Cheetah-12 RS232 Bootloader/Downloader for the MC9S12 Processor

This product is sold under license to the purchaser to be used without royalty cost within the purchaser's end product. The sale of this product to the purchaser does not constitute license to freely distribute the source code which remains property of Zanthic Technologies Inc.

Zanthic Technologies Inc. reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Although the information in this document has been carefully reviewed and is believed to be reliable, Zanthic Technologies Inc. does not assume any liability arising out of the application or use of any product or circuit described herein. Zanthic Technologies Inc. products are not authorized for use as components in life support devices wherein a failure or malfunction of the component may directly threaten life or injury. Should Buyer purchase or use Zanthic Technologies Inc. products for any such intended or unauthorized application, Buyer shall indemnify and hold Zanthic Technologies Inc. and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury, death or damage associated with such unintended or unauthorized use, even if such claim alleges that Zanthic Technologies Inc. was negligent regarding the design or manufacture of the part.

Zanthic Technologies Inc. makes no representations about the suitability of this protocol for any purpose. It is provided "as is" without express or implied warranty.

Introduction:

The Cheetah-12 RS232 bootloader is a small program that resides at the top of memory within a HCS12 processor. This program will allow a user program to be downloaded across a RS232 connection from a PC at 38400 baud. A downloading program for the PC is provided (with source code) to allow an S-Record file to be chosen and sent one line at a time through a serial port. Each line of the S-Record is responded to with a success or failure notice. Additional commands are used to erase flash and start the user program.

At reset, the bootloader program is run first. In order to run the user program, three criteria have to be met

1) The memory location at 0x1000 must NOT contain the value 0x55AA
2) The user program reset vector at 0xEFFE,F must contain something other than the value 0xFFFF
3) The CRC value at 0xEF7E,F is compared to memory and must pass.

If all tests pass, the Cheetah bootloader will jump to the user program and will not be active until the next time the processor is reset or until the user program chooses to jump back to the bootloader. Note that after the bootloader gives control to the user program, no resources are used by the bootloader. This is where the flag located at 0x1000 comes into play. In order for an existing user program to allow a new firmware program to be uploaded, the user program must place the value 0x55AA at location 0x1000 and then re-enter the bootloader. More details and sample code are given below.

User program interrupts are handled by the creation of a second interrupt table, identical to the one normally used but located within the user program space at 0xEF80-0xEFFF instead of the usual 0xFF80-0xFFFF. This requires one addition jump instruction that is executed when an interrupt occurs. This jump is handled within the Cheetah firmware and is transparent to the user program. The user program can simple remap where the vector table goes with a setting of 0xEF80 instead of 0xFF80.

Included in this package:

1) PC Cheetah RS232 program with VB6 source code (allows the download of S19 files through the RS232 port
2) Cheetah12 bootloader for HC12 with source code in C (compiled with Imagecraft compiler)
3) Sample user program. Demonstrates how a user program would be written to work under the Cheetah boot-loader and includes a sample interrupt function. Compiled under Imagecraft ICC12 compiler.
4) Documentation

Not included in this package:

1) Suitable compiler for the HC12 for developing user programs or for making changes to the bootloader
2) VB6 for recompiling PC source code if changes are required.

Limitations:

1) Source code for the HC12 Cheetah Bootloader was compiled under Imagecraft ICC12 compiler. Compiler changes are left as an exercise for the user.
2) The RS232 port speed is fixed at 38400 as it is impossible to guess as to the method of choosing baud rates on the final HC12 board. This was done to minimize hardware dependence.

Downloading S-Records

S-Records are an ASCII text readable file that contains the machine code for the processor you are downloading to. The basic format of an S-Record is
SxNNAAAADDDD….CC<CR>
Where
Sx is the S record type S0, S1,S2,S8, S9 etc
NN is the number of bytes to come in this line
AAAA is the address of the code to be saved (variences shown below)
DDDD is the data
CC is the checksum
<CR> is a carriage return/line feed

A sample line might be
S104C02A3DD4
Where this is a S1 type, with 4 bytes (including the checksum), address C02A, Data 3D, and checksum D4 (add all the bytes, length, address and data and take one's complement)
A longer line might look like
S111C02B34B7751B9EC63F7B003B1C0038805B
This is 11 (hex) bytes, address C02B, data 34, B7, 75 etc and checksum 5B

S-Record Types:

The Cheetah12-RS232 bootloader supports the following S record types
S0 – ignored, but responded to
S1 – 16 bit address
S2 – 24 bit address (more on this below)
S8 – end of S-record, generate memory CRC
S9 – end of S-record, generate memory CRC

Paged memory support, banked or linear

There are two main ways of formatting data in an S-record to allow writes to the paged memory of the HC12, banked and linear. This bootloader supports the banked method, which means the S2 record gives the page number and the address. For example, to write to page 0x30 at address 0x8000, the S record would look like
S2NNPPAAAADDDD…
NN is the number of bytes to come in this line
PP is the page number, 0x30 in our example
AAAA is the address, 0x8000 in our example
DD is the data

Data length formatting

The data being stored to flash memory must be in 16bit words. If your compiler does not generate 16bit whole words per each S-record line, you may have to use a conversion program like SRecCVT. This program is available free of charge off the internet.

