・アップ動作は正常に行われていることが確認できました。

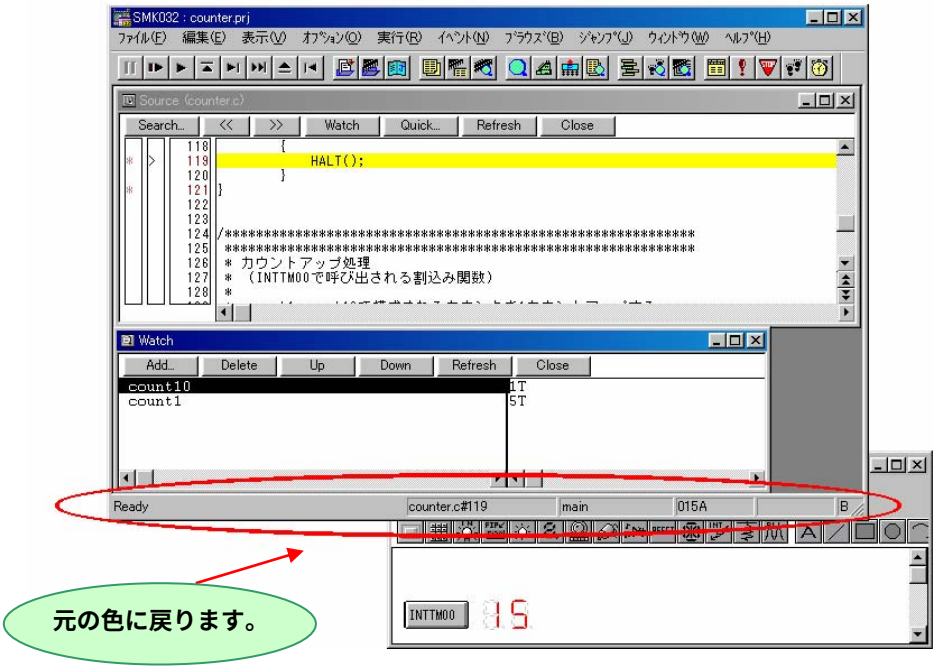
最後に，桁溢れ処理が正しく行われているかを確認します。

INTTM00 ボタンを 100 回以上押して確認することもできますが，ここでは，より簡単に確認する方法を紹介します。

まず，SM78Kxx の停止ボタン ，またはメニューの［実行(R)］→［ストップ(S)］を選択し，プログラムを停止してください。



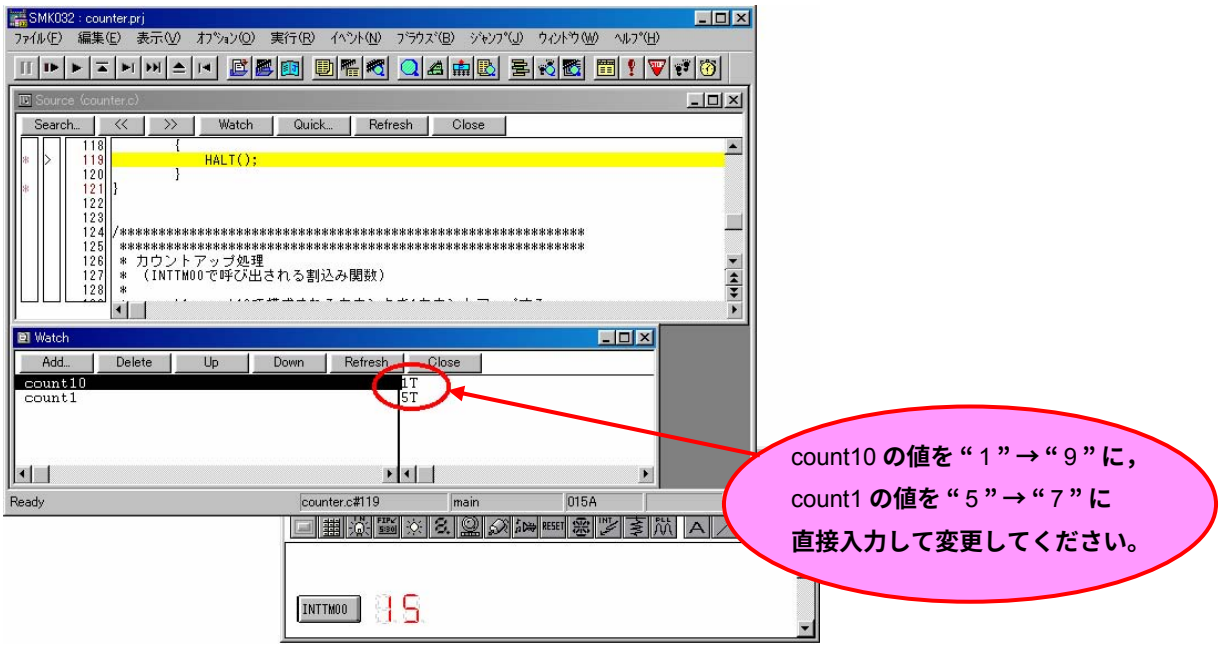
プログラムを停止します。

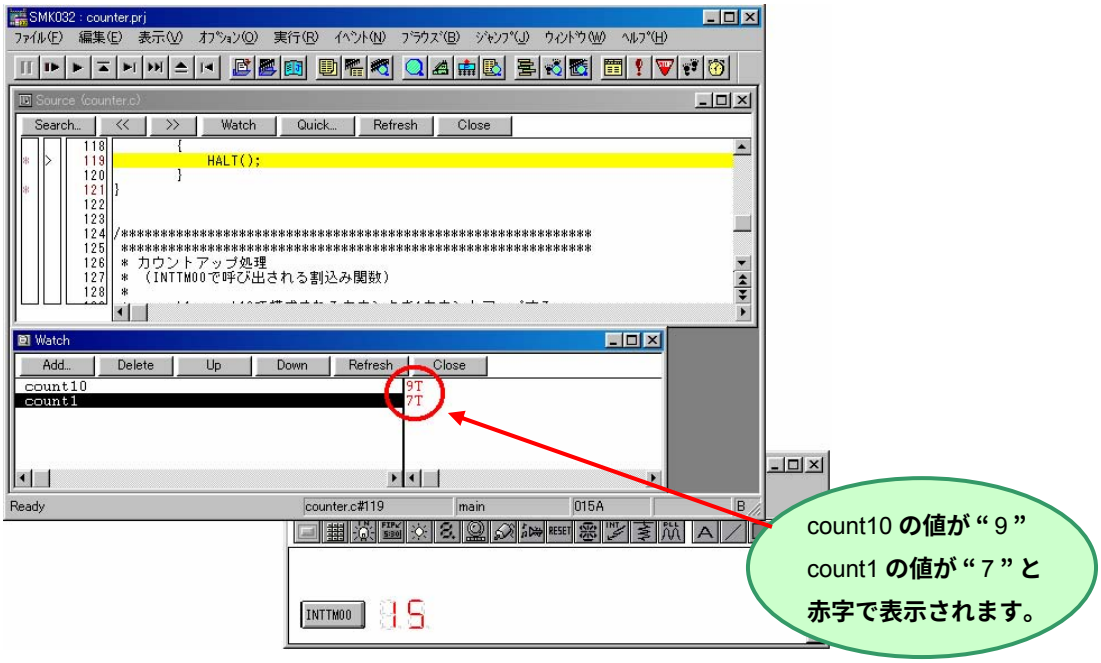


元の色に戻ります。

次に，ウォッチ・ウインドウのデータ値表示／設定エリアの値を変更します。

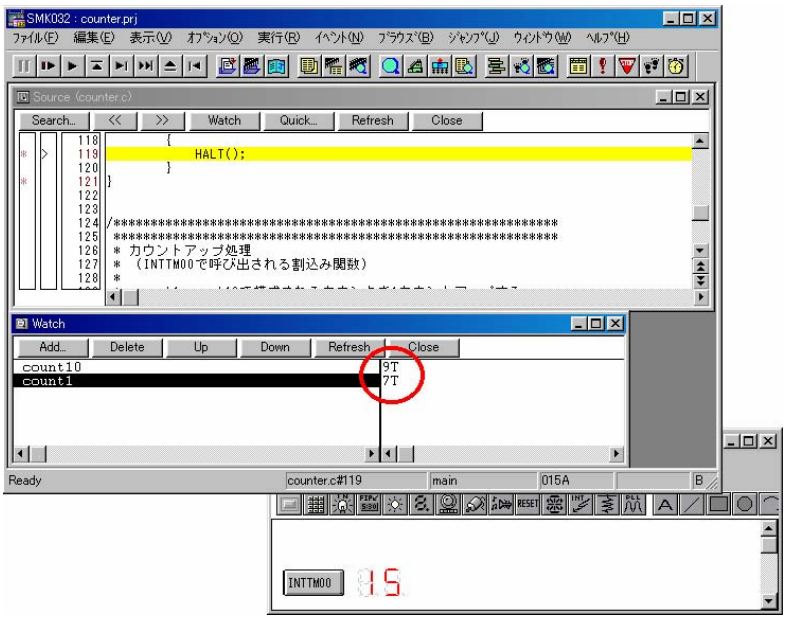
ウォッチ・ウインドウのデータ値表示／設定エリアにカーソルを合わせ，count10 の値を“ 9 ”に，count1 の値を“ 7 ”に変更します。





