SIEMENS

Microcontrollers

ApNote AP1608

☑ additional file
APXXXX01.EXE available

In-circuit Programming of the C166 Family Flash Devices

The Siemens 16-bit Microcontroller family comprises two flash devices, the SAB 88C166 with 32 Kbytes on chip flash memory and the C167CR-16F with 128 Kbytes on chip flash memory

Author: K. Westerholz / Tools Group, Munich

AP1608 ApNote - Revision History		
Actual Revision : Rel.60 (5.10)		Previous Revison: Rel.41
Page of actual Rel.	Page of prev. Rel.	Subjects changes since last release)
		SW change in AP160860.EXE

1 Apstract

The Siemens 16-bit Microcontroller family comprises two flash devices, the SAB 88C166 with 32 Kbytes on chip flash memory and the C167CR-16F with 128 Kbytes on chip flash memory; both partitioned into 4 independent banks. The C167CR-16F can be programmed more than 1000 times. They are suited most for applications requiring high flexibility combined with standard hardware.

For instance, applications like motor control demand for a Flash device to be programmed at the end of the production line in its final environment. Furthermore, prototyping and small series are now much easier with the flash devices. To offer this flexibility, it is necessary to enable in-circuit programming at the end of the production line or even in the field. If that flexibility is not required, C166 Flash devices can also be preprogrammed by means of dedicated programming boards.

This application note describes the necessary steps to be taken enabling in-circuit programming for the C167CR-16F. The process of programming presented here is based on the built-in boot strap loader of the C166 devices. All software including the programming data is downloaded from a host PC into the internal RAM of the microcontroller. Since the application requires only 1 Kbytes of internal RAM, no assumptions about the hardware environment have to be made. The solution described here employs a standard evaluation board . No changes to the board are necessary.

The application code is provided by our mail box Tel:+49 89 498431 and via our distribution channels. For more detailed technical information regarding the devices please refer to our data sheets.

Hardware requirements

Three steps are needed, firstly providing the 12V programming voltage, secondly activating the on chip boot strap loader, and thirdly connecting the device to a host PC.

All C167 devices have a Vpp pin dedicated to the 12V programming voltage. You only have to connect a 12V power supply to this pin. Using the Ertec EVA Board, you connect the programming voltage with the pin E7 of the on board connector.

The second prerequisite is that the bootstrap loader mode (BSL) is enabled. The C167 enters BSL mode if pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in BSL is activated, independent of the selected bus mode. The bootstrap loader is stored in a special Boot-ROM, not part of the 128 Kbytes Flash memory area. The Ertec EVA Board provides a switch to activate the BSL mechanism (BSL=ON).

In order to download the application from a PC, a serial link has to be established. The C167 already provides an asynchronous serial interface that only has to be directly connected to COM1 of your PC. Supposing you are using an Ertec EVA Board, you can directly connect its serial connector with the COM1 interface of your PC

2 In-circuit programming algorithm

The whole mechanism of in-circuit programming presented here relies on the BSL. By means of the BSL the connection between your target and the host PC is established. The number of bytes that are transmitted by means of the BSL are 32 bytes of code or data.

They are stored at a fixed location (FA40h-FA5Fh) in the internal RAM. Afterwards this code sequence is automatically executed starting at address FA40h. By means of this initial user code sequence, a second more comprehensive loader can be loaded and executed that is able to load larger programs. In the application presented, the second loader is employed to download a full featured hex-loader. Then the hex-loader downloads further applications. The very first application automatically downloaded by the hex loader is a small shell enabling actions like status inquiries, erasing, and programming the Flash memory. Routines needed for the applications like erasing the Flash memory banks are downloaded on demand because of memory restrictions.

