

Getting started with the Freescale HCS12 family using CodeWarrior

Application Note AN1002

Author	Frank Voorburg
Date	13-Feb-06
Version	1.02
Status	Released
Abstract	The goal of this application note is to help you get started with software development for your Freescale HCS12 family microcontroller using Metrowerks CodeWarrior development environment. I will describe three commonly used approaches ranging from low cost software development using the D-BUG12 serial monitor to a more powerful configuration using P&E's BDM MultiLink background debug mode (BDM) interface to allow source level debugging.

Table of Contents

1	Overview.....	3
1.1	Document History	3
1.2	Introduction	3
2	Software Development in RAM using D-BUG12.....	5
2.1	Step 1 – Building the S-record	5
2.2	Step 2 – Downloading using D-BUG12	6
2.3	Step 3 – Running the example application	7
2.4	A closer look.....	7
2.4.1	Project files.....	7
2.4.2	Memory configuration.....	7
2.4.3	Entry point.....	8
2.4.4	Limitations	8
3	Software Development in ROM using P&E's BDM MultiLink	10
3.1	Step 1 – Building the executable	10
3.2	Step 2 – Flash programming using BDM MultiLink.....	11
3.3	Step 3 – Running and debugging the example application.....	11
3.4	A closer look.....	12
3.4.1	Project files.....	12
3.4.2	Memory configuration.....	13
3.4.3	Limitations	14
4	Software Development in RAM using P&E's BDM MultiLink.....	15
4.1	Flash programming the interrupt vector jump table using BDM MultiLink	15

4.2	Step 1 – Building the executable	15
4.3	Step 2 – Downloading using BDM MultiLink	16
4.4	Step 3 – Running and debugging the example application.....	16
4.5	A closer look.....	17
4.5.1	Project files.....	17
4.5.2	Memory configuration.....	17
4.5.3	Vector tables.....	17
4.5.4	Limitations	18
5	Conclusion	19
6	Contact	20
	References.....	21

1 Overview

1.1 Document History

<i>Version</i>	<i>Date</i>	<i>Author</i>	<i>Description</i>
1.00	16-Jun-03	Frank Voorburg	Creation
1.01	30-Dec-03	Frank Voorburg	Support for CodeWarrior HC12 version 3.0
1.02	26-Jun-05	Frank Voorburg	Support for CodeWarrior HC12 version 3.1

1.2 Introduction

The goal of this application note is to help you get started with software development for your Freescale (formerly known as Motorola) HCS12 family microcontroller using Metrowerks CodeWarrior development environment. I will describe three commonly used approaches ranging from low cost software development using the D-BUG12 serial monitor to a more powerful configuration using P&E's BDM MultiLink background debug mode (BDM) interface to allow source level debugging. For each approach, I will explain how to get an example application running on your microcontroller board by going through a few simple steps. Once completed, we will look into how it all works so you can start using it for your own projects as soon as possible.

From my own experience I know that it can be overwhelming at first to get into microcontroller programming. Especially if you're looking for a low cost development environment, you have to do a significant amount of research and studying to find out what you need and how it all fits in the big picture. When creating this application note, I did this research and studying for you and together with a handful of simple examples, I will explain to you how to get started quickly. Using this application note I hope to spark your interest and increase your enthusiasm for microcontrollers.

The software development environment that is used in this application note is Metrowerks CodeWarrior for HC12 targets version 3.1. For this release, a Special Edition is available for free, enabling students, hobbyists and professionals to use this high quality software package to develop their own HC12 applications in the C-language. The Special Edition can be downloaded from their website (<http://www.metrowerks.com>). The limitation of the Special Edition is that your software program can't be larger than 32 kilobytes.

Besides having Metrowerks CodeWarrior for the HC12 installed on your computer, you will also need a Freescale HCS12DP256 based evaluation board with either the D-BUG12 monitor programmed in the internal flash or the BDM MultiLink interface from P&E Micro (<http://www.pemicro.com>) to be able to use some of the examples.