その状態で，リターン・キーを押してください。

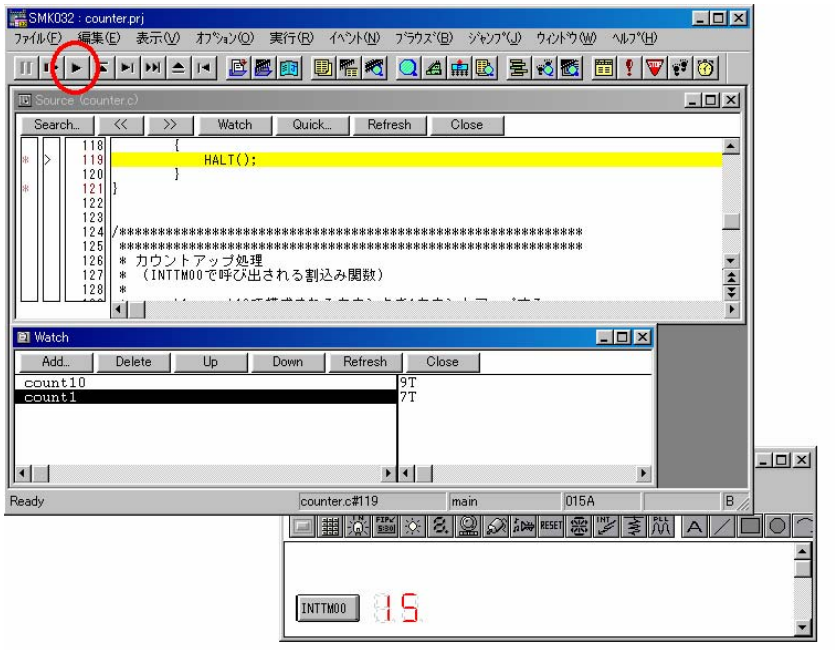
count10，count1 **の値が，赤字から黒字に変わります。**

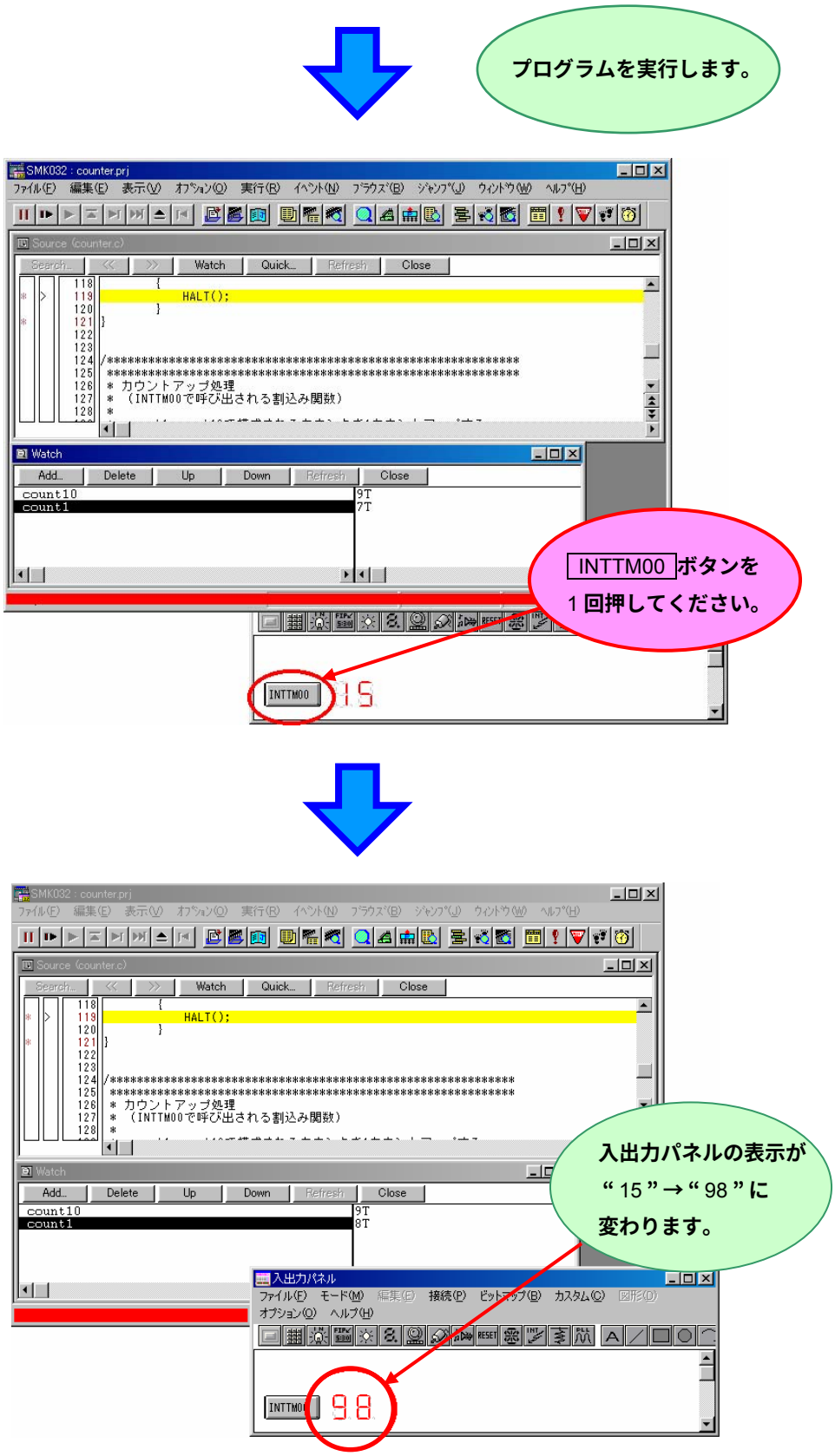


これで，count10 **に“　9　”，**count1 **に“　7　”の値が設定されました。**

それでは，この状態から継続して実行してみましょう。

SM78Kxx **のスタート・ボタン**　▶　**，またはメニューの［実行(R)］→［継続して実行(G)］を選択してください。**

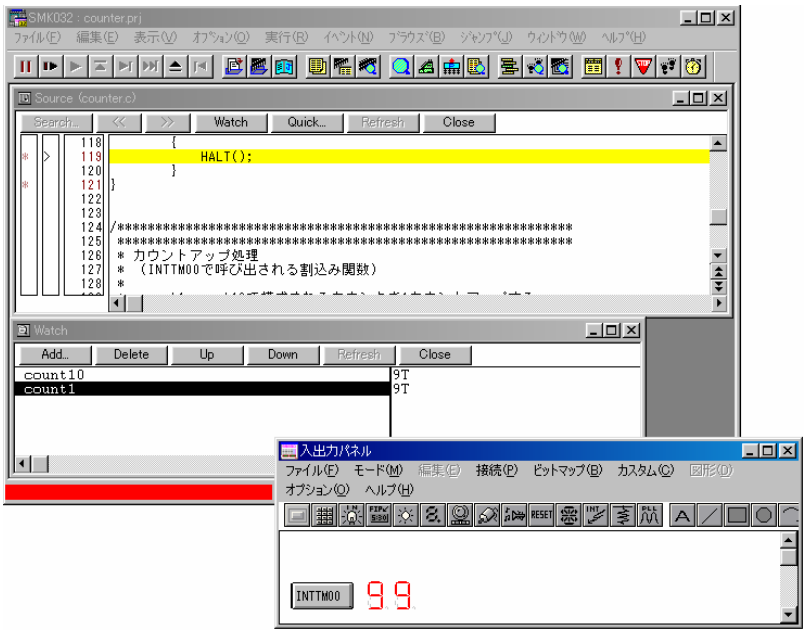




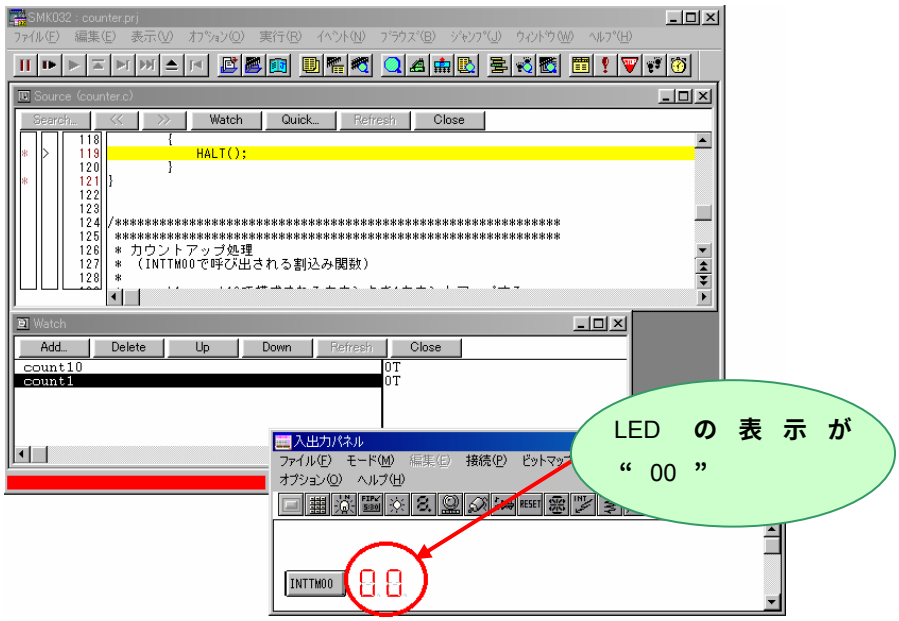
これで，INTTM00 ボタンを 98 回押したときと同じ状態になりました。

それでは，さらに INTTM00 ボタンを押してください。

INTTM00 ボタンを
1 回押してください。



INTTM00 ボタンを
1 回押してください。

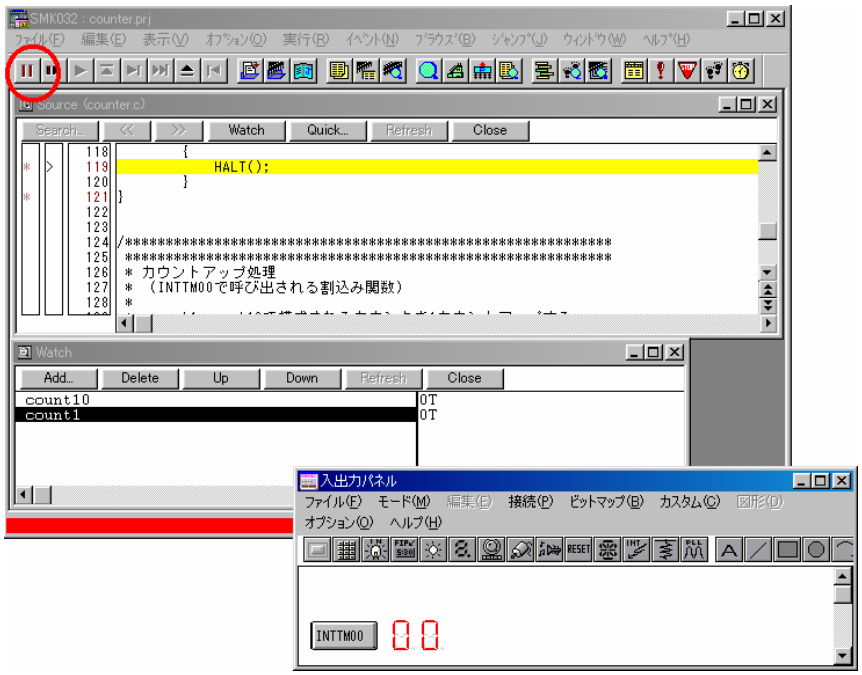


桁溢れ処理は，正常に行われています。

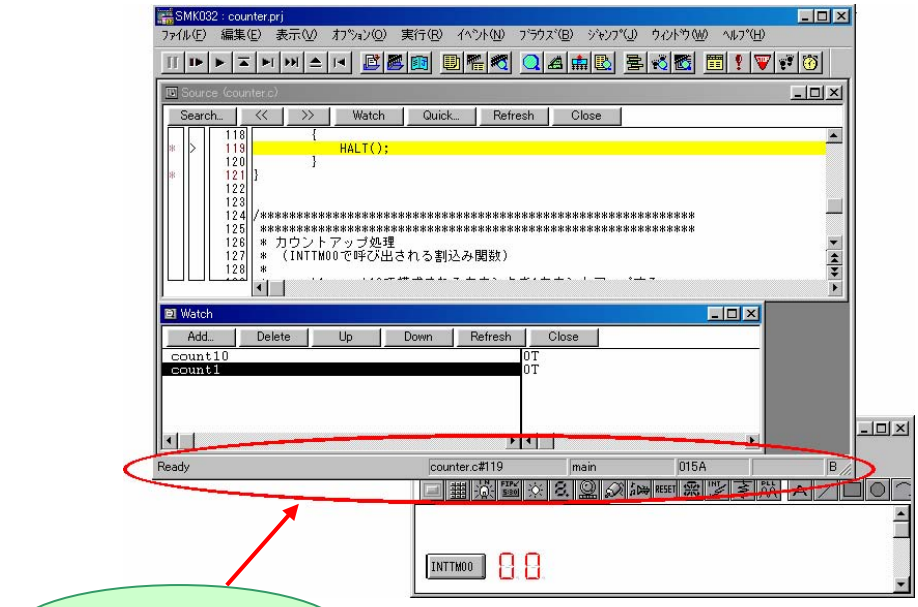
これで，プログラムのすべての確認が終了しました。

プログラムを停止します。

SM78Kxx の停止ボタン  **，またはメニューの［実行(R)］→［ストップ(S)］を選択してください。**



プログラムを停止します。

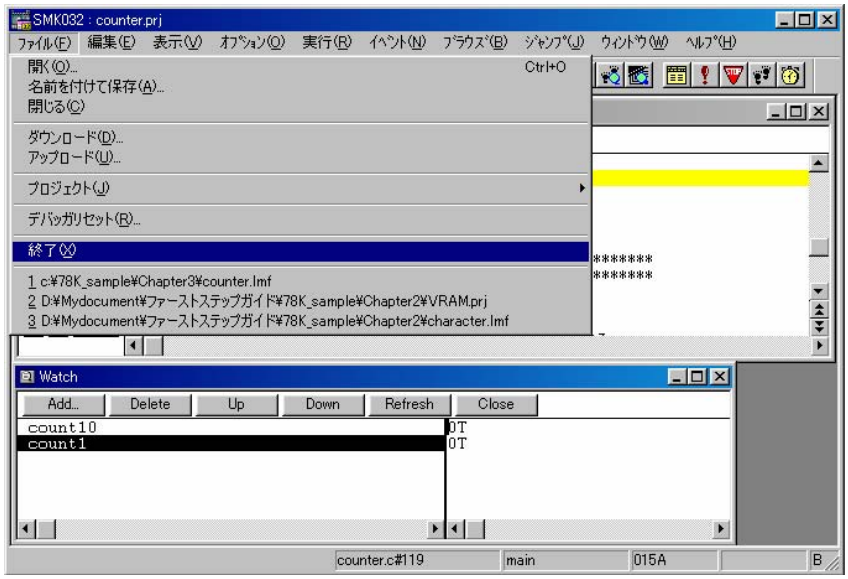


元の色に戻ります。

終　了

SM78Kxx を終了します。

SM78Kxx メニューの［ファイル(F)］→［終了(X)］を選択してください。



終了確認ダイアログが表示されます。

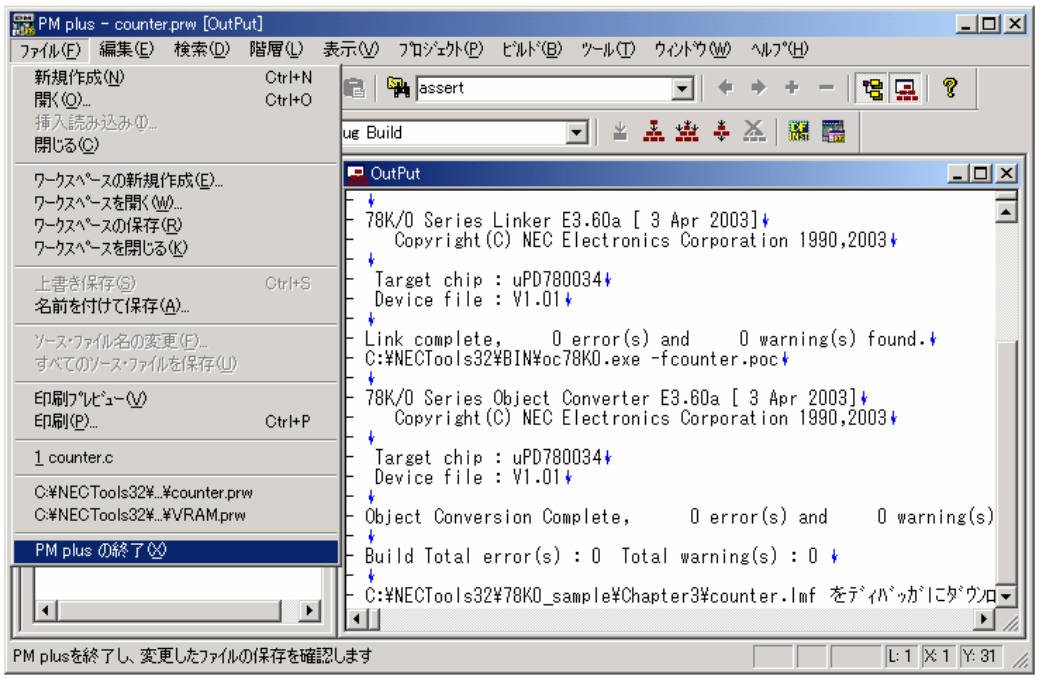
この章で設定した入出力パネルの状態などをプロジェクト・ファイルに保存する場合は［はい(Y)］を，保存しない場合は［いいえ(N)］を押してください。

「環境」とは，外部部品，ウインドウの状態などのことです。

詳細については，SM78K シリーズ システム・シミュレータ Ver.2.52 ユーザーズ・マニュアル 操作編（U16768J）を参照してください。

PM plus を終了します。

PM plus のメニューの［ファイル(F)］→［PM plus の終了(X)］を選択してください。



PM plus のプロジェクト情報は逐次保存されているため，保存の確認はされません。

第 4 章　プログラミング編

この章では，78K シリーズ CPU 用の機種依存処理を C 言語で記述するための方法について，サンプル・プログラムを使用しながら解説します。

ここでは，サンプル・プログラムとして，スロット・プログラムを使用します。

スロット・プログラムでは，拡張 C 言語仕様の「レジスタ名を用いた特殊機能レジスタへのアクセス」，「割り込み／例外関数の記述」，および「部分的な割り込み禁止の設定」の機能を使用しています。

この章の全体の流れを，次に示します。

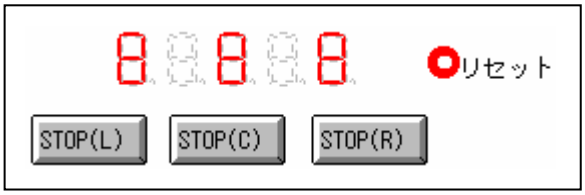
スロット・プログラムの仕様

この章の操作を行う前に，スロット・プログラムの概要を理解していただく必要があります。

外部仕様は次のとおりです。

外部仕様

- ある装置に，5 桁の 7 セグメント LED，3 つの四角いボタン，リセット・ボタンがあります。



- LED は 1 つおきに使用し，0 から 9 までの数字を連続で表示し，9 を表示した次は 0 に戻ります。
- 四角いボタンの表面には，それぞれ“ STOP(L) ”, “ STOP(C) ”, “ STOP(R) ”と書かれており，このボタンを押すと連続して数字を表示している LED の動きが止まります（ある数字を表示したままになります）。
  - STOP(L) ボタンを押すと，LED の左端の桁の数字が停止します。
  - STOP(C) ボタンを押すと，LED の中央の桁の数字が停止します。
  - STOP(R) ボタンを押すと，LED の右端の桁の数字が停止します。
- リセット・ボタンを押すと初期状態に戻り，止まっていた LED が動き出します。

基本仕様は次のとおりです。

基本仕様

- スロット表示
  - 0～9 までの数値をカウント・アップ表示します。
  - 9 を表示したら，0 に戻ります。
- ボタンの動作
  - STOP(L) ボタンを押すと，INTP0 割り込みが発生します。
  - STOP(C) ボタンを押すと，INTP1 割り込みが発生します。
  - STOP(R) ボタンを押すと，INTP2 割り込みが発生します。
- 割り込み関数の設定
  - INTP0 割り込みが発生したら，関数 stop_btn_Left を実行します。
  - INTP1 割り込みが発生したら，関数 stop_btn_Center を実行します。
  - INTP2 割り込みが発生したら，関数 stop_btn_Right を実行します。
- 割り込み関数の処理
  - 関数 stop_btn_Left を実行すると，左端の桁の表示を固定します。
  - 関数 stop_btn_Center を実行すると，中央の桁の表示を固定します。
  - 関数 stop_btn_Right を実行すると，右端の桁の表示を固定します。
- ターゲット CPU 環境の初期化
  - 使用するポートの初期化を行います。
  - 割り込みの使用を可能にし，割り込みの優先順位の設定を行います。

内部仕様は次のとおりです。

内部仕様

- LED を点灯する位置と表示する数字は，次の変数で保持します。

変数	内容
unsigned char place;	表示(点灯)位置の指定
unsigned char num_data[10];	表示用数字データ

- プログラムは，メイン関数，ターゲット CPU 環境の初期化部，スロットの表示部，および割り込み関数で構成します。

ファイル名	処理関数	処理内容
slot.c	メイン関数 void main();	・ターゲット CPU 環境の初期化関数（init_target()）の呼び出し ・スロット表示関数（slot()）の呼び出し
	ターゲット CPU 環境の初期化 void init_target(void);	・ポート，割り込みレベルなど，ターゲット CPU 環境の初期化 ・割り込みを受け取り可能状態に設定
	スロット表示 void slot(void);	・表示用数字（0~9）をループさせ LED に表示 ・ループ中での割り込みを許可
interrupt_func.h（関数宣言） interrupt_func.c（関数定義）	STOP(L)ボタン処理 __interrupt void stp_btn_Left(void);	・INTP0 割り込みで起動 ・左端の桁の表示を現在の数値のまま固定
	STOP(C)ボタン処理 __interrupt void stp_btn_Center(void);	・INTP1 割り込みで起動 ・中央の桁の表示を現在の数値のまま固定
	STOP(R)ボタン処理 __interrupt void stp_btn_Right(void);	・INTP2 割り込みで起動 ・右端の桁の表示を現在の数値のまま固定

