In-system FLASH Programming with C16x/ST10 APNT_138

Author: Reinhard Keil (rk@keil.com), Keil Elektronik GmbH
☎ ++49 89 456040-13

OVERVIEW

In-system FLASH device programming typically requires that during the FLASH programming, no code executes from the device that is being programmed. This is a problem since most systems use a single large FLASH device. When this is the case, the Flash programming routines must be copied into and executed from RAM.

The Keil C166 development tools ***Version 4.02 and higher*** help solve this problem!

The L166 Linker/Locator is now able to store function code in FLASH memory while allowing you to define a different execution address (usually in RAM). You can easily copy the code into RAM and execute it. This simplifies the FLASH programming and minimizes code overhead.

This application note describes how to use the Keil C16x/ST10 development tools and the L166 Linker/Locator for flash programming.

LINKER EXTENSIONS

Extensions in the L166 Linker/Locator allow you to:

- Execute code and access constants at different RAM addresses than what is actually stored in ROM.
- Access the section starting address and section length directly from the C language.

These extensions let you relocate your FLASH **Erase** and FLASH **Write** routines to RAM before execution. You no longer have to write a separate program for the In-System FLASH programming software. Additionally, no code overhead is introduced since function pointers are not required to call routines that modify the FLASH.

COMPILER EXTENSIONS

A new header file (**SROM.H**) is included in the **INC** directory. This file contains macros that provide you with the ROM address, RAM address, and length of the FLASH programming functions.

You use these macros when copying the FLASH routines from FLASH to RAM.

CHANGING THE EXECUTION ADDRESS OF A SECTION

The L166 **SECTIONS** directive has been extended as follows:

Syntax: **SECTIONS** (***sectionname***%***classname*** (***exec_address***) [***store_address***])

Description: Brackets ("[]") in the **SECTIONS** directive signal to the L166 Linker/Locator that a section will be marked as an SROM section. An SROM section gets an execution address that is typically in RAM space and a storage address that is in ROM.

An SROM section is split into two logical sections: the execution section and the storage section. The class name for the execution section is unchanged from prior versions while the storage section receives a class name of SROM.

The ***exec_address*** defines the physical address for the execution section. It is specified in parentheses after the section name. This is the address where the code or constants will be copied before execution.

The ***store_address*** defines the physical address for the storage section. The address specified may be prefixed with an exclamation point ("**!**") to tell the linker not to reserve memory at the ***exec_address***. Use this if you want the section copied over RAM that is temporarily used, for example, the System Stack area. If no physical address is specified, the section will be located in the memory area defined for the SROM class.

Examples: In the following example, the section with the name **?PR?ABC** and the class **NCODE** is assigned an execution address of 0xFA00 and a storage address of 0x1000.

```
L166 abc.obj SECTIONS (?PR?ABC%NCODE (0xFA00) [0x1000])
```

In the following example, the ***store_address*** is prefixed with an exclamation point ("**!**"), so there will be no memory reserved at the execution address range (0FA00h).

```
L166 abc.obj SECTIONS (?PR?ABC%NCODE (0xFA00), [!0x1000]
```

In the following example, empty brackets are used. The section will be stored in the SROM memory class. Therefore, you must specify an address range for that memory class.

```
L166 abc.obj SECTIONS (?PR?ABC%NCODE (0xFA00), []
```

In the following example, only an exclamation point ("**!**") is specified for the ***store_address***. This means that no memory is reserved at the ***exec_address*** and the section will be stored in the SROM memory class.

```
L166 abc.obj SECTIONS (?PR?ABC%NCODE (0xFA00), [!]
```

GETTING THE SECTION STARTING ADDRESS AND LENGTH

In your C program, you need to know the starting address and length of the code that programs the FLASH. To get this information, you must create external variable declarations that the linker resolves at link time. The linker uses information it gets from the section (specified with the **SECTIONS** directive) to resolve these symbols.

The external variable names depend on the name of the section. For example, if ABC is the section name:

- **`_PR_ABC_l`** is the length in bytes of the section ?PR?ABC.
- **`_PR_ABC_s`** is the ***store_address*** of the section ?PR?ABC.
- **`_PR_ABC_t`** is the ***exec_address*** (or target address) of the section ?PR?ABC.

Note that the underscore character (“_”) is used in place of the question mark character (“?”) in the C extern declarations.

The file **SROM.H** contains the following macro definitions to help you access the section related information.