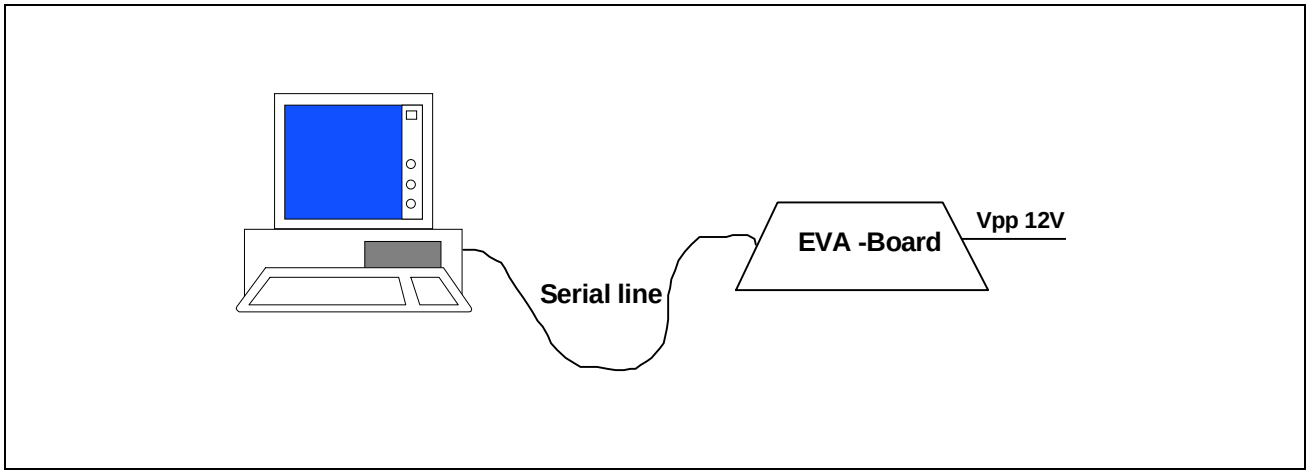

Figure 1:
Setup for in-circuit programming

Assuming you want to start a communication with your target. You firstly invoke your server program „flash.exe“ on the host side. Then you activate the BSL mode of the microcontroller by a hardware reset. When the BSL is active, the controller stays listening till the PC sends a zero byte

Typing in the command „connect“, the server establishes a connection by executing the connection procedure [Fig.]. After the target has received the zero byte, the BSL sends back an identification byte characteristic for each C166 derivative, see table..

Table 1:
ID-Byte overview

Chip	ID-Byte
C165	B5h
80C166	55h
C167 Step AD	A5h
C167 Step BA	C5h
C167CR	C5h

In return, the BSL is expecting 32 Bytes of code and data. On the host side you start this initialization sequence by the connect command that sends the zero byte, checks the identification byte and provides the 32 Bytes user software, the second loader, the hex-loader, and the shell.