Additional Commands

In order to control other functions within the bootloader, addition commands were created. These commands are referred to as 'C' commands, because, like the S-records use the 'S' character to start the line, these commands start with 'C' (makes sense, doesn't it?) The bootloader will respond with a lower case 'c' according to the following:

"C1" ping the HC12 to see if it is there.
"c2" pong response
"C3" erase flash: Takes 5-15 seconds for response, depending on flash memory
"c4" flash erased
"C5" generate CRC: Takes 3 seconds for response
"c6" CRC generated
"C7" Restart user program. Note that there is no response to C7

Note that the CRC is generated automatically at the end of a S-Record download with the reception of either the S8 or S9 record. The extra 'C' command to generate the CRC is provided as a test command.
These commands can be typed into an RS232 terminal program with the carriage return to terminate the line. This can be used for testing if required.

Program Integrity Check:

At the completion of an S-Record download, with the reception of a S8 or S9 type of record, the bootloader will generate a CRC checksum through the following pages of memory: 0x4000-0x7FFF, 0x8000-0xBFFF pages 30 to 3D, and 0xC000-0xEFFF, with one exception, the 16 bit location 0xEF7E,F is not included as this is where the CRC result is stored. The bootloader source code will have to be modified if your particular version of the HC12 has a different flash memory structure. When the bootloader starts, one of the checks is to verify the user program integrity by rechecking the CRC. This takes about 3 seconds at boot-up time.

User program requirements

1) Interrupt vectors will be located in memory at 0xEF80-0xEFFF instead of the usual 0xFF80-0xFFFF
2) User RAM should be set to start at 0x1002 instead of 0x1000. This is explained under "Restarting the bootloader"
3) Memory locations 0xEF7E,F are used to store the 16 bit CRC value. This location is located just below the user vector table and should not be used for the program storage itself as the CRC will be generated there automatically.
4) User program S-Records must be formatted to 16 bit word sizes. A program such as SRecCvt.exe can be used to format the S-Records appropriately.

Restarting the boot-loader from within a running program:

There needs to be some mechanism for the user to enter into the bootloader mode after a valid user program is running. This can be accomplished in two different ways. The first way is to save the value 0x55AA to location 0x1000 and then jump directly into the bootloader by jumping to the address located at the HC12 reset vector (0xFFFE,F) which would normally point to 0xF000, the start of the bootloader program. (There is also a hard coded routine at location 0xFDF2 that will jump to the reset vector when called, see program code below) The problem with this approach is that if any interrupts are still active, they will continue to fire which can lead to issues, both with the fact that the bootloader has re-initialized RAM and that as soon as the bootloader runs the user may erase the flash memory, which, of course, includes the user program interrupt vectors and code. If this approach is taken, care must be exercised in shutting down all interrupts prior to entering the bootloader.

The second approach is to use the COP timer to force a reset of the processor which effectively disables all interrupts. The use program would contain two sections of code, one that would cause a COP timeout (this assumes, of course, that the COP timer was enabled to start with) and the COP timeout function, within the user program, that would decide whether this is a COP timeout or a request to enter the bootloader. Sample code as follows.

Note: The Cheetah12 bootloader does not have the COP timer enabled by default. See source code for comments.

```
#pragma interrupt_handler COP_Handler
void COP_Handler(void)
{
if (*(unsigned int *)0x1000 != 0x55AA)// if not re-entering the
bootloader
 {
                                    // run code here if this is a real
                                    // COP timeout.  If you want, you
                                    // could clear the flag at 0x1000
                                    // and run the reset, the same as
                                    // below
 }
else                                // jump to the restart with the
 {
  asm ("jmp $FDF2");     // We can't make a jump to FFFE as that is a
 }                       // vector address but we can make a jump to the
                         // function in the Cheetah bootloader that will
                         // then jump to the restart function (jump
                         // function is hardcoded to FDF2)
}
```

At this point the processor will be restarting with the value in RAM intact. At the beginning of the Cheetah program, where it checks for the dip switch and the presence of the user vector, the program would also check for the presence of the value 0x55AA at the 0x1000 location. If the value is present than it would stay in the boot loader program and not run the user program.

Files located on distribution disk

Directories

- Documentation (contains this file)
- HC12 executables (contains S19 file for bootloader)
- PC Install (contains install program for PC downloader)
- Source code
- HC12 (source code for HC12 Cheetah bootloader)
- Sample user program (test program)
- Code Warrior (conversion notes)
- PC (source code for VB6 download program)

Converting bootloader to compile under Codewarrior

There are only a couple of differences in converting the existing C source code to compile under the Code Warrior compiler. Please refer to the directory Source code/HC12/CodeWarrior for notes and sample code