SROM_PS

This macro defines all the external symbols that are required to access the SROM section information for a program section.

```
#define SROM_PS(n)                                                \
extern unsigned char huge _PR_##n##_s_;  /* section source start */ \
extern unsigned char huge _PR_##n##_l_;  /* section len          */ \
extern unsigned char huge _PR_##n##_t_;  /* target address       */
```

Use this macro in your program as follows:

```
SROM_PS (FLASH)    // information definitions for section ?PR?FLASH
```

SROM_PS_SRC

This macro returns the storage address for a program code section.

```
#define SROM_PS_SRC(n) ((void *)       &_PR_##n##_s_)
```

Use this macro in your program as follows:

```
  void *source;
  source = SROM_PS_SRC (FLASH)    // get the store_address for section ?PR?FLASH
```

SROM_PS_TRG

This macro returns the target address or execution address for a program code section.

```
#define SROM_PS_TRG(n) ((void *)       &_PR_##n##_t_)
```

Use this macro in your program as follows:

```
  void *target;
  target = SROM_PS_TRG (FLASH)    // get the exec_address for section ?PR?FLASH
```

SROM_PS_LEN

This macro returns the length for a program code section.

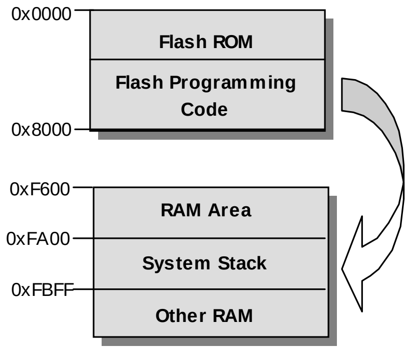
```
#define SROM_PS_LEN(n) ((unsigned int) &_PR_##n##_l_)
```

Use this macro in your program as follows:

```
  int length;

  length = SROM_PS_LEN (FLASH)    // get the length for section ?PR?FLASH
```

EXAMPLE PROGRAM

An example program (**FLASH**) will help demonstrate the new features in C166 Version 4.02 that make FLASH programming easier.



The **FLASH** example program assumes that the target hardware has FLASH memory from 0000h to 8000h and RAM from 0F600h to 0FDFFh. This is the typical configuration for single-chip C16x/ST10 applications.

Target systems with external FLASH and RAM are programmed similarly. FLASH and RAM devices may be located anywhere in the C16x/ST10 address space.

Before your FLASH **Erase** and **Write** routines are executed, program code is copied to RAM space.

The C program and the Linker invocation are shown in the following section.

FLASH PROJECT

The FLASH project contains the following source files:

SROM.C Contains the C code that copies and executes the FLASH programming routines. This module may be extended to become the actual functions used in your software.

PFLASH.A66 This source file contains the programming routines for the **PFlash** area in the **C167CS**. Available functions in this module are:

- **PFlash_Write:** Write 64 Bytes to the PFlash area.
- **PFlash_Erase:** Erase Sector in the PFlash area.
- **PFlash_Reset:** Reset the PFlash module to read mode and clear status.

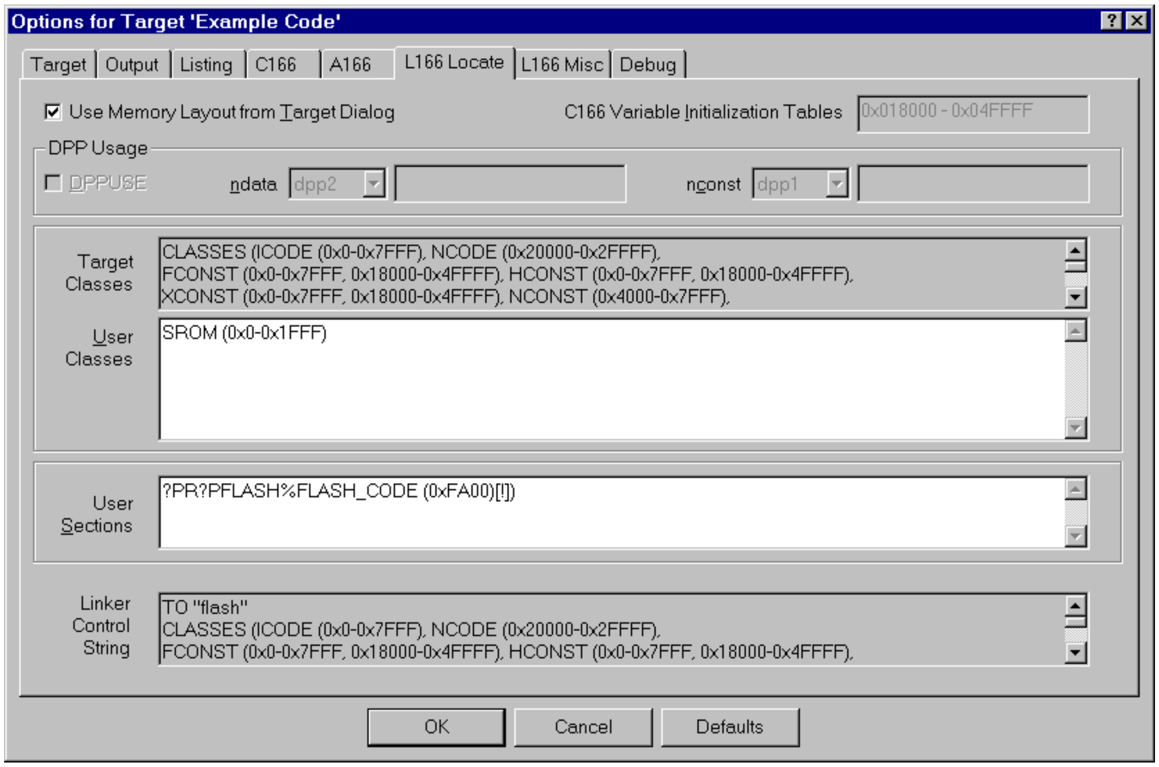
The code in this module is stored in the **?PR?FLASH** section in the **FLASH_CODE** class.