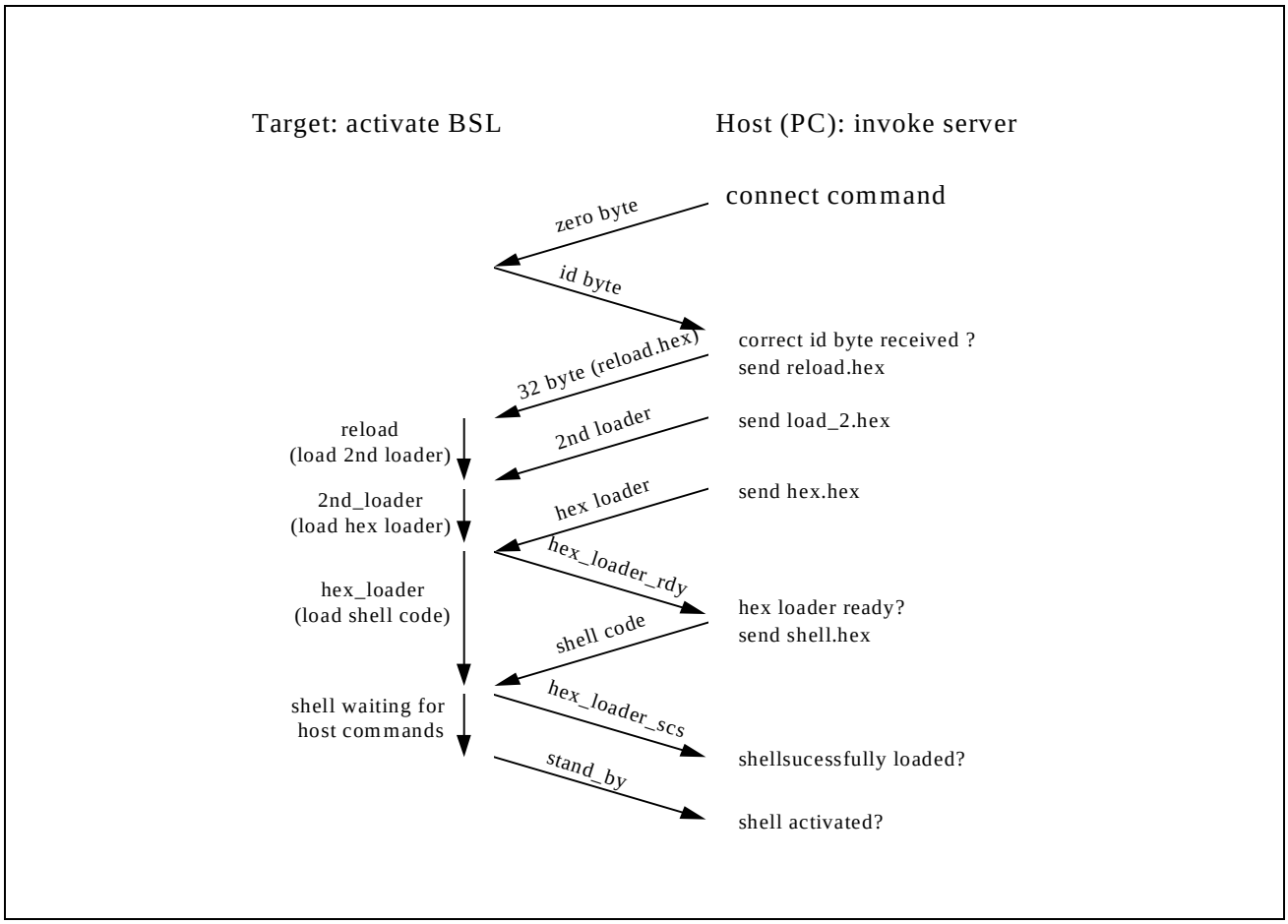

Figure 2:
Connection process

This initialization phase is terminated by the stand by message of the hex-loader and the shell. Afterwards both reside in the internal RAM. The server indicates this state by printing „stand by“ and prompting for new commands.

Now you can enter specific commands for instance „erase“ and „program“ the flash memory. In both cases the host issues a command to the shell monitor running on the C167. The shell decodes the command, then the application code is downloaded by the hex loader and finally executed [Fig.]. Supposing a further command is entered, the code will be again downloaded from the host into the internal RAM. Thus, the running system comprises three parts, the shell interpreting user commands, the hex-loader for downloading the code and data, and the code for the present command to be executed.