- 使用する割り込みは次の 3 つです。
  - INTP0
  - INTP1
  - INTP2

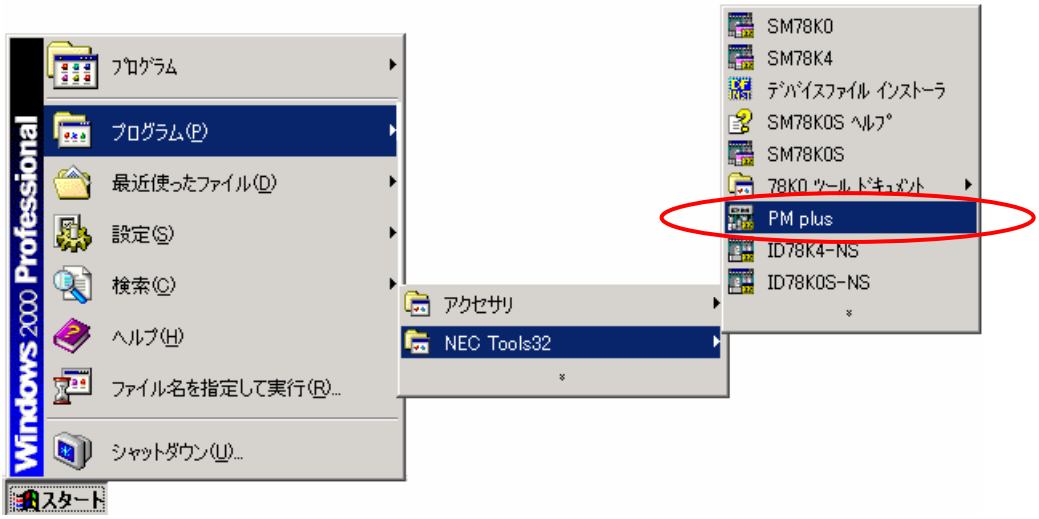
- LED の制御と割り込みの入力には，入出力ポートとして 78K0 は P0, P5, P4, 78K0S は P2, P1, P4, 78K4 は P2, P5, P4 を使用します。

	78K0	78K0S	78K4	
➢	P0	P2	P2	（割り込みの入力用）
➢	P5	P1	P5	（7 セグメント LED 点灯用）
➢	P4	P4	P4	（7 セグメント LED 桁指定用）

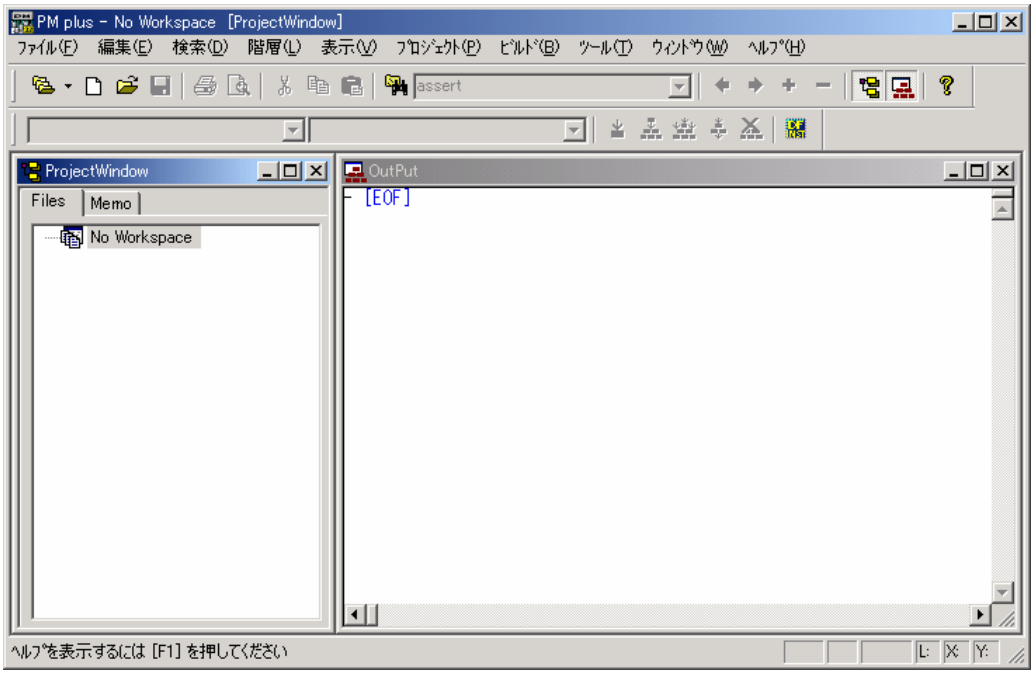
スロット・プログラムの動作確認

スロット・プログラムの動作を確認するために，まず，PM plus **を起動します。**

Windows **スタート・メニューの［プログラム**(P)**］→［**NEC Tools32**］→［**PM plus**］を選択してください。**



PM plus **が**
起動します。

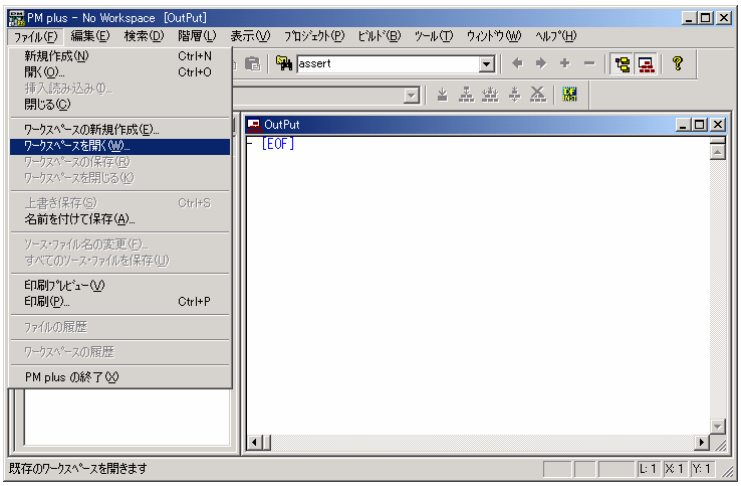


ワークスペース・ファイルの読み込み

この章では，あらかじめ作成されているワークスペース・ファイルを使用します。

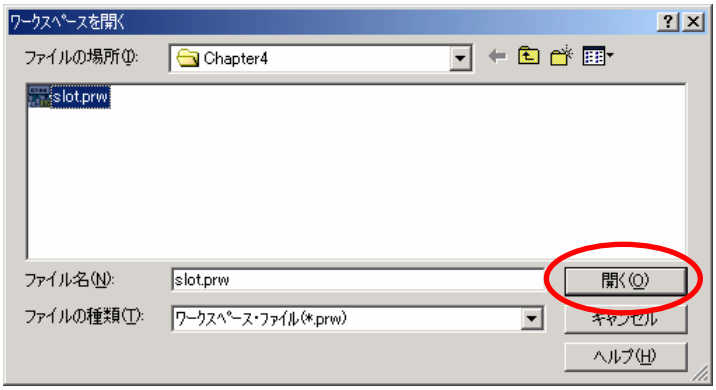
PM plus のメニューの［ファイル(F)］→［ワークスペースを開く(W)...］を選択し，“ slot.prw ”を指定してください。

