Flash Integration Application Note

µ-blox ag
Bertastrasse 1
CH8003 Zürich
Switzerland
http://www.u-blox.ch

23rd May 2000

Abstract

The µ-blox GPS receivers have 4 or 8Mbit of flash memory. The program code is stored and executed from the flash memory. It uses only a fraction of the space available. This application note describes and provides sample code, how to implement the interface to the flash memory for custom use.
For detailed information on specific flash memorys the reader is referred to the datasheets of the flash memory manufacturers, such as AMD, Fujitsu or SGS Thomson Microelectornics.

1 Flash Types

The 29LV*00 or 29W*00 flash memories are organized in sectors. These flash memories offer fast read access times, allowing the operation with high-speed microprocessor without wait states. Commands are written to the command register using standard microprocessors write timings. Register contents serve as input to an internal state-machine which controls the erase and programming circuitry.
The flash is 16 bits wide, do not try to write byte values. Only even addresses are allowed. The flash base address is `0x0A000000`. Programming is allowed in any sequence and across sector boundaries. Beware that a data "0" cannot be programmed back to "1". Attempting to do so may either hang up the device or result in an apparent success according to the data polling algorithm but a read will show that the data is still "0". Only erase operations can convert "0"'s to "1"'s. A sector is typically erased and verified in 1 second. A word is typically written in about 15 µs.
The following tables describe the architecture of the flash memory families used. In order to determine the flash architecture used in your module you must identify the flash. Section 2 provides code samples on this topic.

4 Mbit					
Top			Bottom		
Sector #	Size (kByte)	Address Offset	Sector #	Size (kByte)	Address Offset
0	64	`0x000000`	0	16	`0x000000`
1	64	`0x010000`	1	8	`0x004000`
2	64	`0x020000`	2	8	`0x006000`
3	64	`0x030000`	3	32	`0x008000`
4	64	`0x040000`	4	64	`0x010000`
5	64	`0x050000`	5	64	`0x020000`
6	64	`0x060000`	6	64	`0x030000`
7	32	`0x070000`	7	64	`0x040000`
8	8	`0x078000`	8	64	`0x050000`
9	8	`0x07A000`	9	64	`0x060000`
10	16	`0x07C000`	10	64	`0x070000`

Table 1: 29LV400 Flash Memory Sector Architecture

8 Mbit					
Top			Bottom		
Sector #	Size (kByte)	Address Offset	Sector #	Size (kByte)	Address Offset
0	64	`0x000000`	0	16	`0x000000`
1	64	`0x010000`	1	8	`0x004000`
2	64	`0x020000`	2	8	`0x006000`
3	64	`0x030000`	3	32	`0x008000`
4	64	`0x040000`	4	64	`0x010000`
5	64	`0x050000`	5	64	`0x020000`
6	64	`0x060000`	6	64	`0x030000`
7	64	`0x070000`	7	64	`0x040000`
8	64	`0x080000`	8	64	`0x050000`
9	64	`0x090000`	9	64	`0x060000`
10	64	`0x0A0000`	10	64	`0x070000`
11	64	`0x0B0000`	11	64	`0x080000`
12	64	`0x0C0000`	12	64	`0x090000`
13	64	`0x0D0000`	13	64	`0x0A0000`
14	64	`0x0E0000`	14	64	`0x0B0000`
15	32	`0x0F0000`	15	64	`0x0C0000`
16	8	`0x0F8000`	16	64	`0x0D0000`
17	8	`0x0FA000`	17	64	`0x0E0000`
18	16	`0x0FC000`	18	64	`0x0F0000`

Table 2: 29LV800 Flash Memory Sector Architecture

2 Flash Functions

The following subsections list the files used to access the flash. Make a call to `flash_init` before using any other function. This is typically done in `main.c::main()` just before the call to `TRK_StartTrack()`. After that, you can make your calls to `flash_read()`, `flash_write()` or `flash_erase_sector()`.
The section 3 describes how to include these files to your project.