Each described software development approach is located in a separate section and has the same structure. First an overview of the approach is given, followed by a step-by-step explanation on how to get the example application running. After this, a more in-depth description is given. All the example applications for the HCS12DP256 are the same in functionality. They toggle an LED, connected to an output port of the microcontroller, on and off every 1 second. The evaluation board I used for this was the MiniDRAGON from EVBplus (<http://www.evbplus.com>), which has an external crystal of 16MHz and an LED (actually, a 7-segment display) connected to PORTH that I use in the example applications. It is not necessary to have the same evaluation board. If you

have a different evaluation board, just keep in mind that you might have to modify the examples if you have an LED connected to a different port pin and that the blink rate might not be 1 second if you have a different external crystal frequency.

2 Software Development in RAM using D-BUG12

In this configuration, no additional development hardware is required besides an HCS12DP256 based evaluation board. It does however assume that a monitor program called D-BUG12 is present in the flash memory of the HCS12DP256. D-BUG12 is a popular monitor program written by Gordon Doughman, who in my eyes is the HC12 guru at Freescale. Because of its ease of use and powerful features, most HCS12 evaluation board manufacturers pre-program this monitor program into the flash memory before shipping it to you. In case you're still looking for an HCS12DP256 evaluation board and don't want to purchase additional hardware for your software development, pay attention that D-BUG12 is resident in the flash memory of the evaluation board, so you can use the low cost approach for software development as described in this section of the application note.

D-BUG12 allows you to communicate with the evaluation board by using a simple terminal program. In order to use D-BUG12, you have to connect your evaluation board to one of your PC's COM ports using a serial cable. If your evaluation board ships with D-BUG12 pre-programmed, chances are that this serial cable is part of the shipment. Under MS Windows, you can use the terminal program HyperTerminal, which comes for free with MS Windows, or you can use any other terminal program. The one I prefer is called MiniIDE and can be downloaded from <http://www.mgtek.com/miniide/>. This is the terminal program you will see in the screenshots in this document.

All sounds good, huh? Great low cost solution and no additional development hardware required. You might wonder, what's the catch? Well, the catch is that D-BUG12 only allows you to download and debug your software applications in RAM and/or EEPROM, not into flash memory. Is this a problem? Nope, because the HCS12DP256 has 12 kilobytes of RAM available and this is plenty for developing your first software programs. Besides this we also don't have to worry about the interrupt vectors. D-BUG12 reroutes all interrupt vectors to RAM locations, meaning interrupts can still be used.

2.1 Step 1 – Building the S-record

First the example project should be opened up in CodeWarrior. To do this, find the file "Hcs12dp256.mcp" in the ".\Example1"-subfolder and double-click it. This should launch CodeWarrior and open up the example project. Before the example application can be downloaded to your board, the source files need to be compiled into object files and then the object files need to be linked together to create the so-called S-Record. This is easier than it sounds. In CodeWarrior, click on the green "Play"-button:

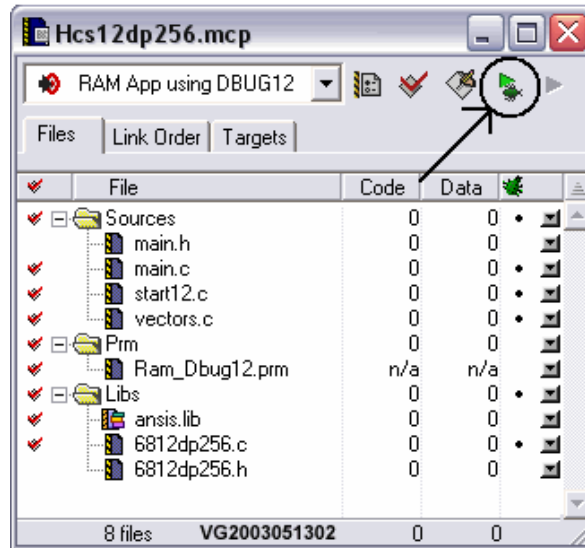


Figure 1 - Building the S-record

After this you should see that a file called “Hcs12dp256.s19” appeared in the “bin”-subdirectory. This file is the S-record that I mentioned earlier.