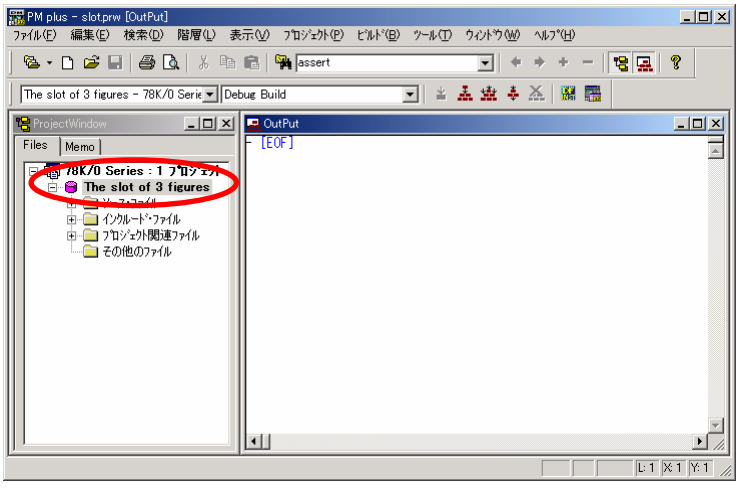
環境を設定していない方は「サンプル環境」をご覧ください。



Chapter4 ディレクトリを開いてください。

“ slot.prw ”を指定し，開く(O) ボタンを押します。



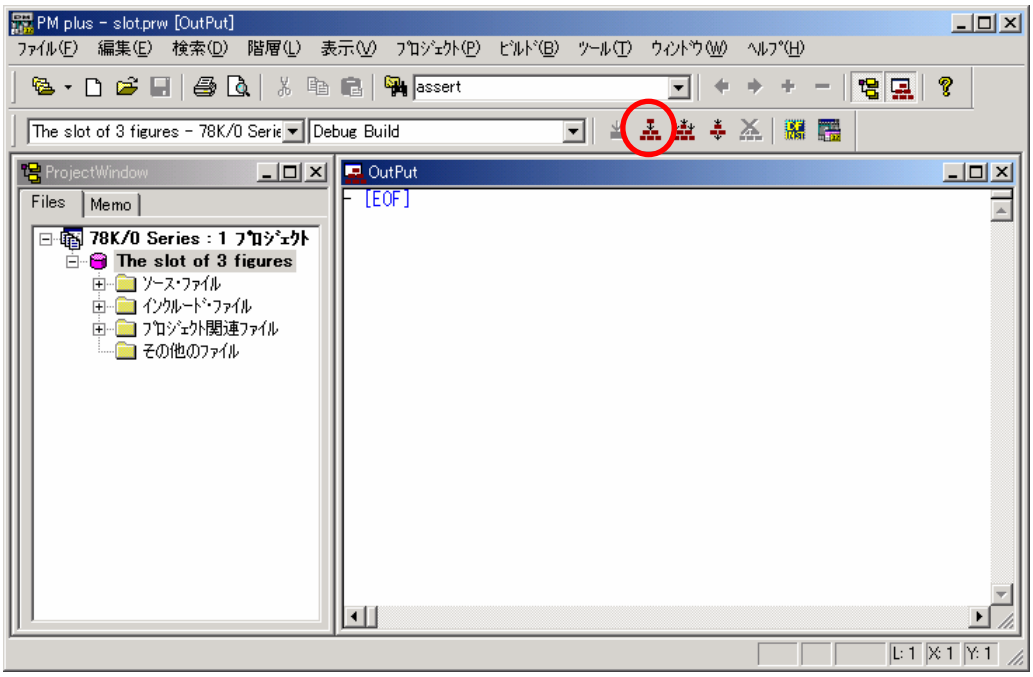


ワークスペース・ファイル“ slot.prw ”を読み込みます。

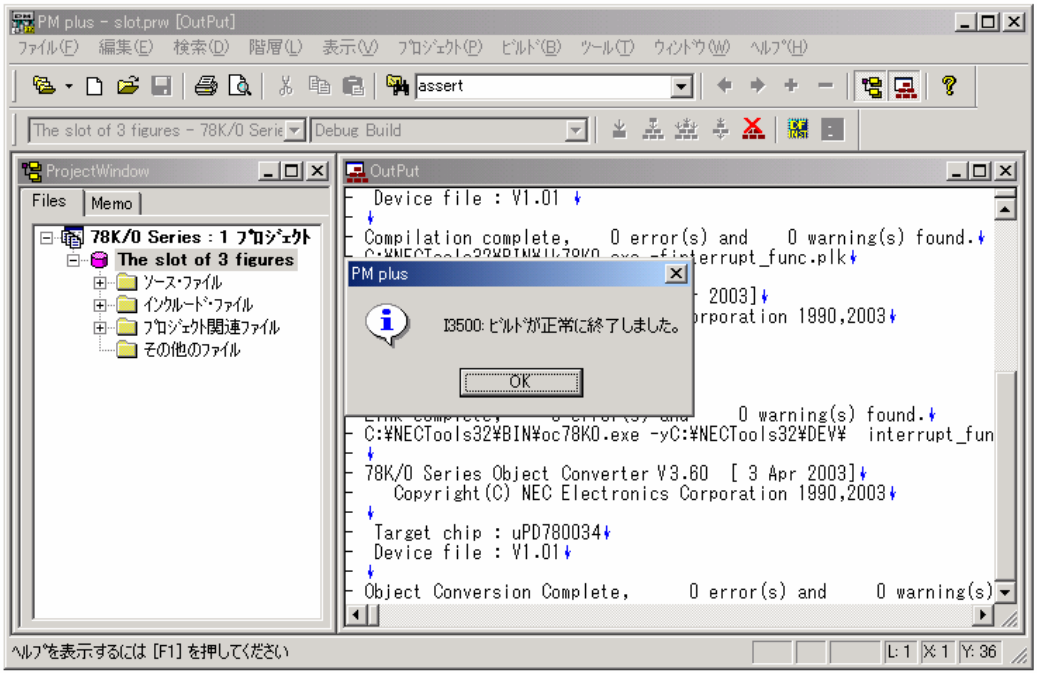
実行形式の作成

プロジェクトの実行形式を作成します。

PM plus **のビルド・ボタン**  **，またはメニューの［ビルド(B)］→［ビルド(B)］を選択してください。**



ビルド処理を実行します。

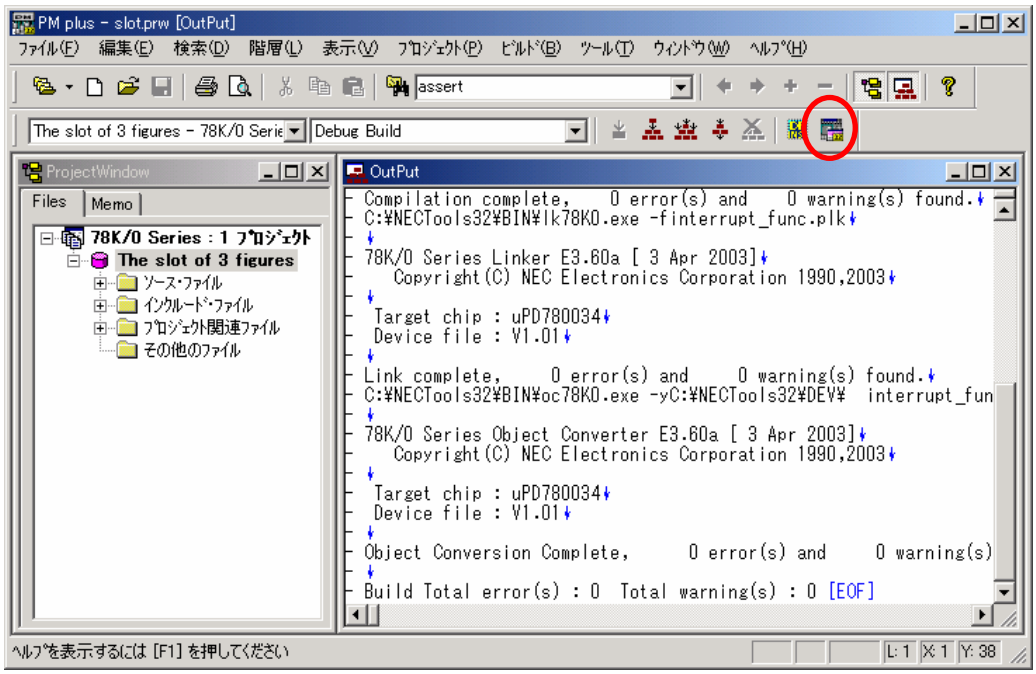


ビルド処理を正常に終了し，実行形式が作成されました。

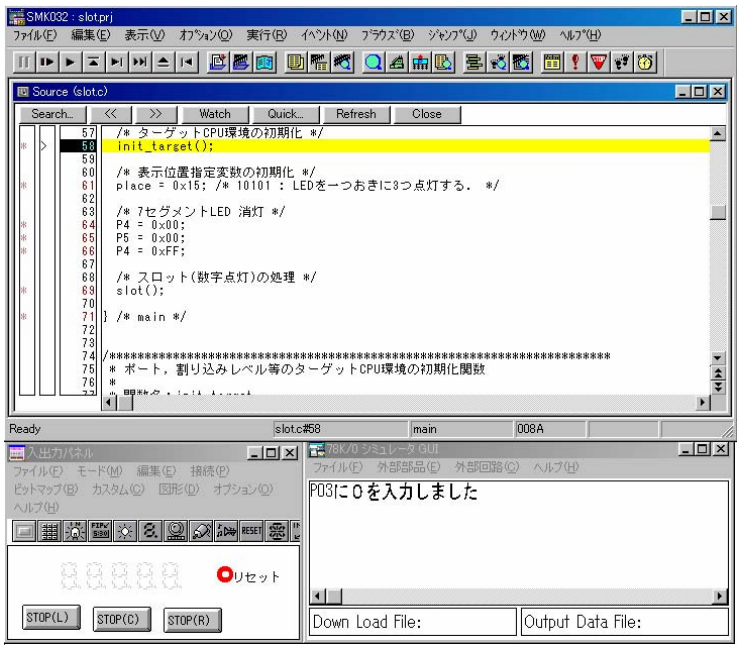
システム・シミュレータ（SM78Kxx）の起動

SM78Kxx を起動します。

PM plus のディバグ・ボタン  ，またはメニューの［ビルド(B)］→［ディバグ(D)］を選択してください。



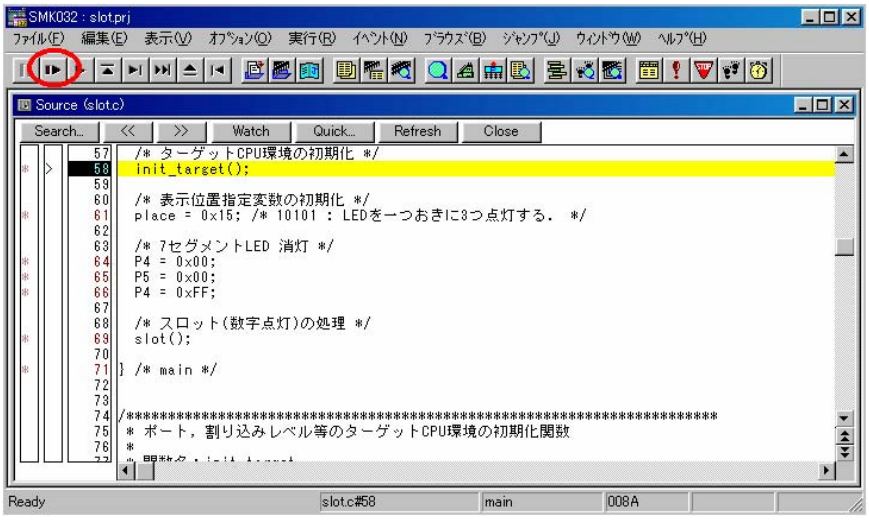
SM78Kxx が起動します。



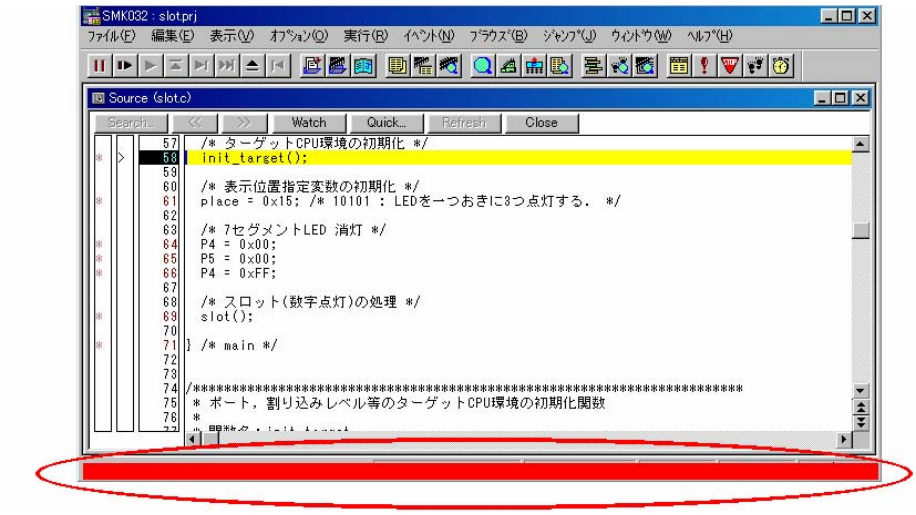
プログラムの実行

スロット・プログラムを実行します。

SM78Kxx のリスタート・ボタン　，またはメニューの［実行(R)］→［リスタート(R)］を選択してください。



プログラムを実行します。



プログラムの実行中は，ステータス表示エリアが赤く変化します。

プログラムの実行中，入出力パネルの LED の各桁は，数字（0～9）をカウント・アップ表示する動作を繰り返しています。

では，この実行中のスロット・プログラムを操作してみましょう。
入出力パネルの各ボタンを操作して，LED 表示の変化を確認してください。

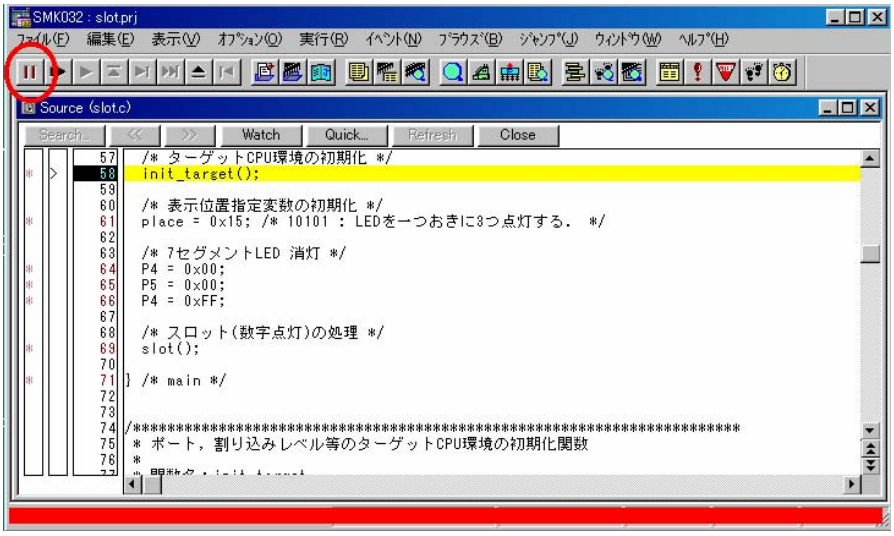
STOP(L) ボタンを押すと，LED の左端の桁の数字が停止します。
STOP(C) ボタンを押すと，LED の中央の桁の数字が停止します。
STOP(R) ボタンを押すと，LED の右端の桁の数字が停止します。
○リセット ボタンを押すと，初期状態に戻り，LED の各桁はカウントアップを開始します。

これで，スロット・プログラムの動作確認を終了します。

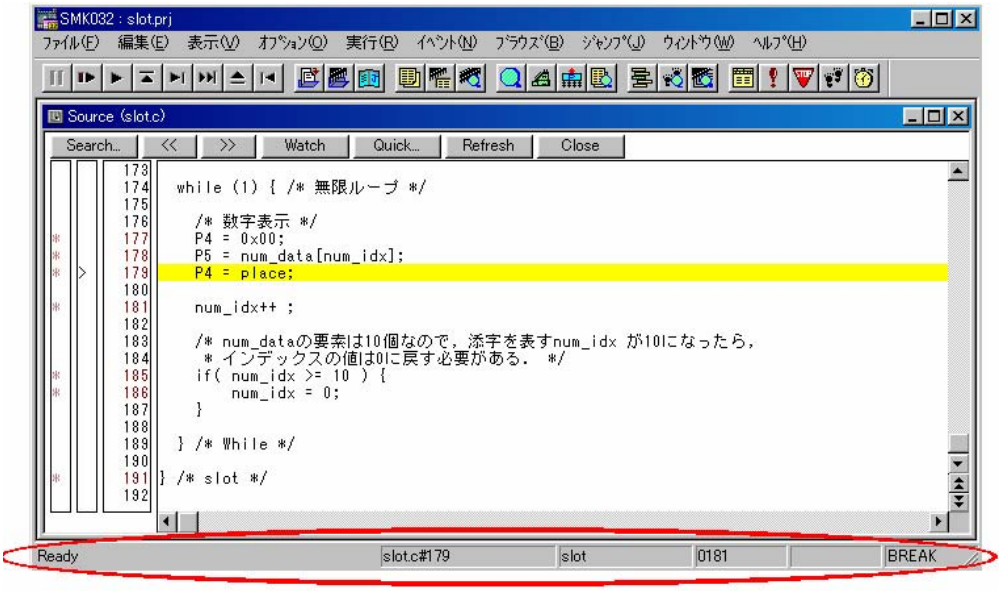
プログラムの停止

プログラムを停止します。

SM78Kxx の停止ボタン ，またはメニューの［実行(R)］→［ストップ(S)］を選択してください。



プログラムを停止します。

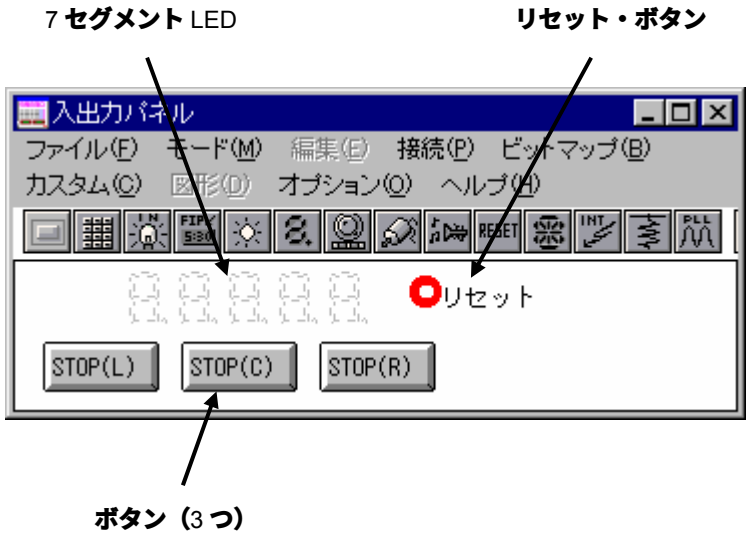


プログラムを停止すると，ステータス表示エリアの色が元に戻ります。

入出力パネルの解説

スロット・プログラムで使用する入出力パネルには，「7 セグメント LED」，「ボタン（3 つ）」，「リセット・ボタン」が設定されています。

それぞれの設定について解説します。



まず，7 セグメント LED 端子の設定について解説します。

入出力パネル・ウインドウの 7 セグメント LED 端子設定ボタン  ，またはメニューの［接続(P)］→［7 セグメント LED(S)...］を選択してください。

7 セグメント LED 端子設定ダイアログが開きます。

7 セグメント LED 端子設定ダイアログが 開きます。

78K0 と 78K4 は入出力ポート P5, P4, 78K0S は P1, P4 の各ビットを，7 セグメント LED 端子に接続しています。

スロット・プログラムでは，LED は 5 桁使用するため，桁信号 1 から桁信号 5 まで設定しています。

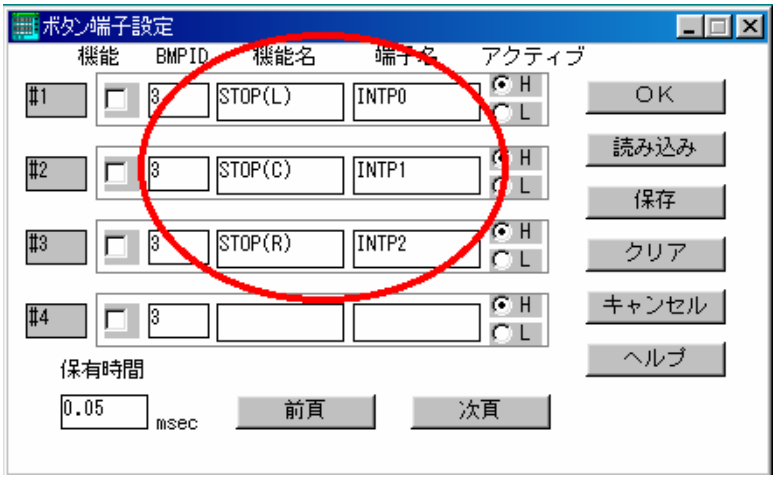
7 セグメント LED 端子設定の詳細については，「第 3 章　入門編」をご覧ください。

次に，ボタンの設定について解説します。

入出力パネル・ウインドウのボタン  ，またはメニューの［接続(P)］→［ボタン(B)...］を選択してください。

ボタン端子設定ダイアログが開きます。

ボタン端子設定ダイアログが開きます。



ボタン端子設定ダイアログでは，入出力パネル・ウインドウに表示するボタンの接続情報を設定します。

ボタンの接続は，任意の端子に対して可能で，表示されたボタンを押すことで入力値を与えることができます。

この章では，ボタンを外部割り込み端子（INTP0 ～ 2）に接続しています。

内部割り込みボタンは，ボタンを押すことで内部割り込みを発生させますが，ここではボタンを端子に接続し，端子の Low-High 状態を認識して，割り込みのトリガにしています。