***Note** – Erasing a sector takes about 1 second. This influences the GPS receiver performance. To ensure proper functioning use* `RxMPauseTrack()` *before and* `RxMRestartTrack()` *after your call to* `flash_erase_sector()`*. If you erase more than one sector, a hotstart may be necessary. Use this function pair only when the RTC is set like in the following example.*

```
RTCset = GetRealGoodTime(&tow, &wnum);
if (RTCset)
  RxMPauseTrack();
flash_erase_sector(sector);
if (RTCset)
  RxMRestartTrack();
```

2.1 flashsec.src

The file `flashsec.src` defines the new sections. The code that accesses the flash must be located in the SRAM. We define two sections `Pflashram` and `Pflashrom`. We must copy the code from section `Pflashrom` to `Pflashram` before using it. This is done by `flash_config()`.

```
;/* ********************************************************************** */
;/*                                                                        */
;/* u-blox AG/Zurich,Switzerland                                           */
;/*                                                                        */
;/* This source file is the sole property of u-blox AG. Reproduction       */
;/* or utilization of this source in whole or part is forbidden without    */
;/* the written consent of u-blox AG.                                      */
;/*                                                                        */
;/* ********************************************************************** */
          .SECTION Pflashram,CODE,ALIGN=4
          .SECTION Pflashrom,CODE,ALIGN=4
          .SECTION C,DATA,ALIGN=4
__F_ROM_B .DATA.L  (STARTOF Pflashrom)
__F_ROM_E .DATA.L  (STARTOF Pflashrom) + (SIZEOF Pflashrom)
__F_RAM_B .DATA.L  (STARTOF Pflashram)
__F_RAM_E .DATA.L  (STARTOF Pflashram) + (SIZEOF Pflashram)
          .EXPORT  __F_ROM_B
          .EXPORT  __F_ROM_E
          .EXPORT  __F_RAM_B
          .EXPORT  __F_RAM_E
          .END
```

2.2 flash.h

Just include this file into your source files when you plan to use the flash functions.

```
/* ********************************************************************** */
/*                                                                        */
/* u-blox AG/Zurich,Switzerland                                           */
/*                                                                        */
/* This source file is the sole property of u-blox AG. Reproduction       */
/* or utilization of this source in whole or part is forbidden without    */
/* the written consent of u-blox AG.                                      */
/*                                                                        */
/* ********************************************************************** */
void flash_init();
unsigned char flash_sectors();
unsigned short* flash_sector_to_baseaddress(unsigned char Sector);
unsigned long flash_sector_size(unsigned char Sector);
char* flash_identify_manufacturer(unsigned short Code);
char* flash_identify_devicetype(unsigned short Code);

void flash_config();
int flash_identify(unsigned short* pType, unsigned short* pManu);
int flash_write(volatile unsigned short* pAddress, unsigned short Value);
#define flash_read(pAddress) ((unsigned short) (*((volatile unsigned short*) pAddress)))
int flash_erase_sector(unsigned char Sector);

extern unsigned char  Flash_Sectors;
extern unsigned char  Flash_FirstFree_Sector;
extern unsigned short* Flash_FirstFree_Address;
extern unsigned short* Flash_LastFree_Address;
```