2.2 Step 2 – Downloading using D-BUG12

You are now ready to download the S-record “Hcs12dp256.s19” to your board. I will demonstrate this using MiniIDE. Note that any other terminal program can be used aswell. Launch MiniIDE and connect to the terminal. Now reset your evaluation board and verify that you see the D-Bug12 prompt. At the prompt type “LOAD” followed by an <enter> and then select to download the “Hcs12dp256.s19” file.

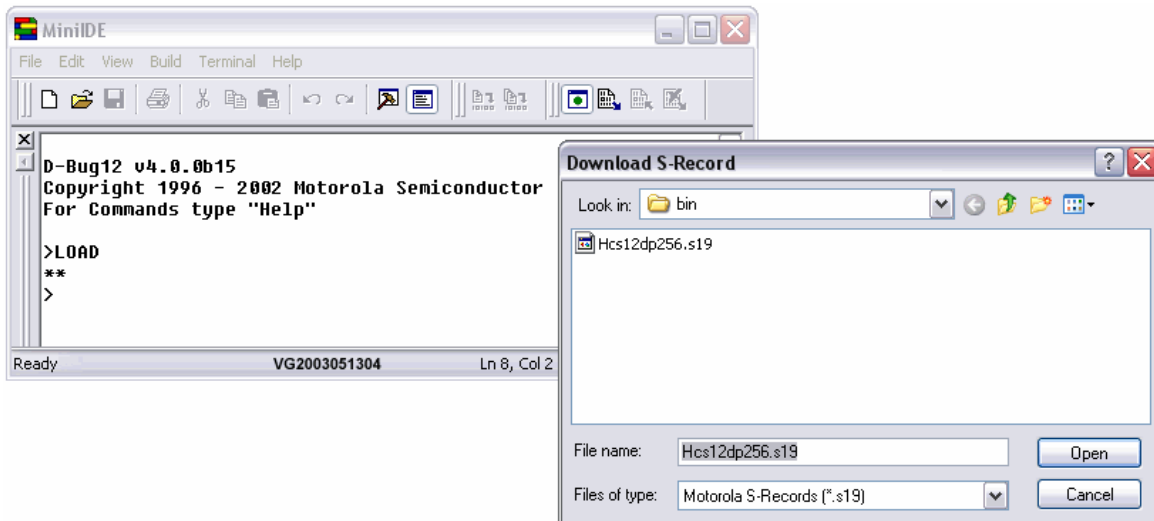


Figure 2 - Downloading using D-BUG12

2.3 Step 3 – Running the example application

The example application is now downloaded to RAM and is waiting there for you to give the command to start running. Type in “G 1829” followed by an <enter> to start the example application. You should see the LED toggle on and off every second. That’s it!

2.4 A closer look

2.4.1 Project files

Figure 3 provides an overview of the files that are in the project for the example application. The actual example application is implemented in “main.c” and “main.h”. “vectors.c” contains a table with elements that are linked to the interrupt vectors (remapped by D-BUG12 to RAM). “start12.c” is a compiler specific file that makes sure the microcontroller is initialized before function main() is called.

File	Code	Data
Sources	230	24
main.h	0	0
main.c	45	1
start12.c	57	23
vectors.c	128	0
Prm	0	0
Ram_Dbug12.prm	n/a	n/a
Libs	40K	2K
ansis.lib	41782	2015
6812dp256.c	0	611
6812dp256.h	0	0
Total	41K	2K

Figure 3 - Project files

The “Ram_Dbug12”.prm file is used to explain to the CodeWarrior linker where the compiled code should be located in the memory of the HCS12. More about this in the next section. The compiler libraries are also specific for the CodeWarrior compiler. “6812dp256.h” (of “mc9s12dp256.h” for newer versions of CodeWarrior) for example, contains definitions for all the HCS12 registers on your HCS12DP256 based evaluation board. Just keep them in your project and they’ll make your life as a software developer easier.