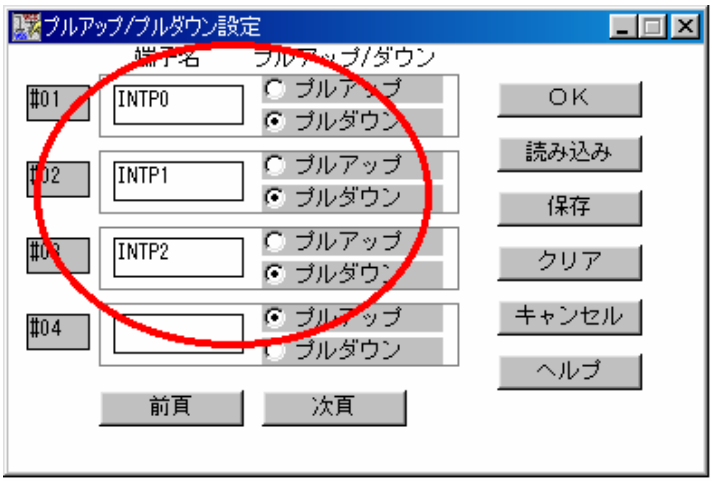
詳細については，「スロット・プログラムの仕様」をご覧ください。

ボタンを設定する場合，あわせて[プルアップ／プルダウンの設定]が必要です。

入出力パネル・ウインドウのボタン  ，またはメニューの［接続(P)］→［プルアップ／プルダウン設定(W)...］を選択してください。

プルアップ／プルダウン設定ダイアログが開きます。

プルアップ／プルダウン設定
ダイアログ が開きます。



この章では，ボタンを接続した外部割り込み端子（INTP0 ～ 2）をプルダウンに設定しています。

プルアップ／プルダウンの設定とは？

SM78Kxx の外部部品の一部には，非動作時の出力が未定義なものがあります。

ボタンもその一つで，このような外部部品では，プルアップ／プルダウン設定により，非動作時の端子状態を設定する必要があります。

なお，プルアップ／プルダウン設定は，ボタンなどの外部部品の設定の前に行う必要があります。

詳細については，ユーザーズ・マニュアル「SM78K シリーズ システム・シミュレータ 操作編」を参照してください。

次に，リセット・ボタンの設定について解説します。

入出力パネル・ウインドウのボタン　  　，またはメニューの［接続(P)］→［リセットボタン(R)］を選択すると，リセット・ボタンは初期状態の位置に移動します。

リセット・ボタンが，初期状態の位置（左上隅）に移動します。

リセット・ボタンの位置は，［モード(M)］→［配置(E)］を選択後，リセット・ボタンをドラッグ＆ドロップすることにより，自由に変更できます。

位置の変更後は，［モード(M)］→［実行(S)］を選択し，実行モードに戻してください。

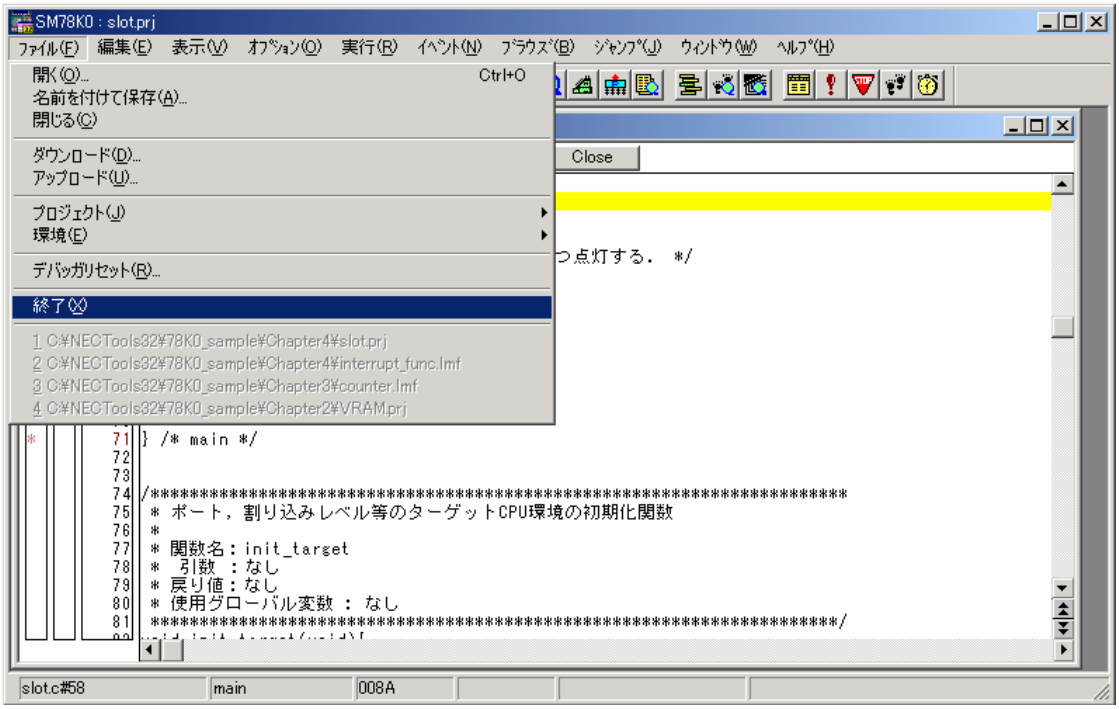
シミュレーション中にリセット・ボタンを押すと，リセット信号がシミュレータに入力されます。

これで，入出力パネルの各設定の解説を終わります。

終　了

SM78Kxx を終了します。

SM78Kxx メニューの［ファイル(F)］→［終了(X)］を選択してください。

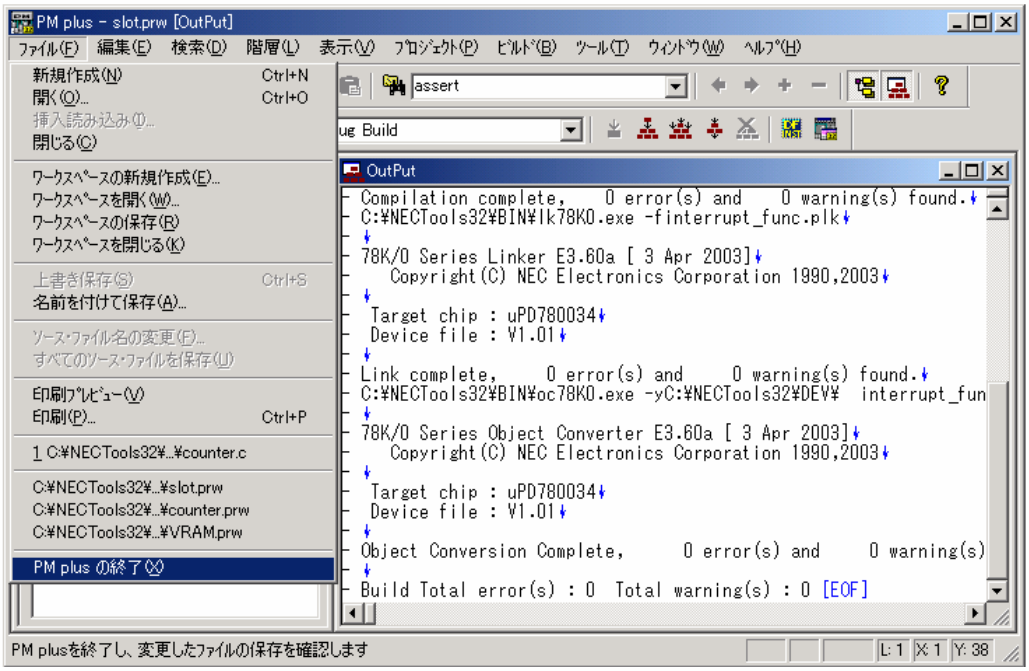


終了確認ダイアログが表示されます。

この章で設定した入出力パネルの状態などをプロジェクト・ファイルに保存する場合は［はい(Y)］を，保存しない場合は［いいえ(N)］を押してください。

PM plus を終了します。

PM plus のメニューの［ファイル(F)］→［PM plus の終了(X)］を選択してください。



PM plus のプロジェクト情報は逐次保存されているため，保存の確認はされません。

プログラム解説

ここでは，スロット・プログラムで使用している次の機能を C 言語で記述する方法について解説します。

- デバイス内蔵の特殊機能レジスタへのアクセス
- 割り込みや例外発生時の関数の定義
- 割り込みを制御する関数
- CPU を制御する命令

なお，スロット・プログラムのソース・リストは「付録 スロット・プログラム・ソースリスト」に添付してありますので，あわせてご覧ください。

CC78Kxx C コンパイラは，ANSI 規格で規定された言語仕様をサポートしています。
また，機種依存処理をできるだけ C 言語で記述可能にするために，ANSI 規格の C 言語仕様を拡張しています。
この拡張言語仕様により，割り込み時処理や SFR の参照など，機種に依存した処理を，オブジェクト効率を保ちながら C 言語で記述することが可能となり，プログラムの継承性や開発効率を向上させることができます。

拡張仕様には，次のものがあります。

- 外部変数の saddr 領域への割り付け指定（sreg 変数）
- 関数引数や自動変数の saddr 領域やレジスタへの割り付け指定（nerec, noauto 関数）
- 短い命令での関数呼び出し指定（callt 関数, callf 関数[注1]）
- SFR へのアクセス
- 割り込み処理の C 言語記述（レジスタ・バンク[注2]切り替え可能）
- 割り込み禁止／許可命令の出力
- C 言語ソース・プログラムへのアセンブラ記述挿入
- CPU 制御命令の出力
- 2 進数定数の記述

拡張言語仕様部分についての詳細は，ユーザーズ・マニュアル 「CC78Kxx C コンパイラ・パッケージ　言語編」を参照してください。

注 1. callf 命令は，78K0S ではサポートされていません。
2. 78K0S ではサポートされていません。

レジスタ名を用いた特殊機能レジスタへのアクセス『#pragma sfr』

各デバイスで内蔵している周辺機能のためのレジスタを，特殊機能レジスタと呼びます。
特殊機能レジスタを使うには，ソースの先頭部分にプラグマ指令『#pragma sfr』を記述します。

```
#pragma sfr
```

スロット・プログラムのソースでは，slot.c の始めに，#pragma sfr を記述しています。

[slot.c]

```
/* 特殊機能レジスタ名（SFR 名）の有効化 */
#pragma sfr
```

プラグマ指令『#pragma sfr』を記述すると，特殊機能レジスタ名を，通常の符号なし（unsigned）外部変数のように使用することができますが，『#pragma sfr』の記述なしに，特殊機能レジスタ名を使用すると「変数が未定義である（error: E2210: 特殊機能レジスタ名: not defined）」というエラーが出ます。

例

```
/* #pragma sfr の記述がない場合 */
main() {
    P1 = 0; /* P1 の未定義エラー */
}
```

特殊機能レジスタの詳細については，各デバイスのユーザーズ・マニュアル を参照してください。

割り込み関数の登録
『#pragma interrupt』または『#pragma vect』と『__interrupt』

割り込みとは，現在実行中の処理を強制的に中断させて，別の処理を行うことです。割り込みで中断された処理は，割り込んだ処理の終了後に再開します。

割り込みを発生させることを，割り込み要求と呼びます。

割り込み要求時に行われる処理は，関数として記述でき，割り込みの要因によって，実行する関数を指定することができます。このような関数を“割り込み関数”と呼びます。

ある関数を，割り込み関数とするには，次の作業が必要です。

- 関数名と割り込みの要因（割り込み要求名）とを対応づける
- 関数を割り込み関数に指定する

関数名と割り込み要求名とを対応づけるには，プラグマ指令『#pragma interrupt』または『#pragma vect』を使用します。

```
#pragma interrupt 割り込み要求名 関数名
```

```
#pragma vect 割り込み要求名 関数名
```

このプラグマ指令に記述された割り込み要求名に従い，関数を割り込み関数として登録します。

指定可能な割り込み要求名については，各デバイスのユーザーズ・マニュアルを参照してください。

関数を割り込み関数に指定するには，関数定義（もしくは関数宣言）に，修飾子 __interrupt を付けます。

```
__interrupt 関数定義 or 関数宣言
```

割り込み関数に指定された関数は，通常のレジスタの退避／復帰に加え，割り込みに対応したレジスタの退避／復帰も行います。また，reti 命令で復帰します。

割り込み関数に指定できる関数は，通常，戻り値も引き数もない関数（"void Func(void) "型）です。

スロット・プログラムのソースでは，関数名と割り込み要求名との対応づけを slot.c で，関数の割り込み関数の指定を interrupt_func.h で行っています。

[slot.c]

```
#pragma interrupt INTP0 stp_btn_Left
#pragma interrupt INTP1 stp_btn_Center
#pragma interrupt INTP2 stp_btn_Right
```

[interrupt_func.h]

```
__interrupt void stp_btn_Left(void);
__interrupt void stp_btn_Center(void);
__interrupt void stp_btn_Right(void);
```

部分的な割り込みの可否設定
『DI();』と『EI();』

ある処理を行っている間はマスカブル割り込みを禁止にし，処理が終わったらマスカブル割り込みを受け付けるようにするということも C 言語で記述できます。

割り込みの受け付けを部分的に禁止もしくは許可するには，割り込み制御関数（DI/EI）を利用します。
まず『#pragma DI』，『#pragma EI』を指定します。

```
#pragma DI
#pragma EI
```

```
DI();
```

DI 関数は，割り込みを禁止（di 命令を生成）します。

```
EI();
```

EI 関数は，割り込みを許可（ei 命令を生成）します。

指定例:

```
#pragma DI
#pragma EI

void func() {
      int i,j,k;
          ……
      DI();   /* 割り込み禁止 */

       /*
        * 割り込み禁止状態で行う必要のある処理
        */

      EI();   /* 割り込み許可 */
          ……
      return;
}
```

スロット・プログラムのソースでは，ボタンが押されたことによって割り込みを発生させるため，数字をカウント・アップするループの前に，割り込みを有効にしています。

[slot.c]

```
   /* 割り込み許可 */
   EI();
```

CPU 制御命令出力
『HALT();』と『STOP();』と『BRK();』と『NOP();』

CPU 制御を行う命令を，関数形式で C 言語で記述できます。
BRK 命令は 78K0, 78K4 のみサポートされています。

機能を使用する場合には，#pragma 指令で使用することを宣言します。

```
#pragma HALT
#pragma STOP
#pragma BRK
#pragma NOP
```

```
HALT();
```

78K0, 78K0S の場合，HALT 関数は，halt 命令を生成します。
78K4 の場合，STBC を操作するコードを生成します。

```
STOP();
```

78K0，78K0S の場合，STOP 関数は，stop 命令を生成します。
78K4 の場合，STBC を操作するコードを生成します。

```
BRK();
```

78K0，78K4 の場合，BRK 関数は，brk 命令を生成します。

```
NOP();
```

NOP 関数は，nop 命令を生成します。

指定例:

```
#pragma HALT
#pragma STOP
#pragma BRK
#pragma NOP

void func() {
        ……

    HALT();   /* halt 命令出力 */
    STOP();   /* stop 命令出力 */
    BRK();   /* brk 命令出力 */
    NOP();   /* nop 命令出力 */


        ……
    return;
}
```

【注意】

CC78K4 で，HALT()，STOP()は，STBC の CK1/CK0 の値を調べて，それに応じた HALT 用または STOP 用の値を STP/HLT に設定する命令を出力します。

（STBC に対しては，「MOV STBC,#値」でしか設定できません。）

そのため，STBC の 2, 3, 6, 7 ビット目には，0 とした命令が出力されます。

2, 3, 6, 7 ビット目が，0 固定ではないデバイスの場合には，HALT()，STOP()は，使用できませんので，注意してください。

7	6	5	4	3	2	1	0
0	0	CK1	CK0	0	0	STP	HLT

付　録

ここでは付録として，各章で扱った次の項目の詳細について紹介します。

uoVRAM.dll の作成方法

ここでは，第 2 章で使用する仮想 VRAM の外部部品（uoVRAM.dll）を Microsoft Visual C++（以後 VC++）で作成する方法を紹介します。

なお，第 2 章のサンプル環境には，完成された uoVRAM.dll が添付されていますので，ここでの操作は第 2 章を進めるにあたって必要な操作ではありません。本節はお客様ご自身で外部部品ユーザ・オープン・インタフェースを使用した外部部品を作成する際の参考資料です。

外部部品ユーザ・オープン・インタフェースの詳細については，SM78K シリーズ システム・シミュレータ Ver.2.30 以上 ユーザーズ・マニュアル 外部部品ユーザ・オープン・インタフェース仕様編（U15802J）を参照してください。

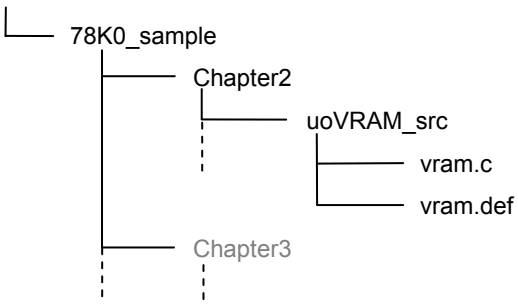
uoVRAM.dll のソース・ファイルは，他のサンプル・プログラムと一緒に入っています。作成には VC++ Ver.5 以上が必要です。ここでは，VC++ Ver.6 での作成方法を紹介します。

＜使用するファイル＞

uoVRAM.dll の作成には，自己展開型圧縮ファイルに入っているソース・ファイル（vram.c，vram.def）と，SM78K0 と同時にインストールされる外部部品ユーザ・オープン・インタフェース用ファイル（upart32.cpp，uparts32.h）を使用します。