2.3 flash.c

This file demonstrates the integration of the flash.

```
/* ********************************************************************** */
/*                                                                        */
/* u-blox AG/Zurich,Switzerland                                           */
/*                                                                        */
/* This source file is the sole property of u-blox AG. Reproduction       */
/* or utilization of this source in whole or part is forbidden without    */
/* the written consent of u-blox AG.                                      */
/*                                                                        */
/* ********************************************************************** */
#include "umanager.h"
#include "sh1reg.h"
#include "csection.h"
#include "lp.h"

#include "flash.h"

/* defines                                                                */
/* ********************************************************************** */

/* flash manufacturers */
#define FLASH_MANUFACTURER_AMD     0x0001
#define FLASH_MANUFACTURER_FUJITSU 0x0004
#define FLASH_MANUFACTURER_ST      0x0020

/* flash types */
/* AMD / Fujitsu Family */
```

```
#define FLASH_TYPE_29LV400T   0x22B9 /*  4 Mbit Top    */
#define FLASH_TYPE_29LV400B   0x22BA /*  4 Mbit Bottom */
#define FLASH_TYPE_29LV800T   0x22DA /*  8 Mbit Top    */
#define FLASH_TYPE_29LV800B   0x225B /*  8 Mbit Bottom */
#define FLASH_TYPE_29LV160T   0x22C4 /* 16 Mbit Top    */
#define FLASH_TYPE_29LV160B   0x2249 /* 16 Mbit Bottom */
/* ST Family */
#define FLASH_TYPE_29W400T    0x00EE /*  4 Mbit Top    */
#define FLASH_TYPE_29W400B    0x00EF /*  4 Mbit Bottom */
#define FLASH_TYPE_29W800T    0x00D7 /*  8 Mbit Top    */
#define FLASH_TYPE_29W800B    0x005B /*  8 Mbit Bottom */
#define FLASH_TYPE_29W160T    0x00C4 /* 16 Mbit Top    */
#define FLASH_TYPE_29W160B    0x0049 /* 16 Mbit Bottom */

/* flash sector sizes */
#define FLASH_4MB_SEC  11     /*  4 Mbit */
#define FLASH_8MB_SEC  19     /*  8 Mbit */
#define FLASH_16MB_SEC 35     /* 16 Mbit */
/* checks type for size / architecture */
#define IS_FLASH_4MB_TOP(x)  ((x ==  FLASH_TYPE_29LV400T) || (x ==  FLASH_TYPE_29W400T))
#define IS_FLASH_4MB_BOT(x)  ((x ==  FLASH_TYPE_29LV400B) || (x ==  FLASH_TYPE_29W400B))
#define IS_FLASH_8MB_TOP(x)  ((x ==  FLASH_TYPE_29LV800T) || (x ==  FLASH_TYPE_29W800T))
#define IS_FLASH_8MB_BOT(x)  ((x ==  FLASH_TYPE_29LV800B) || (x ==  FLASH_TYPE_29W800B))
#define IS_FLASH_16MB_TOP(x) ((x ==  FLASH_TYPE_29LV160T) || (x ==  FLASH_TYPE_29W160T))
#define IS_FLASH_16MB_BOT(x) ((x ==  FLASH_TYPE_29LV160B) || (x ==  FLASH_TYPE_29W160B))
/* checks type for size */
#define IS_FLASH_4MB(x)  (IS_FLASH_4MB_TOP(x) || IS_FLASH_4MB_BOT(x))
#define IS_FLASH_8MB(x)  (IS_FLASH_8MB_TOP(x) || IS_FLASH_8MB_BOT(x))
#define IS_FLASH_16MB(x) (IS_FLASH_16MB_TOP(x) || IS_FLASH_16MB_BOT(x))
/* checks type for top or bottom */
#define IS_FLASH_TOP(x)  (IS_FLASH_4MB_TOP(x) || IS_FLASH_8MB_TOP(x) || IS_FLASH_16MB_TOP(x))
#define IS_FLASH_BOT(x)  (IS_FLASH_4MB_BOT(x) || IS_FLASH_8MB_BOT(x) || IS_FLASH_16MB_BOT(x))

/* global variables                                                        */
/* *********************************************************************** */

/* type of the flash will be stored here */
static unsigned short Flash_Type = 0x0000;

/* first usable sector of the flash (zero based) */
static unsigned char  Flash_FirstFree_Sector = 0;
/* last usable sector of the flash */
static unsigned char Flash_Sectors = 0;

/* special flash address */
/* flash base address */
#define FLASH_BASE 0x0A000000
/* first usable address in the flash */
static unsigned short* Flash_FirstFree_Address = 0;
/* last usable address in the flash */
static unsigned short* Flash_LastFree_Address = 0;