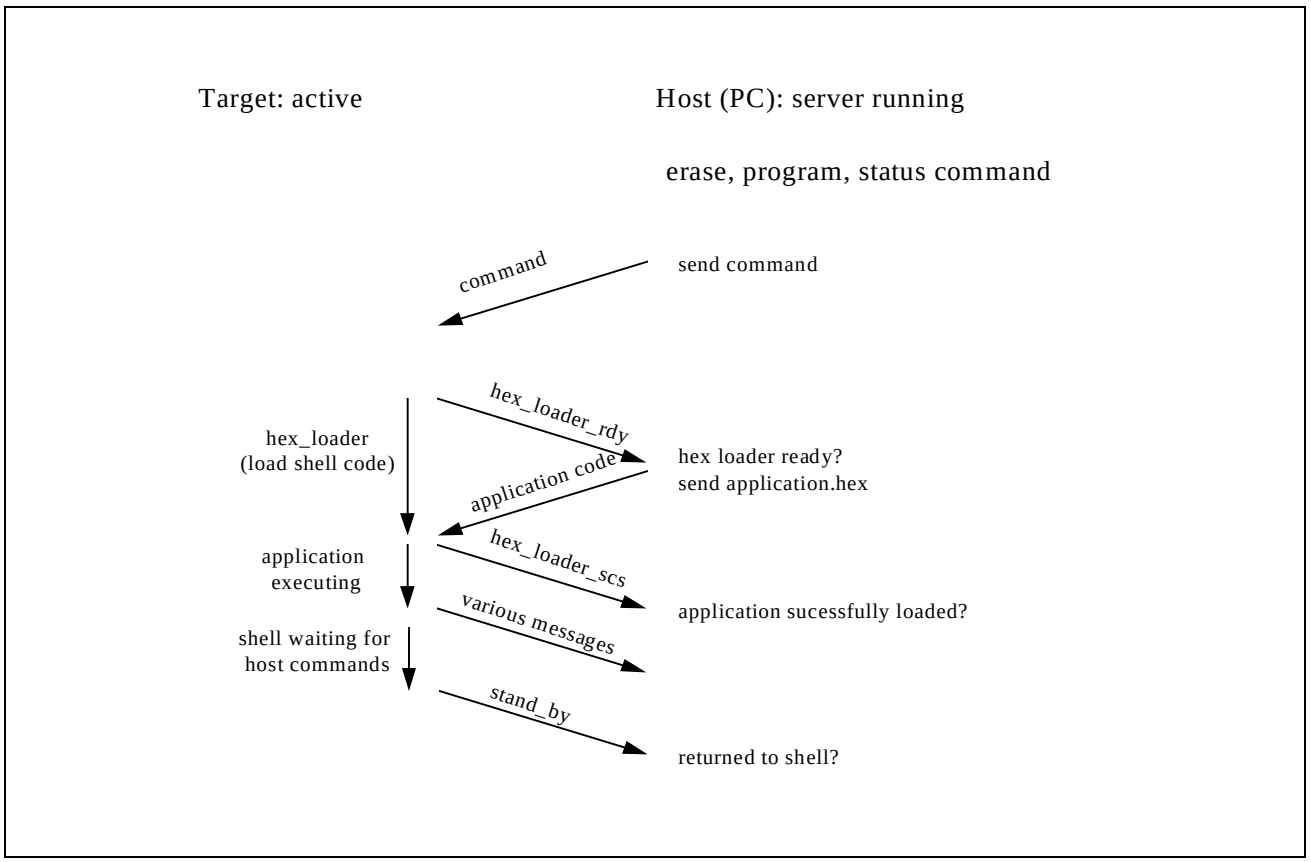

Figure 3:
Processing a command

Due to the limited RAM size the previous application code will be overwritten. Below you will find a memory map depicting the memory utilization. It shows that the hexloader code overwrites the 2nd loader. The reload code transferred by the BSL, is replaced by the wait code and the unlock sequence. The status, the erase, and the programming algorithm reside in the area above the hexloader .

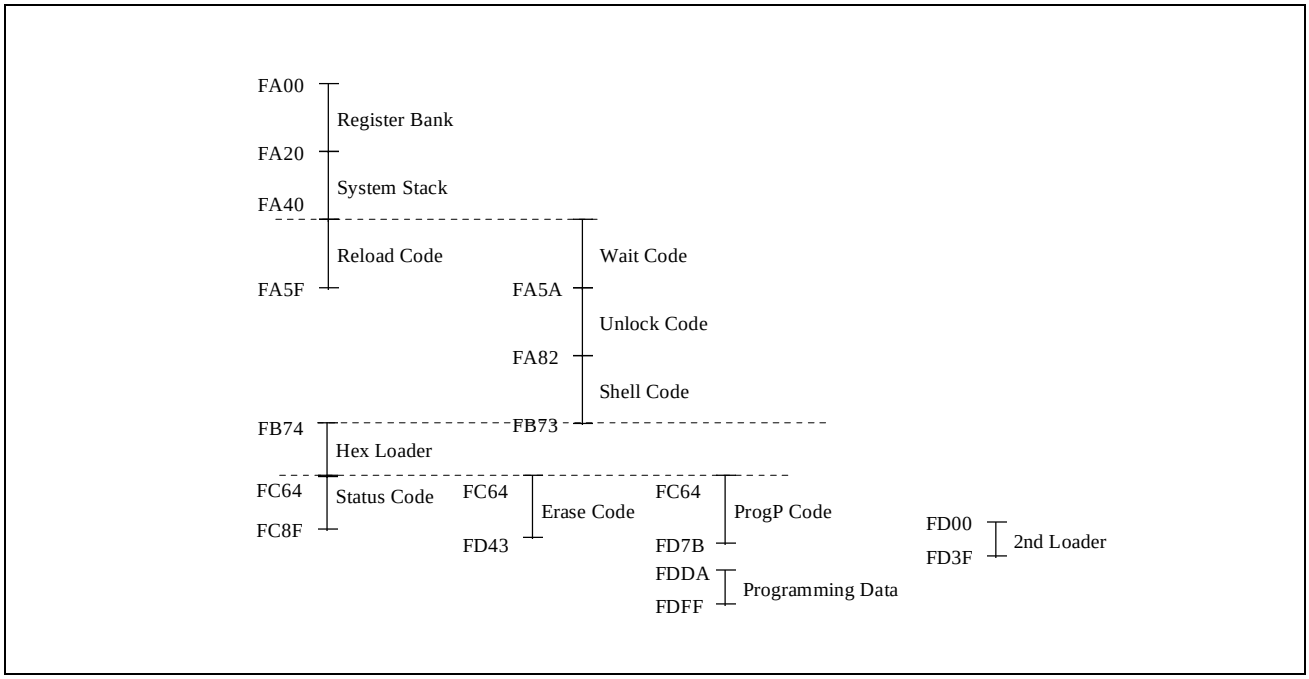

Figure 4:
Memory layout

3 Hex Loader

The hex loader on the target site has the task of transferring data to the internal RAM. It interprets the Intel hex file format. Furthermore, it distinguishes between application code executed to program the flash memory and users' programming data. The application code is a means to program the device and to maintain the host target communication. The programming data is intended to be stored in the Flash memory. According to Intels hex file format, programming data has the record type 0. The record type 2 is reserved the for extended address records in order to enable the addressing of more the 64 Kbytes. In order to distinguish code from data records, we have defined the record type 4 for code.