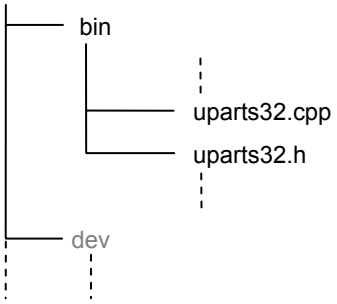
- 仮想 VRAM ソース・ファイル

  サンプル・プログラムを展開したディレクトリ



- 外部部品ユーザ・オープン・インタフェース用ファイル

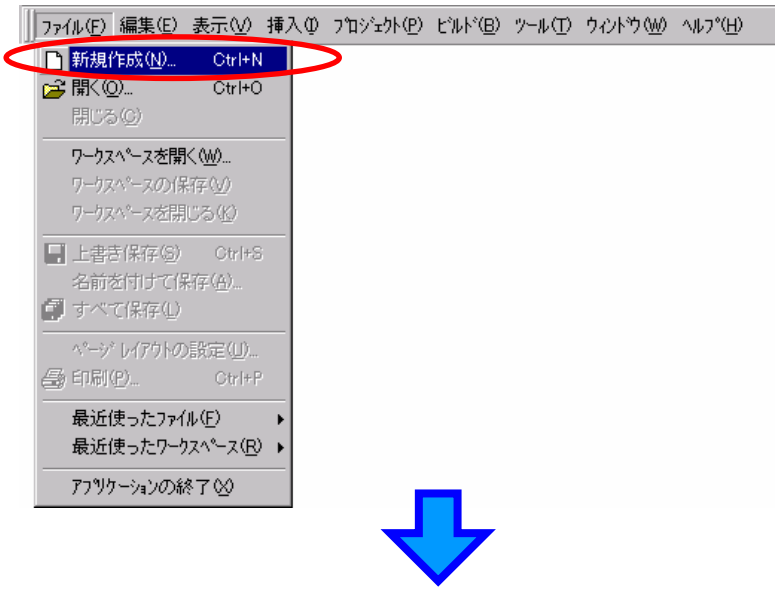
  C:¥NECTools32 （SM78K0 をインストールしたディレクトリ）



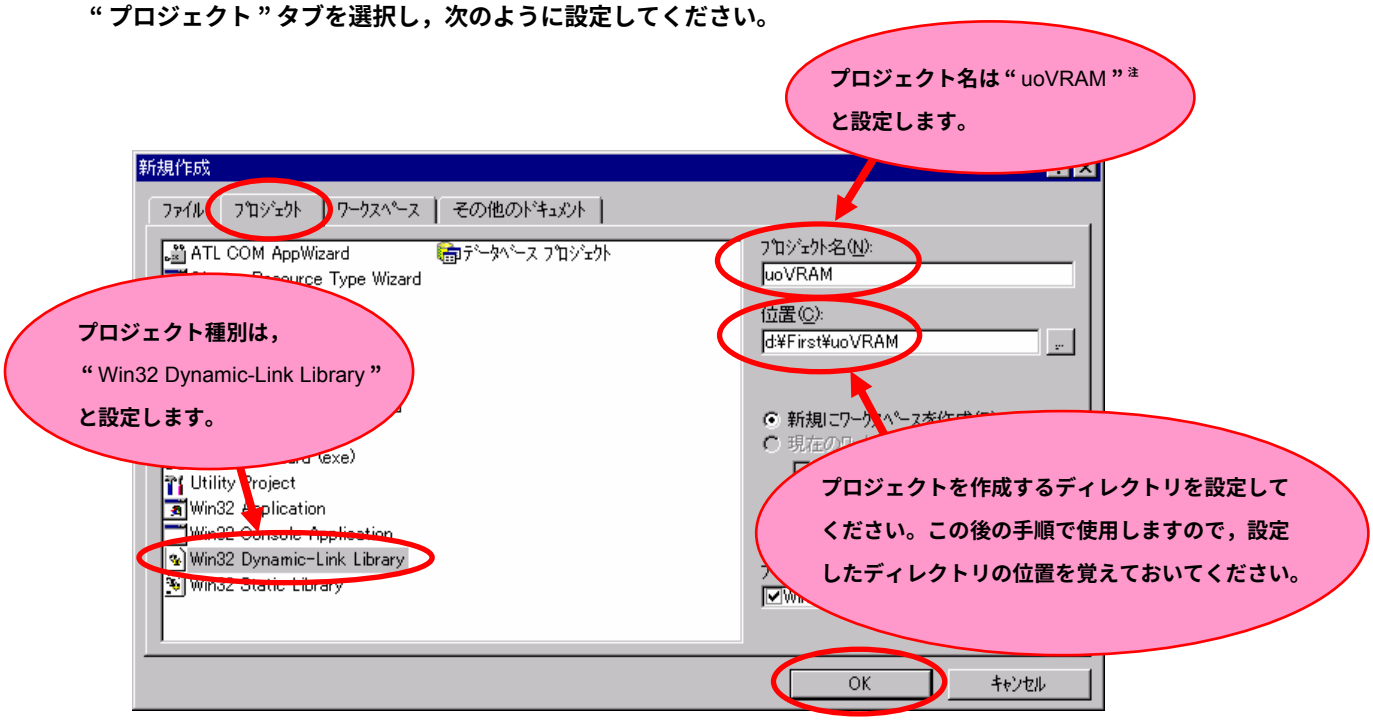
＜uoVRAM.dll の作成手順＞

VC++ Ver.6 でビルドする手順は次のとおりです。

1. VC++を起動し，新規に“ Win32 Dynamic-Link Library ”プロジェクトを作成します。
   まず，メニューの［ファイル(F)］→［新規作成(N)...］を選択してください。



“ プロジェクト ”タブを選択し，次のように設定してください。

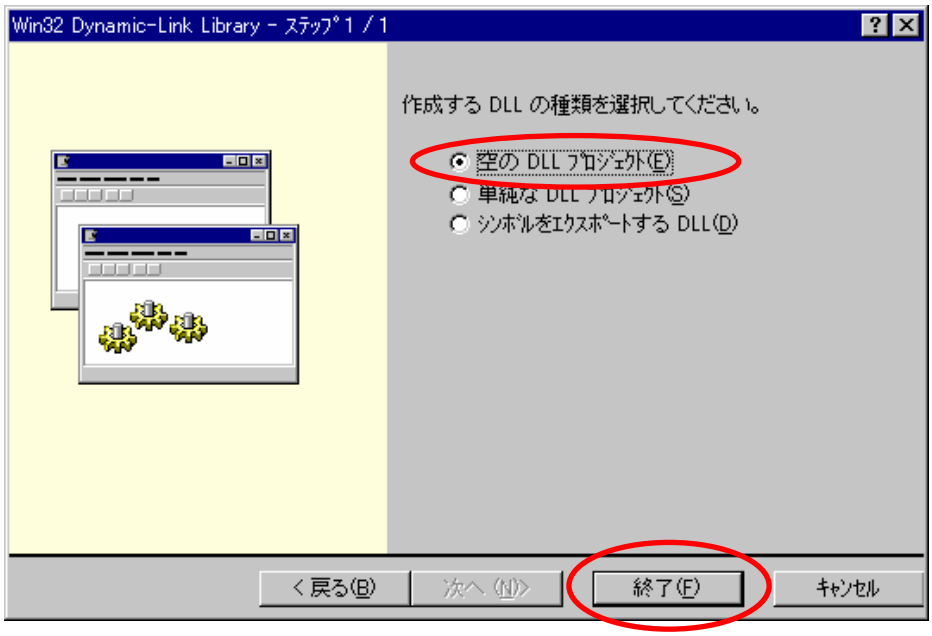


設定が完了したら，OK ボタンを押してください。

注 プロジェクト名は“ uoVRAM ”としてください。これ以外のプロジェクト名を使用する場合は，VC++が出力する DLL のファイル名が“ uoVRAM.dll ”になるようにオプションを変更してください。SM78Kxx の外部部品ユーザ・オープン・インタフェースでは，DLL のファイル名と，DLL から EXPORT する関数名を対応づける必要があります。本ソースをそのまま利用する場合，DLL のファイル名に uoVRAM.dll 以外を使用すると，SM78Kxx は DLL を正常に読むことができません。詳細については，**SM78K シリーズ システム・シミュレータ Ver.2.30 以上 ユーザーズ・マニュアル 外部部品ユーザ・オープン・インタフェース仕様編（U15802J）**を参照してください。

DLL の種類を選択します。

“空の DLL プロジェクト(E)”を選択してください。

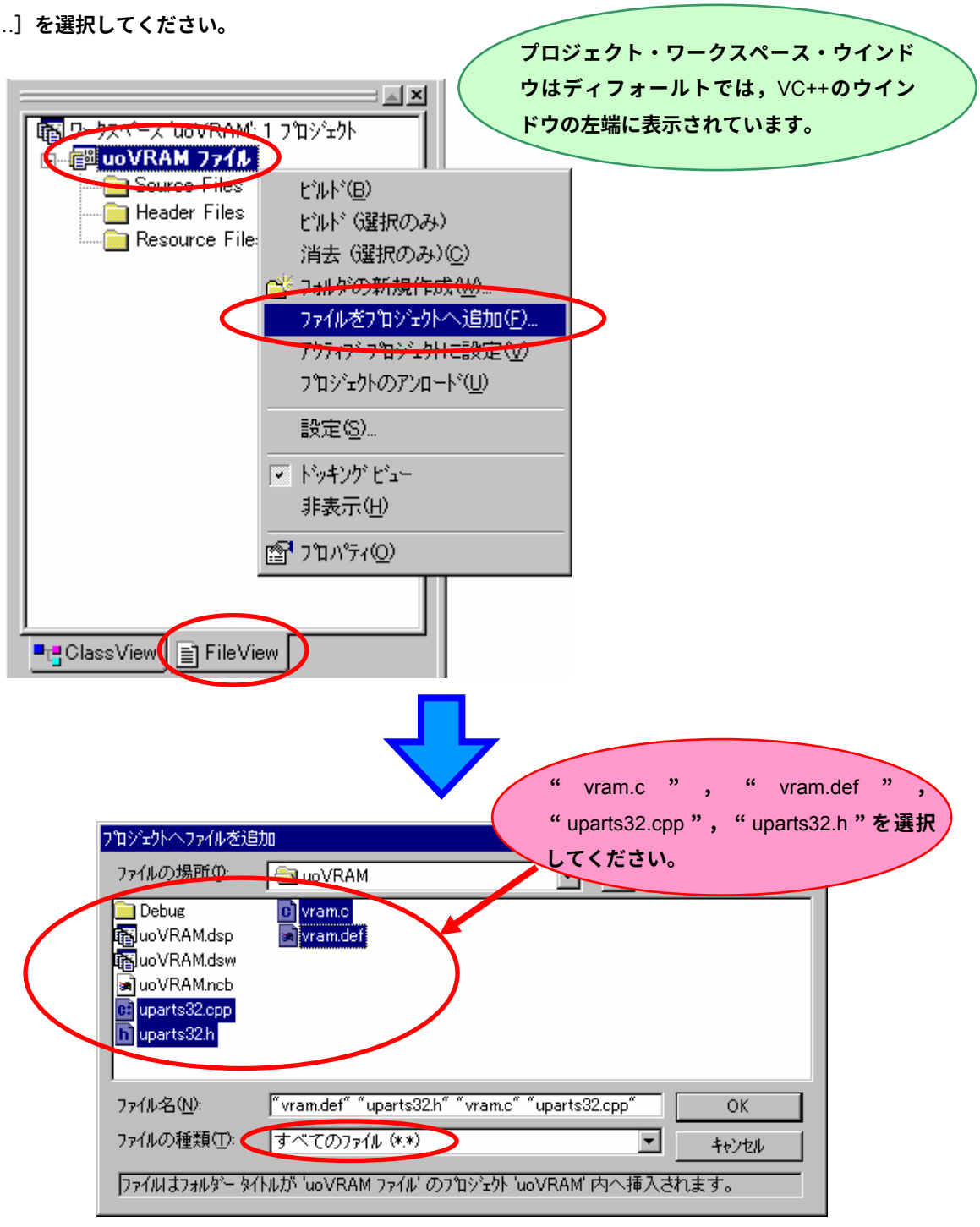


選択後，終了(F) ボタンを押すとプロジェクトが生成されます。

2. 1.でプロジェクトを作成したディレクトリに，仮想 VRAM のソース・ファイル（vram.c, vram.def）と外部部品ユーザ・オープン・インタフェース用ファイル（uparts32.cpp, uparts32.h）をコピーします（ファイルのある位置については<使用するファイル>をご覧ください）。

3. 2.でコピーしたファイル（vram.c, vram.def, uparts32.cpp, uparts32.h）を 1.で作成したプロジェクトに登録します。VC++のプロジェクト・ワークスペース・ウインドウの“File View”タブの中で設定します。

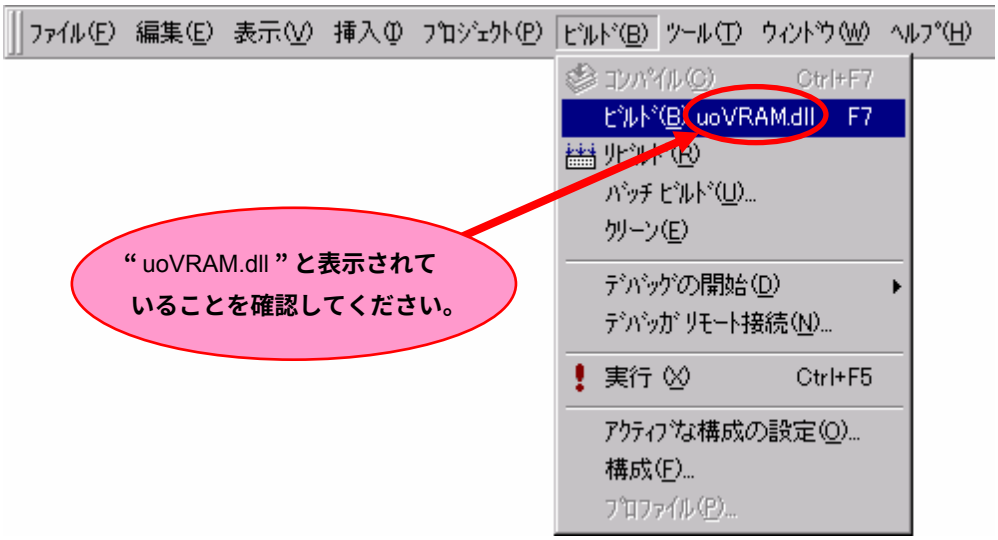
   まず，“File View”タブを選択してください。
   次に，“uoVRAM ファイル”を右クリックしてプルダウン・メニューを表示し，［ファイルをプロジェクトへ追加(F)…］を選択してください。



“ファイルの種類(T)”を“すべてのファイル (*.*)”にすると，4 つのファイルをまとめて選択できます。
選択後，OK ボタンを押すと，プロジェクトにファイルが登録されます。

ビルドを実行します。

メニューの［ビルド(B)］→［ビルド(B)］を選択してください。



以上の作業により，“uoVRAM.dll”が作成されます。

カウンタ・プログラム・ソースリスト

[counter.c]

78K0 のソースリスト (1/5)

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004
 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */


/************************************************************
 ************************************************************
 *
 * カウンタプログラム（μＰＤ７８００３４用)
 *
 ************************************************************
 ************************************************************/

/* コンパイラ定義 */
#pragma NOP
#pragma HALT
#pragma EI

/* 特殊機能レジスタ名(SFR 名)の有効化 */
#pragma sfr

/* 関数 interrupt1()を，INTTM00 の割込み関数に設定 */
#pragma vect INTTM00 interrupt1

/* カウンタ値保持変数 */
volatile int count1;  /* LED に表示する数値(1 の位)  */
volatile int count10; /* LED に表示する数値(10 の位) */