/* prototypes of private functions                                         */
/* *********************************************************************** */

int flash_sram_identify(unsigned short* pType, unsigned short* pManu);
int flash_sram_write(volatile unsigned short* pAddress, unsigned short Value);
int flash_sram_erase_sector(volatile unsigned short* pAddress);

/* functions                                                               */
/* *********************************************************************** */

/* *********************************************************************** */
/* init the whole flash stuff, all the global variables are setup here      */
/* ********************************************************************** */
void flash_init()
{
  unsigned short* Address;
  unsigned short Manu;
  unsigned char i;
  extern int* _F_ROM_E;
  /* config the flash */
  flash_config();
  /* identify the flash, we just need the flash type */
  flash_identify(&Flash_Type, &Manu);
  /* do the special initialisation depending on flash type */
  /* find the number of sectors */
  Flash_Sectors = flash_sectors();
  /* find the first usable sector and address. the first free address is */
  /* _F_ROM_E, but we assume, that we want to be able to erase it, so the */
  /* first usable address is the base address of the next sector */
  Flash_FirstFree_Sector = 0;
  Flash_FirstFree_Address = 0;
  for (i = 0; i < Flash_Sectors; i ++)
  {
    Address = flash_sector_to_baseaddress(i);
    if ((int*) Address > _F_ROM_E)
    {
      Flash_FirstFree_Sector = i;
      Flash_FirstFree_Address = Address;
      break;
    }
  }
  /* now we look for the last usable address in the flash */
  Flash_LastFree_Address = (unsigned short*) (flash_sector_size(Flash_Sectors)
        + (unsigned long) flash_sector_to_baseaddress(Flash_Sectors));
  Flash_LastFree_Address --;
  /* print out debug message */
  PRINTF("FLASH Manufacturer %s (Code 0x%4.4X)",
   flash_identify_manufacturer(Manu), Manu);
  PRINTF("FLASH Device Type %s (Code 0x%4.4X)",
   flash_identify_devicetype(Flash_Type), Flash_Type);
  PRINTF("FLASH Sectors %u to %u Address 0x%8.8X to 0x%8.8X",
        Flash_FirstFree_Sector, Flash_Sectors - 1,
        Flash_FirstFree_Address, Flash_LastFree_Address);
}

/* ********************************************************************** */
/* finds and returns the number of sectors for a given flash type           */
/* ********************************************************************** */
unsigned char flash_sectors()
{
  if (IS_FLASH_4MB(Flash_Type))
    return FLASH_4MB_SEC;
  else if (IS_FLASH_8MB(Flash_Type))
    return FLASH_8MB_SEC;
  else if (IS_FLASH_16MB(Flash_Type))
    return FLASH_16MB_SEC;
  else
    return 0;
}

/* ********************************************************************** */
/* calculates and returns the base address for a given sector               */
/* ********************************************************************** */
unsigned short* flash_sector_to_baseaddress(unsigned char Sector)
```

```
{
  unsigned long Address;
  unsigned char i;
  /* add the sizes of the sectors to find the base address */
  Address = FLASH_BASE;
  for (i = 0; i < Sector; i ++)
    Address += flash_sector_size(i);
  return (unsigned short*) Address;
}

/* ******************************************************************** */
/* returns the sector size for a given sector                           */
/* ******************************************************************** */
unsigned long flash_sector_size(unsigned char Sector)
{
  unsigned char Index;
  /* only calc size of sectors that are available */
  if (Sector >= Flash_Sectors)
    return 0;
  /* bottom: smaller sectors = smaller sector sizes */
  /*        larger sectors  = larger sector sizes  */