If the hex loader detects programming data, one data record is transmitted and stored in the data buffer provided for programming data. Afterwards the programming algorithm places the data at the location indicated by the corresponding memory address. In contrast, application code will be stored into the internal RAM, where it is executed then.

4 Status

Assigned to the flash memory is the Flash Control Register (FCR) providing status information and control flags. The status register contains 11 bits of information:

```
FWE       Flash Write Enable Bit
FEE       Flash Erase Enable Bit
FBUSY     Flash Busy Bit
RPROT     Flash Read Protection Activation Bit
FCVPP     Flash Control Vpp Bit
VPPREV    Flash Revelation Bit
CKCTL     Internal Flash Timer Clock Control
WDWW      Word/ Double Word Writing Bit
BE        Bank Erase Select (2 bits)
FWSET     Flash Writing Mode Set Bit
```

With the command „status“ the present configuration and the status of the Flash memory can be inspected. The successful completion is indicated by returning to the stand by mode of the shell. To inspect the status register, the file status.hex is downloaded into the target. It contains the status register inspection routine, the routine to unlock the Flash register and a wait routine. Finally, the controller transmits the content of the FCR to the host where it is displayed.

Actually, the FCR is a virtual register mapped into the active address space of the Flash memory while the Flash writing mode is active. In order to enable access to the FCR, the writing mode must be entered by means of the unlock sequence:

```
mov   Rn,#4000          ; Initialize Flash pointer
mov   DPP1:POF FCR, Rn  ; First pass of unlock sequence
mov   [Rn], Rn          ; Second pass of unlock sequence
```

After waiting 10us, the FCR can be retrieved by any direct access to an even address in the active address space of the Flash memory. Writing mode is quitted again by the key instruction

`mov MEM, Rn.`

`MEM` denotes a memory location within the Flash memory and Rn denotes an arbitrary register.

5 Erase

The Flash memory is partitioned into four memory banks of 48 Kbytes, 48 Kbytes, 24 Kbytes, and 8 Kbytes. They can be individually programmed and erased at least 1000 times. After the memory has been erased it contains only ones. Four commands are provided (erase0, erase1, erase2, and erase3) to erase specific memory banks.

The erasing algorithm starts programming by unlocking the Flash memory, then setting the FCR register. Afterwards the memory content is deleted by setting its content to zero to avoid that memory is damaged by overerasing. Finally, the memory is erased by the key instruction

`mov[Rn],Rn.`

During erasing, ensure that register Rn is using a data page pointer that is initialized with 5. Register Rn does not need to point to the bank that is going to be erased. To check if erasing has been successfully, the verify mode is encountered:.

```
cmp  DATAWR1,[FLASHPOINT]    ; First step of VM read
call Wait4                   ; Wait 4us
cmp  DATAWR1,[FLASHPOINT]    ; Second step of VM read
```

Register DATAWR1 is a general purpose register initialized with ones. The ONES register provided by the C166 family, cannot be used because the compare instruction only goes with general purpose registers. The FLASHPOINT register points to the Flash memory location to be checked. Please keep in mind that indirect address pointers are only possible using the registers R0 to R3 when employing arithmetic, logical or compare instruction.

Supposing the memory bank has been successfully erased, you will receive the message „bankX successfully erased“, and the target again enters the stand-by mode of the shell. If you instead receive the message „ERROR ! Setzero not successfully!“, the flash memory has been damaged so that it cannot be programmed anymore. The error message „ERROR ! Erasing not successfully !“ indicates that memory has been successfully programmed with zeros. However, it cannot be erased.

The time it takes to erase the flash memory is not fixed. It depends on the number of erasing pulses required until the very last memory cell is erased. It has shown that the erasing time varies between 1s for the smallest bank and the 11s for the largest bank.