/************************************************************
 ************************************************************
 * ７セグメント LED に数値を表示
 *
 *  count1 に保持している値を LED の１桁目(1 の位)，count10 に
 *  保持している値を LED の 2 桁目(10 の位)に表示する。
 *
```

(2/5)

```
 * 関数名：putLED
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
 ************************************************************/
void putLED()
{
        static unsigned char  /* 7 セグメント LED '0' - '9'のパターン */
                box[10]={0x77,0x24,0x6b,0x6d,0x3c,0x5d,0x5f,0x74,0x7f,0x7d};/*←ERROR*/

        /* 7 セグメント LED にパターンを転送 (1 の位) */
        P4 = 0;
        P5 = box[count1];
        P4 = 1;

        /* 時間調整 */
        NOP();
        NOP();

        /* 7 セグメント LED にパターンを転送 (10 の位) */
        P4 = 0;
        P5 = box[count10];
        P4 = 2;
        return;
}

/************************************************************
 ************************************************************
 * デバッグ用メイン関数
 *
 *  以下の処理を行う。
 *  ・LED 表示の為に P5，P4 のモードを PM5,PM4 により設定する。

 *  ・カウンタ値(count1，count10)を初期化する。
 *  ・INTTM00 割込みを初期化し，割込みを許可する。
 *  ・INTTM00 割込み発生まで，CPU を HALT 状態にする。
 *    また，INTTM00 割込みが終了したら CPU を HALT 状態にし，
 *    次の INTTM00 割込みを待つ。
 *
 * 関数名：main
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
```

(3/5)

```
************************************************************/
void main(){
        /***************************************
         * 初期化処理
         ***************************************/
        /* 7 セグメント LED に出力するポートのモードを設定 */
        PM5 = 0x00;  /*P5 (P50-P57)は出力  */
        PM4 = 0x00;  /*P4 (P40-P47)は出力  */

        /* INTTM00 割込みの割込みレベルの設定と割込みの有効化 */
        TMPR00 = 0;/* INTTM00 割込みレベルを高優先順位レベルにする */
        TMMK00 = 0;/* INTTM00 割込みの受付を許可する */

        /* カウンタの初期化 */
        count1  = 0;/* LED に表示する数値の初期化(1 の位)  */
        count10 = 0;/* LED に表示する数値の初期化(10 の位) */

        /* 初期値の表示 */
        putLED();  /* LED に数値を表示 */


        /***************************************
         * メインループ
         ***************************************/
        /* この後の処理は割込み処理のみなので、
           割り込み処理の無い時は、HALT 状態にする。 */
        EI();    /* 割込みの許可 */
        while(1)
        {
            HALT();
        }
}
```

(4/5)

```
/************************************************************
 ************************************************************
 * カウントアップ処理
 *  (INTTM00 で呼び出される割込み関数)
 *
 *  count1,count10 で構成されるカウンタを 1 カウントアップする。
 *  99 までカウントしたら次は 0 にする。
 *  また，putLED 関数で，LED にカウンタの値を表示する。
 *
 * 関数名：interrupt1
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
```

(5/5)

```
************************************************************/
__interrupt
void interrupt1()
{
        /***************************************
         * カウンタのカウントアップ
         ***************************************/
        count1++ ;  /* 1 の位をカウントアップ */

        /* 桁上がり処理 */

        /* 1 の位の桁上がりが必要かどうか？ */
        if(count1=10)                              /*←ERROR*/
        { /* 1 の位が 10 なので，桁上がり処理を行う */
                count1 = 0; /* 1 の位を 0 に設定 */
                count10++;  /* 10 の位をカウントアップ */

                /* 10 の位の桁上がりが必要かどうか？ */
                if(count10==10)
                {
                /* 10 の位が 10 なので，桁上がり処理を行う */
                    count10 =0; /* 10 の位を 0 に設定 */
                                /* LED が２桁なので 99 の次は 0 に戻す */
                }

        }

        /***************************************
         * LED に数値を表示
         ***************************************/
        putLED();       /* LED に数値を表示 */
        return;
}
```

78K0S のソースリスト (1/4)

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004
 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */


/************************************************************
 ************************************************************
 *
 * カウンタプログラム（μＰＤ７８９０４６用)
 *
 ************************************************************
 ************************************************************/

/* コンパイラ定義 */
#pragma NOP
#pragma HALT
#pragma EI

/* 特殊機能レジスタ名(SFR 名)の有効化 */
#pragma sfr

/* 関数 interrupt1()を，INTWT の割込み関数に設定 */
#pragma vect INTWT interrupt1

/* カウンタ値保持変数 */
volatile int count1;  /* LED に表示する数値(1 の位)  */
volatile int count10; /* LED に表示する数値(10 の位) */

/************************************************************
 ************************************************************
 * ７セグメント LED に数値を表示
 *
 *  count1 に保持している値を LED の１桁目(1 の位)，count10 に
 *  保持している値を LED の 2 桁目(10 の位)に表示する。
 *
 * 関数名：putLED
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
 ************************************************************/
```

(2/4)

```
void putLED()
{
        static unsigned char  /* 7セグメント LED '0' - '9'のパターン */
                box[9]={0x77,0x24,0x6b,0x6d,0x3c,0x5d,0x5f,0x74,0x7f,0x7d};/*←ERROR*/

        /* 7セグメント LED にパターンを転送 (1の位) */
        P0 = 0;
        P1 = box[count1];
        P0 = 1;

        /* 時間調整 */
        NOP();
        NOP();

        /* 7セグメント LED にパターンを転送 (10の位) */
        P0 = 0;
        P1 = box[count10];
        P0 = 2;

        return;
}

/************************************************************
 ************************************************************
 * デバッグ用メイン関数
 *
 *  以下の処理を行う。
 *  ・LED表示の為にP1，P0のモードをPM1,PM0により設定する。
 *  ・カウンタ値(count1，count10)を初期化する。
 *  ・INTWT割込みを初期化し，割込みを許可する。
 *  ・INTWT割込み発生まで，CPUをHALT状態にする。
 *    また，INTWT割込みが終了したらCPUをHALT状態にし，
 *    次のINTWT割込みを待つ。
 *
 * 関数名：main
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
 ************************************************************/
void main(){
        /***************************************
         * 初期化処理
         ***************************************/
        /* 7セグメント LED に出力するポートのモードを設定 */
        PM1 = 0x00;  /*P1 (P10-P17)は出力  */
        PM0 = 0x00;  /*P0 (P00-P07)は出力  */
```

(3/4)

```
/* INTWT 割込みの割込みレベルの設定と割込みの有効化 */
        WTMK = 0;/* INTWT 割込みの受付を許可する */

        /* カウンタの初期化 */
        count1  = 0;/* LED に表示する数値の初期化(1 の位)  */
        count10 = 0;/* LED に表示する数値の初期化(10 の位) */

        /* 初期値の表示 */
        putLED();  /* LED に数値を表示 */


        /**************************************
         * メインループ
         **************************************/
        /* この後の処理は割込み処理のみなので，
           割り込み処理の無い時は，HALT 状態にする。 */
        EI();    /* 割込みの許可 */
        while(1)
        {
            HALT();  /* */
        }
}


/************************************************************
 ************************************************************
 * カウントアップ処理
 *  (INTWT で呼び出される割込み関数)
 *
 *  count1,count10 で構成されるカウンタを 1 カウントアップする。
 *  99 までカウントしたら次は 0 にする。
 *  また，putLED 関数で，LED にカウンタの値を表示する。
 *
 * 関数名：interrupt1
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
 ************************************************************/
__interrupt
void interrupt1()
{
        /**************************************
         * カウンタのカウントアップ
         **************************************/
        count1++ ;  /* 1 の位をカウントアップ */
```

(4/4)

```
        /* 桁上がり処理 */

        /* 1 の位の桁上がりが必要かどうか？ */
        if(count1=10)                                   /*←ERROR*/
        { /* 1 の位が 10 なので，桁上がり処理を行う */
                count1 = 0; /* 1 の位を 0 に設定 */
                count10++;  /* 10 の位をカウントアップ */

                /* 10 の位の桁上がりが必要かどうか？ */
                if(count10==10)
                {
                /* 10 の位が 10 なので，桁上がり処理を行う */
                    count10 =0; /* 10 の位を 0 に設定 */
                                /* LED が 2 桁なので 99 の次は 0 に戻す */
                }

        }

        /***************************************
         * LED に数値を表示
         ***************************************/
        putLED();       /* LED に数値を表示 */
        return;
}
```

78K4 のソースリスト (1/4)

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004

 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */


/************************************************************
 ************************************************************
 *
 * カウンタプログラム（μＰＤ７８４０３５用)
 *
 ************************************************************
 ************************************************************/

/* コンパイラ定義 */
#pragma NOP
#pragma HALT
#pragma EI

/* 特殊機能レジスタ名(SFR 名)の有効化 */
#pragma sfr

/* 関数 interrupt1()を，INTCO0 の割込み関数に設定 */
#pragma vect INTCO0 interrupt1

/* カウンタ値保持変数 */
volatile int count1;  /* LED に表示する数値(1 の位)  */
volatile int count10; /* LED に表示する数値(10 の位) */

/************************************************************
 ************************************************************
 * ７セグメント LED に数値を表示
 *
 *  count1 に保持している値を LED の１桁目(1 の位)，count10 に
 *  保持している値を LED の 2 桁目(10 の位)に表示する。
 *
 * 関数名：putLED
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
 ************************************************************/
```

(2/4)

```
void putLED()
{
        static unsigned char  /* 7 セグメント LED '0' - '9'のパターン */
                box[9]={0x77,0x24,0x6b,0x6d,0x3c,0x5d,0x5f,0x74,0x7f,0x7d};/*←ERROR*/

        /* 7 セグメント LED にパターンを転送 (1 の位) */
        P4 = 0;
        P5 = box[count1];
        P4 = 1;

        /* 時間調整 */
        NOP();
        NOP();

        /* 7 セグメント LED にパターンを転送 (10 の位) */
        P4 = 0;
        P5 = box[count10];
        P4 = 2;
        return;
}

/************************************************************
 ************************************************************
 * デバッグ用メイン関数
 *
 *  以下の処理を行う。
 *  ・LED 表示の為に P5，P4 のモードを PM5,PM4 により設定する。
 *  ・カウンタ値(count1，count10)を初期化する。
 *  ・INTC00 割込みを初期化し，割込みを許可する。
 *  ・INTC00 割込み発生まで，CPU を HALT 状態にする。
 *    また，INTC00 割込みが終了したら CPU を HALT 状態にし，
 *    次の INTC00 割込みを待つ。
 *
 * 関数名：main
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
 ************************************************************/
void main(){
        /***************************************
         * 初期化処理
         ***************************************/
        /* 7 セグメント LED に出力するポートのモードを設定 */
        PM5 = 0x00;  /*P5 (P50-P57)は出力  */
        PM4 = 0x00;  /*P4 (P40-P47)は出力  */
```

(3/4)

```
        /* INTCO0 割込みの割込みレベルの設定と割込みの有効化 */
        CICO0 = CICO0 & 0xF0;/* INTCO0 割込みレベルを高優先順位レベルにする（CPR001=0,CPR000=0） */
        CMK0 = 0;           /* INTCO0 割込みの受付を許可する */

        /* カウンタの初期化 */
        count1  = 0;/* LED に表示する数値の初期化(1 の位)  */
        count10 = 0;/* LED に表示する数値の初期化(10 の位) */

        /* 初期値の表示 */

        putLED();  /* LED に数値を表示 */


        /**************************************
         * メインループ
         **************************************/
        /* この後の処理は割込み処理のみなので，
           割り込み処理の無い時は，HALT 状態にする。 */
        EI();    /* 割込みの許可 */
        while(1)
        {
            HALT();  /* */
        }
}


/************************************************************
 ************************************************************
 * カウントアップ処理
 *  (INTCO0 で呼び出される割込み関数)
 *
 *  count1,count10 で構成されるカウンタを 1 カウントアップする。
 *  99 までカウントしたら次は 0 にする。
 *  また，putLED 関数で，LED にカウンタの値を表示する。
 *
 * 関数名：interrupt1
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *       int count1
 *       int count10
 *
 ************************************************************
 ************************************************************/
__interrupt
void interrupt1()
{
```

(4/4)

```
     /***************************************
     * カウンタのカウントアップ
     ***************************************/
    count1++ ;  /* 1 の位をカウントアップ */

    /* 桁上がり処理 */

    /* 1 の位の桁上がりが必要かどうか？ */
    if(count1=10)                              /*←ERROR*/
    { /* 1 の位が 10 なので，桁上がり処理を行う */
            count1 = 0; /* 1 の位を 0 に設定 */
            count10++;  /* 10 の位をカウントアップ */

            /* 10 の位の桁上がりが必要かどうか？ */
            if(count10==10)
            {
            /* 10 の位が 10 なので，桁上がり処理を行う */
                count10 =0; /* 10 の位を 0 に設定 */
                            /* LED が２桁なので 99 の次は 0 に戻す */
            }

    }

    /***************************************
     * LED に数値を表示
     ***************************************/
    putLED();       /* LED に数値を表示 */
    return;
}
```

スロット・プログラム・ソースリスト

[slot.c]

78K0 のソースリスト (1/4)

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004
 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */

/*********************************************
 * スロット・プログラム (uPD780034 用)
 *********************************************/

/* コンパイラ定義 */
#pragma EI

/* 特殊機能レジスタ名(SFR 名)の有効化 */
#pragma sfr

/* INTP0, INTP1, INTP2 それぞれの割り込み関数として,
 * 関数 stp_btn_Left(), stp_btn_Center(), stp_btn_Right()を設定 */
#pragma interrupt INTP0 stp_btn_Left
#pragma interrupt INTP1 stp_btn_Center
#pragma interrupt INTP2 stp_btn_Right

#include "interrupt_func.h"  /*割り込み関数の宣言 */

/* 表示(LED 点灯)位置の指定 */
unsigned char place;

/* 表示(LED 点灯)用数字データ */
unsigned char num_data[10]
    = {  0x77, 0x24, 0x6b, 0x6d, 0x3c, 0x5d, 0x5f, 0x74, 0x7f, 0x7d, };
       /* '0'   '1'   '2'   '3'   '4'   '5'   '6'   '7'   '8'   '9' */

/* ポート, 割り込みレベル等のターゲット CPU 環境の初期化関数 */
void init_target(void);

/* スロット表示用関数 */
void slot(void);
```

(2/4)

```
/************************************************************************
 * スロット用メイン関数
 *    表示用数字(0~9)をループさせて，LED に表示する．
 *    ループ中に割り込みが発生すると，対応する割り込み関数が呼ばれ，
 *    表示している数値を固定する．
 *
 * 関数名：main
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *      unsigned char place;
 ************************************************************************/
void main(void)
{

  /* ターゲット CPU 環境の初期化 */
  init_target();

  /* 表示位置指定変数の初期化 */
  place = 0x15; /* 10101 : LED を一つおきに 3 つ点灯する． */

  /* 7 セグメント LED 消灯 */
  P4 = 0x00;
  P5 = 0x00;
  P4 = 0xFF;

  /* スロット(数字点灯)の処理 */
  slot();

} /* main */


/************************************************************************
 * ポート，割り込みレベル等のターゲット CPU 環境の初期化関数
 *
 * 関数名：init_target
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 : なし
 ************************************************************************/
void init_target(void){
  /*
   * Port0 を 割り込みの入力に使用
   * Port5 を 7 セグメント LED 点灯用に使用
   * Port4 を 桁指定用に使用
   */

  /* Port0 の全 bit を入力モードにする． */
  PM0  = 0xFF;     /* モード・レジスタ(PM0)の全 bit を入力(1)に設定 */
```