  /* top:    smaller sectors = larger sector sizes  */
  /*        larger sectors  = smaller sector sizes */
  /* generate the index to the sectors. top = reverse, bottom = normal */
  if (IS_FLASH_TOP(Flash_Type))
    Index = Flash_Sectors - Sector - 1;
  else if (IS_FLASH_BOT(Flash_Type))
    Index = Sector;
  else
    return 0;
  /* now detemine the size of the sector */
  switch (Index)
  {
    case 0:
      return 0x04000; /* 16 x 1024 Bytes */
    case 1:
    case 2:
      return 0x02000; /*  8 x 1024 Bytes */
    case 3:
      return 0x08000; /* 32 x 1024 Bytes */
    default:
      return 0x10000; /* 64 x 1024 Bytes */
  }
}

/* ******************************************************************** */
/* convert the manufacturer code into a human readable text             */
/* ******************************************************************** */
char* flash_identify_manufacturer(unsigned short Code)
{
  switch (Code)
  {
    case FLASH_MANUFACTURER_AMD:
      return "AMD";
    case FLASH_MANUFACTURER_FUJITSU:
      return "Fujitsu";
    case FLASH_MANUFACTURER_ST:
      return "SGS Thomson";
    default:
      return "Unknown";
  }
}
```

```
/* ********************************************************************** */
/* convert the devicetype code into a human readable text                 */
/* ********************************************************************** */
char* flash_identify_devicetype(unsigned short Code)
{
  if (IS_FLASH_4MB_TOP(Code))
    return "4Mbit Top";
  else if (IS_FLASH_4MB_BOT(Code))
    return "4Mbit Bottom";
  else if (IS_FLASH_8MB_TOP(Code))
    return "8Mbit Top";
  else if (IS_FLASH_8MB_BOT(Code))
    return "8Mbit Bottom";
  else if (IS_FLASH_16MB_TOP(Code))
    return "16Mbit Top";
  else if (IS_FLASH_16MB_BOT(Code))
    return "16Mbit Bottom";
  else
    return "Unknown";
}

/* functions of basic operations                                          */
/* they call the lowlevel routines and have some security checks          */
/* to minimize sram space the functions were splitted (flash_sram)        */
/* ********************************************************************** */

/* ********************************************************************** */
/* configure the GPIO pin and copy the code from flash to sram            */
/* ********************************************************************** */
void flash_config()
{
  extern int* _F_ROM_B;
  extern int* _F_ROM_E;
  extern int* _F_RAM_B;
  register int *p;
  register int *q;
  /* copy the flash code into sram */
  /* flashrom -> flashram */
  for (p=_F_ROM_B, q=_F_RAM_B; p<_F_ROM_E; p++, q++)
    *q=*p;
  /* PA14 is connected with the wait pin of the flash */
  /* it is an input */
  pPAIOR &= ~0x4000;
  /* PA14 is used as general purpose input output */
  pPACR1 &= ~0x3000;
}

/* ********************************************************************** */
/* identify flash, read the type and manufacturer of the flash,           */
/* returns 0 on error                                                     */
/* ********************************************************************** */
int flash_identify(unsigned short* pType, unsigned short* pManu)
{
  int Ret;
  CSState csState;
  if ((pType == NULL) || (pManu == NULL))
   return FALSE;
  /* enter critical section */
  csState = csEnter();
  TPM_DISABLE();
  /* do identify it */
  Ret = flash_sram_identify(pType, pManu);
  /* exit critical section */
  TPM_ENABLE();
  csExit(csState);
  return Ret;
}

/* ********************************************************************** */
/* writes a value to an address, returns 0 on error                       */
/* ********************************************************************** */
int flash_write(volatile unsigned short* pAddress, unsigned short Value)