SROM.C

```
#include <string.h>
#include <intrins.h>
#include <srom.h>    // SROM Handling definitions
#include <reg167.h>

SROM_PS (PFLASH)     // define SROM program segment from PFLASH.A66

// Write 64 Bytes to target_adr in Flash Memory
extern int  far PFlash_Write (void huge *target_adr, void huge *buffer);

// Erase the Flash memory sector specified by sector_adr
extern int  far PFlash_Erase (void huge *sector_adr);

// Reset the Flash memory device
extern void far PFlash_Reset (void);

unsigned char buf[64] = { 1, 2, 3, 4, 5, 6, 7, 8 };

void main (void)  {
  // copy flash program code to execution address
  hmemcpy (SROM_PS_TRG(PFLASH), SROM_PS_SRC(PFLASH), SROM_PS_LEN(PFLASH));

  _bfld_ (PSW, 0xF000, 0xF000); // disable interrupts
  _nop_ ();
  _nop_ ();
  PFlash_Erase (0x4000);        // Erase sector starting at address 0x4000;
  _bfld_ (PSW, 0xF000, 0x0000); // enable interrupts

  _bfld_ (PSW, 0xF000, 0xF000); // disable interrupts
  _nop_ ();
  _nop_ ();
  PFlash_Write (0x4000, buf);   // program 64 bytes to address 0x4000;
  _bfld_ (PSW, 0xF000, 0x0000); // enable interrupts

  while (1);
}
```

L166 Linker/Locator Settings

In the µVision2 **Options for Target** dialog box in the **L166 Locate** tab, the **?PR?PFLASH** section must be defined as an SROM section in the **User Sections** text box. In this example, the execution address 0xFA00 is assigned to this section. Since no ***store_address*** is specified, the section will be located somewhere within the SROM memory class. The address range for the SROM class is specified in the **User Classes** text box.


Options for Target 'Example Code'

Tabs: Target | Output | Listing | C166 | A166 | L166 Locate | L166 Misc | Debug

☑ Use Memory Layout from Target Dialog

C166 Variable Initialization Tables: 0x018000 - 0x04FFFF

DPP Usage: ☐ DPPUSE — ndata: dpp2 — nconst: dpp1

Target Classes:
```
CLASSES (ICODE (0x0-0x7FFF), NCODE (0x20000-0x2FFFF),
FCONST (0x0-0x7FFF, 0x18000-0x4FFFF), HCONST (0x0-0x7FFF, 0x18000-0x4FFFF),
XCONST (0x0-0x7FFF, 0x18000-0x4FFFF), NCONST (0x4000-0x7FFF),
```

User Classes:
```
SROM (0x0-0x1FFF)
```

User Sections:
```
?PR?PFLASH%FLASH_CODE (0xFA00)[!])
```

Linker Control String:
```
TO "flash"
CLASSES (ICODE (0x0-0x7FFF), NCODE (0x20000-0x2FFFF),
FCONST (0x0-0x7FFF, 0x18000-0x4FFFF), HCONST (0x0-0x7FFF, 0x18000-0x4FFFF),
```

In the USA:
Keil Software, Inc.
16990 Dallas Parkway, Suite 120
Dallas, TX 75248-1903
USA

Sales: 800-348-8051
Phone: 972-735-8052
FAX: 972-735-8055

E-mail: sales.us@keil.com
support.us@keil.com

Internet: http://www.keil.com/

In Europe:
Keil Elektronik GmbH
Bretonischer Ring 15
D-85630 Grasbrunn b. Munchen
Germany

Phone: (49) (089) 45 60 40 - 0
FAX: (49) (089) 46 81 62

E-mail: sales.intl@keil.com
support.intl@keil.com