(3/4)

```
  /* Port5 の全 bit を出力モードにする． */
  PM5  = 0x00;   /* モード・レジスタ(PM5)の全 bit を出力(0)に設定 */

  /* Port4 の全 bit を出力モードにする． */
  PM4  = 0x00;   /* モード・レジスタ(PM4)の全 bit を出力(0)に設定 */

  /* Port5,Port4 を入出力ポートモードで使用するために
   * メモリ拡張モードレジスタをポートモードに設定 */
  MEM = 0x00;

  /*
   * 外部端子による外部割り込み要求の有効エッジは，
   * 外部割り込み立ち上がりエッジ許可レジスタ(EGP)
   * 外部割り込み立ち下がりエッジ許可レジスタ(EGN)で指定する．
   *
   * 本プログラムでは，外部割り込み要求に INTP0,INTP1,INTP2 を使用するため
   * 外部割り込み立ち上がりエッジ許可レジスタ(EGP)を，それぞれ立ち上がりエッジ有効に設定し，
   * 外部割り込み立ち下がりエッジ許可レジスタ(EGN)を無効に設定する．
   */
  EGP = 0x07;  /* 0x07 = XX XX 0 1 1 1
                  *              ↑ ↑ ↑ ↑
                  *              | | |  INTP0
                  *              | | |
                  *              | |  INTP1
                  *              | |
                  *              |  INTP2
                  *              |
                  *               INTP3
                  */

  EGN = 0x00;  /* 0x00 = XX XX 0 0 0 0
                  *              ↑ ↑ ↑ ↑
                  *              | | |  INTP0
                  *              | | |
                  *              | |  INTP1
                  *              | |
                  *              |  INTP2
                  *              |
                  *               INTP3
                  *
                  *
                  * EGP | EGN  |
                  *-------------------------
                  *  0  |  0   | 割込み禁止
                  *  0  |  1   | 立ち下がりエッジ
                  *  1  |  0   | 立ち上がりエッジ
                  *  1  |  1   | 立ち上がり，立ち下がり両エッジ
                  */
  PMK0 = 0;  /* INTP0 の割り込みの受け付け許可 */
  PMK1 = 0;  /* INTP1 の割り込みの受け付け許可 */
  PMK2 = 0;  /* INTP2 の割り込みの受け付け許可 */

} /* init_target */
```

(4/4)

```
/*********************************************************************
 * スロット表示用関数
 *    表示用数字(0~9)をループさせて，LED に表示する．
 *    ループ中に割り込みが発生すると，対応する割り込み関数が呼ばれ，
 *    表示している数値を固定する．
 *
 * 関数名：slot
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *      unsigned char place;
 *      unsigned int num_data[];
 *********************************************************************/
void slot(void) {
  /*
   * 表示用数字を 0~9 までループさせる．
   * 数字の点灯位置は，place 変数に従う．
   */

  /* 表示数値(num_data)用インデックス */
  int num_idx = 0;

  /* 割り込み許可 */
  EI();

  while (1) { /* 無限ループ */

    /* 数字表示 */
    P5 = num_data[num_idx];
    P4 = place;

    num_idx++ ;

    /* num_data の要素は 10 個なので，添字を表す num_idx が 10 になったら，
     * インデックスの値は 0 に戻す必要がある． */
    if( num_idx >= 10 ) {
        num_idx = 0;
    }

  } /* While */

} /* slot */
```

78K0S のソースリスト (1/4)

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004
 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */

/*********************************************
 * スロット・プログラム (uPD789046 用)
 *********************************************/

/* コンパイラ定義 */
#pragma EI

/* 特殊機能レジスタ名(SFR 名)の有効化 */
#pragma sfr

/* INTP0，INTP1，INTP2 それぞれの割り込み関数として，
 * 関数 stp_btn_Left()，stp_btn_Center()，stp_btn_Right()を設定 */
#pragma interrupt INTP0 stp_btn_Left
#pragma interrupt INTP1 stp_btn_Center
#pragma interrupt INTP2 stp_btn_Right

#include "interrupt_func.h"  /*割り込み関数の宣言 */

/* 表示(LED 点灯)位置の指定 */
unsigned char place;

/* 表示(LED 点灯)用数字データ */
unsigned char num_data[10]
    = {  0x77, 0x24, 0x6b, 0x6d, 0x3c, 0x5d, 0x5f, 0x74, 0x7f, 0x7d, };
       /* '0'   '1'   '2'   '3'   '4'   '5'   '6'   '7'   '8'   '9' */

/* ポート，割り込みレベル等のターゲット CPU 環境の初期化関数 */
void init_target(void);

/* スロット表示用関数 */
void slot(void);


/*************************************************************************
 * スロット用メイン関数
 *    表示用数字(0~9)をループさせて，LED に表示する．
 *    ループ中に割り込みが発生すると，対応する割り込み関数が呼ばれ，
 *    表示している数値を固定する．
 *
```

```
●   関数名：main
●   引数 ：なし
●   戻り値：なし
●   使用グローバル変数 :
●   unsigned char place;
 ************************************************************************/
void main(void)
{

  /* ターゲット CPU 環境の初期化 */
  init_target();

  /* 表示位置指定変数の初期化 */
  place = 0x15; /* 10101 : LED を一つおきに 3 つ点灯する． */

  /* 7 セグメント LED 消灯 */
  P0 = 0x00;
  P1 = 0x00;
  P0 = 0xFF;

  /* スロット(数字点灯)の処理 */
  slot();

} /* main */


/************************************************************************
●   ポート，割り込みレベル等のターゲット CPU 環境の初期化関数
 *
●   関数名：init_target
●   引数 ：なし
●   戻り値：なし
●   使用グローバル変数 : なし
 ************************************************************************/
void init_target(void){
  /*
●   Port2 を 割り込みの入力に使用
●   Port1 を 7 セグメント LED 点灯用に使用
●   Port0 を 桁指定用に使用
   */

  /* Port2 の全 bit を入力モードにする． */
  PM2  = 0xFF;     /* モード・レジスタ(PM2)の全 bit を入力(1)に設定 */

  /* Port1 の全 bit を出力モードにする． */
  PM1  = 0x00;    /* モード・レジスタ(PM1)の全 bit を出力(0)に設定 */
```

(3/4)

```
  /* Port0 の全 bit を出力モードにする．*/
  PM0  = 0x00;   /* モード・レジスタ(PM0)の全 bit を出力(0)に設定 */

  /*
●   外部端子による外部割り込み要求の有効エッジは，
●   外部割り込みモード・レジスタ INTM0 で指定する．
   *
●   本プログラムでは，外部割り込み要求に INTP0,INTP1,INTP2 を使用するため
●   外部割り込みモード・レジスタ 0 (INTM0) を，それぞれ立ち上がりエッジ有効に設定する．
   */

  INTM0 = 0x54;  /* 0x54 = 0 1 0 1 0 1 XX
●    ↑   ↑   ↑
●    |   |   |
●    |   |   |
●    |   |  INTP0
●    |   |
●    |  INTP1
●    |
●   INTP2
                 *
●   00: 立ち下がりエッジ
●   01: 立ち上がりエッジ
●   10: RFU(予約)
●   11: 立ち上がり，立ち下がり両エッジ
                 */

  PMK0 = 0;  /* INTP0 の割り込みの受け付け許可 */
  PMK1 = 0;  /* INTP1 の割り込みの受け付け許可 */
  PMK2 = 0;  /* INTP2 の割り込みの受け付け許可 */


} /* init_target */


/************************************************************************
●   スロット表示用関数
●   表示用数字(0~9)をループさせて，LED に表示する．
●   ループ中に割り込みが発生すると，対応する割り込み関数が呼ばれ，
●   表示している数値を固定する．
 *
●   関数名：slot
●   引数 ：なし
●   戻り値：なし
●   使用グローバル変数 :
●   unsigned char place;
●   unsigned int num_data[];
 ************************************************************************/
```

(4/4)

```
void slot(void) {
  /*
   * 表示用数字を 0~9 までループさせる．
   * 数字の点灯位置は，place 変数に従う．
   */

  /* 表示数値(num_data)用インデックス */
  int num_idx = 0;

  /* 割り込み許可 */
  EI();

  while (1) { /* 無限ループ */

    /* 数字表示 */
    P0 = 0x00;
    P1 = num_data[num_idx];
    P0 = place;

    num_idx++ ;

    /* num_data の要素は 10 個なので，添字を表す num_idx が 10 になったら，
     * インデックスの値は 0 に戻す必要がある． */
    if( num_idx >= 10 ) {
        num_idx = 0;
    }

  } /* While */

} /* slot */
```

78K4 のソースリスト (1/4)

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004
 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */

/*********************************************
 * スロット・プログラム (uPD784035 用)
 *********************************************/

/* コンパイラ定義 */
#pragma EI

/* 特殊機能レジスタ名(SFR 名)の有効化 */
#pragma sfr

/* INTP0，INTP1，INTP2 それぞれの割り込み関数として，
 * 関数 stp_btn_Left()，stp_btn_Center()，stp_btn_Right()を設定 */
#pragma interrupt INTP0 stp_btn_Left
#pragma interrupt INTP1 stp_btn_Center
#pragma interrupt INTP2 stp_btn_Right

#include "interrupt_func.h"  /*割り込み関数の宣言 */

/* 表示(LED 点灯)位置の指定 */
unsigned char place;

/* 表示(LED 点灯)用数字データ */
unsigned char num_data[10]
    = {  0x77, 0x24, 0x6b, 0x6d, 0x3c, 0x5d, 0x5f, 0x74, 0x7f, 0x7d, };
       /* '0'   '1'   '2'   '3'   '4'   '5'   '6'   '7'   '8'   '9' */

/* ポート，割り込みレベル等のターゲット CPU 環境の初期化関数 */
void init_target(void);

/* スロット表示用関数 */
void slot(void);


/*************************************************************************
 * スロット用メイン関数
 *    表示用数字(0~9)をループさせて，LED に表示する．
 *    ループ中に割り込みが発生すると，対応する割り込み関数が呼ばれ，
 *    表示している数値を固定する．
 *
```

(2/4)

```
 * 関数名：main
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 :
 *      unsigned char place;
 ************************************************************************/
void main(void)
{

  /* ターゲット CPU 環境の初期化 */
  init_target();

  /* 表示位置指定変数の初期化 */
  place = 0x15; /* 10101 : LED を一つおきに 3 つ点灯する． */

  /* 7 セグメント LED 消灯 */
  P4 = 0x00;
  P5 = 0x00;
  P4 = 0xFF;

  /* スロット(数字点灯)の処理 */
  slot();

} /* main */


/************************************************************************
 * ポート，割り込みレベル等のターゲット CPU 環境の初期化関数
 *
 * 関数名：init_target
 *  引数 ：なし
 * 戻り値：なし
 * 使用グローバル変数 : なし
 ************************************************************************/
void init_target(void){
  /*
   * Port2 を 割り込みの入力に使用
   * Port5 を 7 セグメント LED 点灯用に使用
   * Port4 を 桁指定用に使用
   */

  /* Port2 は入力専用ポートの為，入力モードの設定はいらない． */

  /* Port5 の全 bit を出力モードにする． */
  PM5  = 0x00;   /* モード・レジスタ(PM5)の全 bit を出力(0)に設定 */

  /* Port4 の全 bit を出力モードにする． */
  PM4  = 0x00;   /* モード・レジスタ(PM4)の全 bit を出力(0)に設定 */
```

(3/4)

```
 /* Port5,Port4 を入出力ポートモードで使用するために
●    メモリ拡張モードレジスタをポートモードに設定 */
  MM = 0x00;

 /*
●    外部端子による外部割り込み要求の有効エッジは，
●    外部割り込みモード・レジスタ INTMn(n=1-0)で指定する．
   *
●    本プログラムでは，外部割り込み要求に INTP0,INTP1,INTP2 を使用するため
●    外部割り込みモード・レジスタ 0 (INTM0) を，それぞれ立ち上がりエッジ有効に設定する．
   */

  INTM0 = 0x54;  /* 0x54 = 0 1 0 1 0 1 XX
●    ↑   ↑   ↑
●    |   |   |
●    |   |   |
●    |   |  INTP0
●    |   |
●    |  INTP1
●    |
●    INTP2
                *
●    00: 立ち下がりエッジ
●    01: 立ち上がりエッジ
●    10: RFU(予約)
●    11: 立ち上がり，立ち下がり両エッジ
                */

  PMK0 = 0;  /* INTP0 の割り込みの受け付け許可 */
  PMK1 = 0;  /* INTP1 の割り込みの受け付け許可 */
  PMK2 = 0;  /* INTP2 の割り込みの受け付け許可 */


} /* init_target */


/************************************************************************
●    スロット表示用関数
●    表示用数字(0~9)をループさせて，LED に表示する．
●    ループ中に割り込みが発生すると，対応する割り込み関数が呼ばれ，
●    表示している数値を固定する．
 *
●    関数名：slot
●    引数 ：なし
●    戻り値：なし
●    使用グローバル変数 :
●    unsigned char place;
●    unsigned int num_data[];
 ************************************************************************/
```

(4/4)

```
void slot(void) {
  /*
   * 表示用数字を 0~9 までループさせる．
   * 数字の点灯位置は，place 変数に従う．
   */

  /* 表示数値(num_data)用インデックス */
  int num_idx = 0;

  /* 割り込み許可 */
  EI();

  while (1) { /* 無限ループ */

    /* 数字表示 */
    P4 = 0x00;
    P5 = num_data[num_idx];
    P4 = place;

    num_idx++ ;

    /* num_data の要素は 10 個なので，添字を表す num_idx が 10 になったら，
     * インデックスの値は 0 に戻す必要がある． */
    if( num_idx >= 10 ) {
        num_idx = 0;
    }

  } /* While */

} /* slot */
```

[interrupt_func.h]

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004
 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */

/* スロットの各桁の表示を止める関数 */

/* __interrupt 関数修飾子によって，
 * stp_btn_Left()，stp_btn_Center()，stp_btn_Right()が，
 * 割り込み関数であることを宣言する．
 */
__interrupt void stp_btn_Left(void);
__interrupt void stp_btn_Center(void);
__interrupt void stp_btn_Right(void);
```

[interrupt_func.c]

```
/*
 *  Copyright (C) NEC Electronics Corporation 2000,2004
 *  All rights reserved by NEC Electronics Corporation. ThisProgram must be used solely
 *  for the purpose for which it was furnished by NEC Electronics Corporation. No part
 *  of thisProgram may be reproduced or disclosed to others, in any form,
 *  without the prior written permission of NEC Electronics Corporation.
 */

#include "interrupt_func.h"

extern unsigned char place;  /* 点灯位置の指定 */

/* 7 セグメント LED は，
 * 桁信号に割り当てた端子(本ソースでは port5)の出力がアクティブ時に，
 * 対応する LED を点灯/消灯させることができる．
 * 接続端子の出力情報を受け取り，その値に従って表示し，値が変化するまで
 * 点灯し続ける．
 */

void stp_btn_Left(void) {
  /*
   * 左端の桁の表示を，現在の数値のまま固定する．
   *    (入出力パネル上では，数値の動きが止まったように見える．)
   */

  /* 表示位置指定変数(place)の，
   * 表示を止めたい(固定したい)桁のビットを 0 にする．*/
   place = place & 0xEF; /* 0xEF = 1110 1111 */

}

void stp_btn_Center(void) {
  /* 中央の桁の表示を，現在の数値のまま固定する．*/
  place = place & 0xFB; /* 0xFB = 1111 1011 */
}

void stp_btn_Right(void) {
  /* 右端の桁の表示を，現在の数値のまま固定する．*/
  place = place & 0xFE; /* 0xFE = 1111 1110 */
}
```

［メ　モ］

【発　行】
NECエレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753
電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。
URL(アドレス)　http://www.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）
電　話　：044−435−9494
E-mail　：info@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクス特約店へお申し付けください。