{
  int Ret;
  CSState csState;
  /* only allow writing to addresses in sectors that are not used by the code */
  if ((pAddress > Flash_LastFree_Address) || (pAddress < Flash_FirstFree_Address))
    return FALSE;
  /* we only can write 0’s, this is flash technology */
  if ((*pAddress & Value) != Value)
    return FALSE;
  /* enter critical section */
  csState = csEnter();
  TPM_DISABLE();
  /* do write to the flash */
  Ret = flash_sram_write(pAddress, Value);
  /* exit critical section */
  TPM_ENABLE();
  csExit(csState);
  return Ret;
}

/* ********************************************************************** */
/* erases a sector, returns 0 on error                                    */
/* ********************************************************************** */
int flash_erase_sector(unsigned char Sector)
{
  int Ret;
  CSState csState;
  volatile unsigned short* pBaseAddress;
  /* only allow erasing of sectors that are not used by the code */
  if ((Sector >= Flash_Sectors) || (Sector < Flash_FirstFree_Sector))
    return FALSE;
  /* select sector */
  pBaseAddress = flash_sector_to_baseaddress(Sector);
  /* enter critical section */
  csState = csEnter();
  TPM_DISABLE();
  /* do erase the secor */
  Ret = flash_sram_erase_sector(pBaseAddress);
  /* exit critical section */
  TPM_ENABLE();
  csExit(csState);
  return Ret;
}

/* ********************************************************************** */
/* this code should be placed into SRAM ...                               */
/* ********************************************************************** */

/* flash_config will copy flashrom to flashram */
#pragma section flashrom

/* address to the flash command registers */
#define FLASH_5555 *((volatile unsigned short*) (FLASH_BASE + (0x5555 << 1)))
#define FLASH_2AAA *((volatile unsigned short*) (FLASH_BASE + (0x2AAA << 1)))
```

```
/* polls the busy signal of the flash */
/* used by flash_erase_sector and flash_write */
#define FLASH_WAIT while (!(pPADR & 0x4000))

/* ************************************************************************ */
/* identifys the flash                                                      */
/* ************************************************************************ */
int flash_sram_identify(unsigned short* pType, unsigned short* pManu)
{
  /* read flash manufacturer code */
  FLASH_5555 = 0x00AA;
  FLASH_2AAA = 0x0055;
  FLASH_5555 = 0x0090;
  *pManu = *((volatile unsigned short*)(FLASH_BASE + 0x00000000));
  /* read flash type code */
  FLASH_5555 = 0x00AA;
  FLASH_2AAA = 0x0055;
  FLASH_5555 = 0x0090;
  *pType = *((volatile unsigned short*)(FLASH_BASE + 0x00000002));
  /* read reset */
  FLASH_5555 = 0x00F0;
  return TRUE;
}

/* ************************************************************************ */
/* write to the flash                                                       */
/* ************************************************************************ */
int flash_sram_write(volatile unsigned short* pAddress, unsigned short Value)
{
  /* send write command */
  FLASH_5555 = 0x00AA;
  FLASH_2AAA = 0x0055;
  FLASH_5555 = 0x00A0;
  /* write Value */
  *pAddress=Value;
  /* wait for operation to end */
  FLASH_WAIT;
  return TRUE;
}

/* ************************************************************************ */
/* erase a sector of the flash                                              */
/* ************************************************************************ */
int flash_sram_erase_sector(volatile unsigned short* pAddress)
{
  /* send erase command */
  FLASH_5555 = 0x00AA;
  FLASH_2AAA = 0x0055;
  FLASH_5555 = 0x0080;
  FLASH_5555 = 0x00AA;
  FLASH_2AAA = 0x0055;
  /* select sector */
  *pAddress = 0x0030;
  /* wait for operation to end */
  FLASH_WAIT;
  return TRUE;
}

#pragma section
```