6 Programming

Before you can start programming a specific memory bank, please ensure that the bank is already erased. If it is not erased, you will receive an error message, telling you that the device cannot be programmed.

The command „program“ invokes the programming routine on the host side. Firstly, it prompts for an Intel hex file that contains the user data to be written into the Flash memory. Then it downloads code and data into the target. Afterwards, the flash memory is unlocked and the programming algorithm invokes the hexloader to download the

programming data. The host transmits the data partitioned into records of 32 bytes to the target. At the target site the programming routine writes the data into the flash memory by an indirect move:

```
mov [FLASHPOINT], DATAWR ; Programming: write data to the Flash
```

FLASHPOINT points to the destination location and the register DATAWR contains one data word.

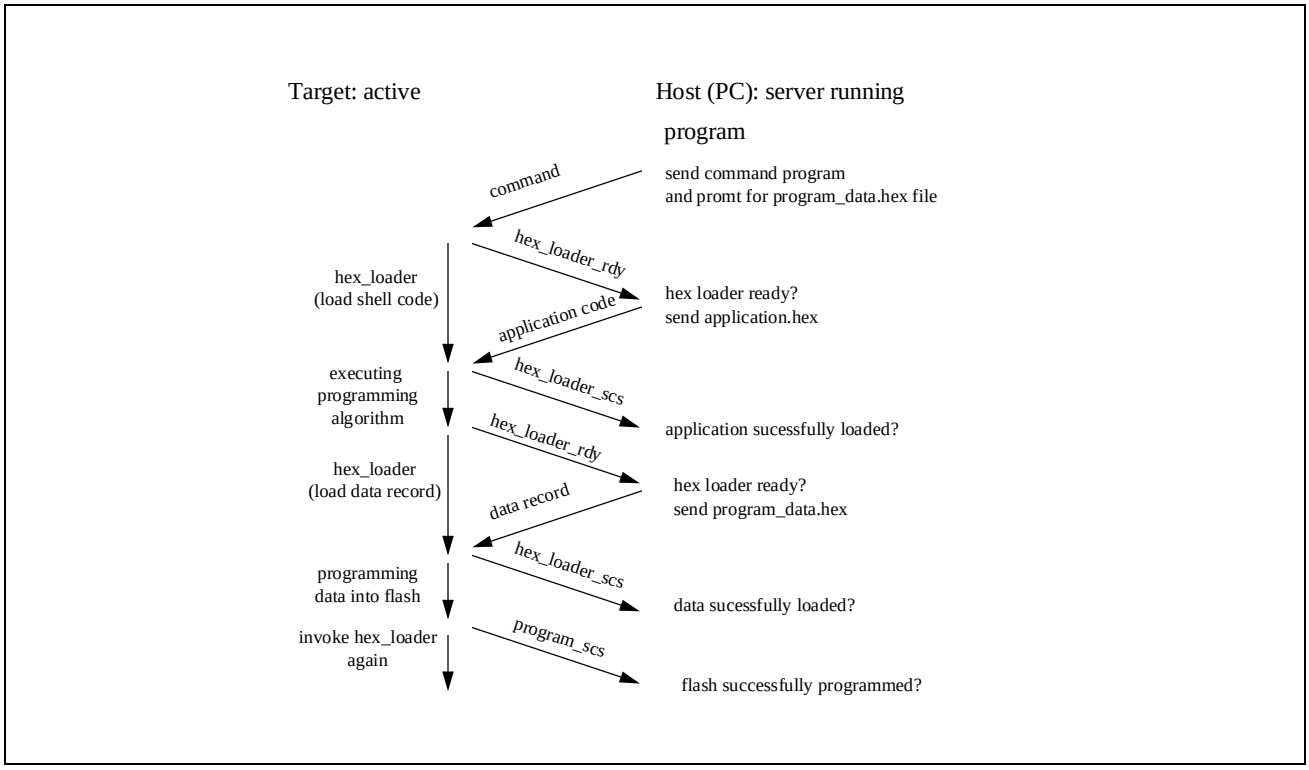

Figure 5:
Memory layout

After a data record has been programmed the verify mode is encountered. The verify instruction sequence is the same as used to verify erasing.

To speed up the programming, the controller provides a double word programming mode controlled by the FCR register, besides the word programming mode used above. It can be employed if the data is aligned to even addresses by performing the above move operation twice, without changing the destination pointer FLASHPOINT. However, the programming speed is limited by the serial communication link. Measurements have shown that a programming speed of 300 Bytes/s can be achieved by this approach.