2.4.2 Memory configuration

CodeWarrior for the HC12 can be used for any HC12 derivative. You are using the HCS12DP256 with the DDebug12 program installed in the flash memory. DDebug12 allows you to download your applications (S-records) into the RAM memory and allows you to do some simple debugging. How the board is configured has to be explained to CodeWarrior, otherwise it would create an invalid S-record. This configuration is done through the “Ram_Dbug12.prm” file. I configured it for you in the following way:

Type	Start (hex)	End (hex)	Size (bytes)
RAM	0x1000	0x16FF	1792
Stack	0x1700	0x17FF	256
ROM	0x1800	0x3BFF	9216
Vectors ¹	0x3E00	0x3E7F	128

I would recommend that if you are new to software development on the HCS12DP256, that you keep these settings for now. You can of course modify these to your liking, as long as you don't cross the physical boundaries of the RAM on the HCS12DP256. Keep in mind that D-BUG12 uses a small amount of RAM between 0x3E80 and 0x3FFF that you shouldn't use. Also keep in mind that these memory types refer to sections read by the linker. Physically, the ROM section is also in RAM on the HCS12DP256 in this configuration.

2.4.3 Entry point

After downloading your application you need to tell D-BUG12 what the start address of your application is if you want to run it. The start of your application is always the address of the function “_Startup”. The address of this function can be found in the so-called MAP file.

```
*****
STARTUP SECTION                                     VG2003051306
-----
Entry point : 0x1829 (_Startup)
_startupData is allocated at 0x1839 and uses 23 Bytes
extern struct _tagstartup {
  unsigned flags      0
  _PFunc   main      0x1856   (main)
}
```

Figure 4 - Start address in MAP file

This is a report generated by the CodeWarrior linker that you can use as a reference to find out where the linker located your variables and functions. For the example project, the MAP file is called “Hcs12dp256.map” and can be found in the “bin”-subdirectory.

2.4.4 Limitations

Two limitations exist when it comes to using CodeWarrior for an evaluation board that has D-BUG12. The first one is that the source level debugger cannot be used, because unfortunately there is no support for communication between CodeWarrior's debugger and D-BUG12 (in both EVB and POD mode).

The second limitation is that the S-record produced by CodeWarrior's linker is valid, but cannot be used by D-BUG12. The first line in an S-Record starts with “S0”, followed by more characters. This “S0”-line contains so-called header information. The header information that CodeWarrior adds to the S-Record is sometimes too long for D-BUG12 to interpret. For this reason I created a small command line converter program called “SRecChgHdr.exe” that is located in the “cmd”-subfolder. This program reads the S-Record generated by CodeWarrior's linker, replaces the “S0”-line with a shorter one, and then writes it to a different output file.

It is possible to have CodeWarrior automatically call this program if you configure your project settings similar to what is shown in the illustration below. The *.sx file is the S-Record generated

¹ Configured in “vectors.c”.

by CodeWarrior and the *.s19 file is the S-Record generated by my SRecChgHdr.exe program and can be used for downloading using DBUG12. Note that if you change the name of the *.sx file generated by CodeWarrior, you must manually adjust this in your project settings as well.