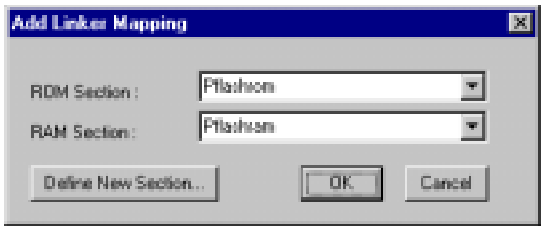
3 How to setup your Project

1. Copy the files `flash.c`, `flash.h` and `flashsec.src` to the project directory.
2. Open the Hitachi compiler and load the project.
3. Add the files `flash.c` and `flashsec.src` to the project (Menu: *Project* -> *Edit...*).
4. Compile the project. Linker warnings should appear.

```
Initialising...
Building...

Compiling...
c:\projectpath\flash.c

Assembling...
c:\projectpath\flashsec.src

Linking...
** 120 START ADDRESS NOT SPECIFIED FOR SECTION(Pflashrom)
** 120 START ADDRESS NOT SPECIFIED FOR SECTION(Pflashram)

Build Complete...
0 Errors , 2 Warnings
```

5. Edit the linker options (Menu: *Options* -> *Linker...*)

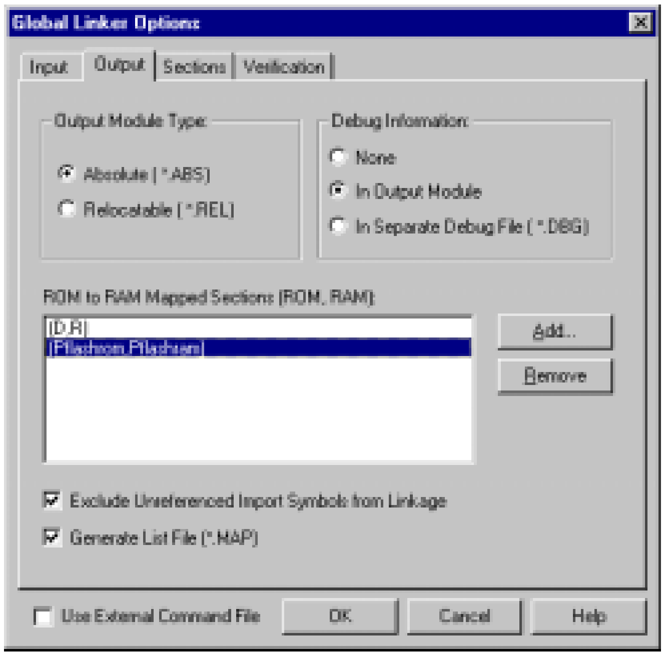
   - In the *Output* register click *Add...*. Add the linker mapping for ROM section Pflashrom to RAM section Pflashram.



ROM Section : Pflashrom
RAM Section : Pflashram

Figure 1: Add Linker Mapping

   - The *Output* register should now look like in figure 2.



Output Module Type: (● Absolute (*.ABS)) ○ Relocatable (*.REL)

Debug Information: ○ None (● In Output Module) ○ In Separate Debug File (*.DBG)

ROM to RAM Mapped Sections (ROM, RAM):

(D,R)
(Pflashrom,Pflashram)

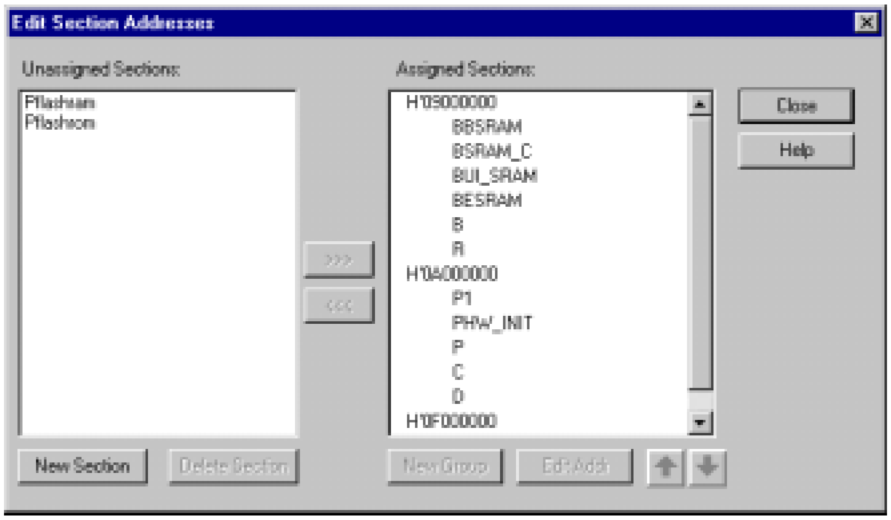
☑ Exclude Unreferenced Import Symbols from Linkage

☑ Generate List File (*.MAP)

☐ Use External Command File

Figure 2: Global Linker Options, Output

- In the *Sections* register click *Edit Section Addresses....* Assign the two unassigned sections Pflashrom and Pflashram.
- The Pflashram section must be assigned to the ram address space, which has the base address `0x09000000`.
- The Pflashrom section must be assigned as the last section of the flash address space, which has the base address `0x0A000000`.



Unassigned Sections:

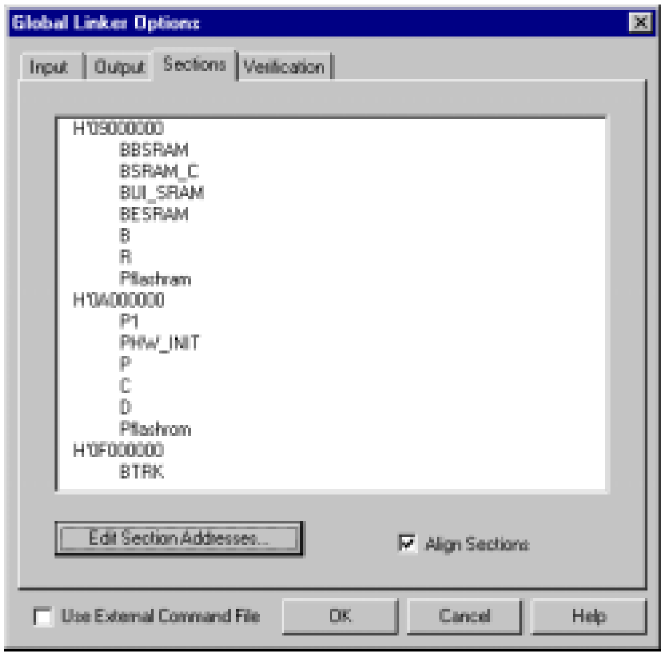
Pflashram
Pflashrom

Assigned Sections:

```
H'09000000
      BBSRAM
      BSRAM_C
      BUI_SRAM
      BESRAM
      B
      R
H'0A000000
      P1
      PHW_INIT
      P
      C
      D
H'0F000000
```

Figure 3: Edit Section Addresses

- The *Sections* register should now look like in figure 4.



```
H'09000000
        BBSRAM
        BSRAM_C
        BUI_SRAM
        BESRAM
        B
        R
        Pflashram
H'0A000000
        P1
        PHW_INIT
        P
        C
        D
        Pflashrom
H'0F000000
        BTRK
```

☑ Align Sections

☐ Use External Command File

Figure 4: Global Linker Options, Sections

6. Include the `flash.h` header file to your source files and implement your calls to the flash functions, don't forget the call `flash_init()` to initialize the flash before reading, writing or erasing the flash.

7. Compile the project again.

4 Related Documents

Related documents can be found at `http://www.u-blox.ch/restricted`.

Revision History		
Date	**Revision**	**Changes**
Mai. 19, 2000	2.00	Added STM support, sram fuctions splitted into high and low function.
Apr. 10, 1999	1.03	Nex STM flash memorys, code changed.
Dec. 10, 1999	1.02	Section size calculation changed.
Mai. 5, 1999	1.00	8MB Flash type changed, Critical Section IRQ proof.
Apr. 16, 1999	0.90	Initial Version