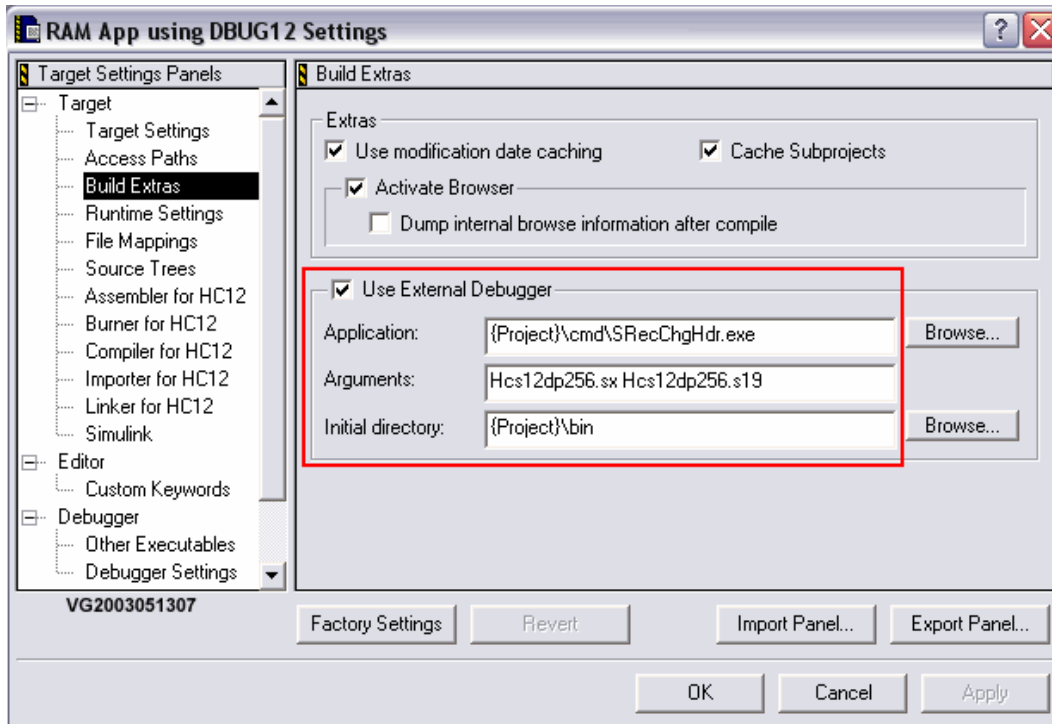


Figure 5 - Invoking the converter program automatically

3 Software Development in ROM using P&E's BDM MultiLink

As mentioned in the previous chapter, using the D-BUG12 serial monitor has some limitations. You can't program your application in the flash memory of the HCS12DP256 and you can't use the source level debugger that comes with the Metrowerks CodeWarrior development environment. If these limitations bother you and you are able to invest in a background debug mode (BDM) interface, then the configuration described in this chapter is of interest to you.

The configuration is as follows. Instead of using a serial cable to connect your evaluation board to your PC, you use P&E Micro's BDM MultiLink (<http://www.pemicro.com>) for the HC12 interface and connect it to the parallel port (LPT1 or 2) of your PC. If you prefer to use a USB port instead, you could also use their new USB MultiLink. You will also no longer use a terminal program but the full featured source level debugger that is part of the Metrowerks CodeWarrior development environment.

The main advantage of this approach is that debugging your software application becomes easier and more visual. You can set breakpoints on a line of C-source code, use step instructions, watch variable values, etc. In addition to this you also have access to all of the microcontroller's memory: RAM, EEPROM, and flash memory.

3.1 Step 1 – Building the executable

First the example project should be opened up in CodeWarrior. To do this, find the file "Hcs12dp256.mcp" in the ".\Example2"-subfolder and double-click it. This should launch CodeWarrior and open up the example project. Before the example application can be downloaded to your board, the source files need to be compiled into object files and then the object files need to be linked together to create the executable of your software application. In CodeWarrior, you do this using the "Make" command. You can find this in the "Project"-menu in CodeWarrior or you can click the button as illustration in Figure 6.

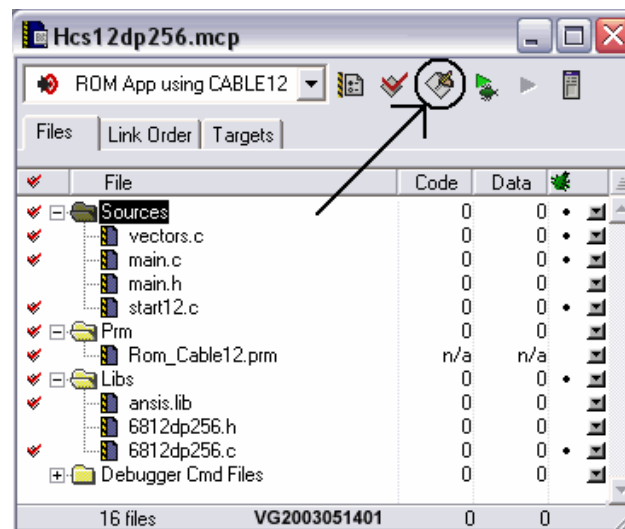


Figure 6 - Building the executable

After this you should see that a file called "Hcs12dp256.abs" appeared in the "bin"-subdirectory. This file is the executable, which we will program into the flash memory in the following step.

3.2 Step 2 – Flash programming using BDM MultiLink

The next step is to program your software application in the flash memory of your HCS12DP256 evaluation board. The source level debugger that is part of the Metrowerks CodeWarrior environment includes functionality to do flash programming when using the P&E Micro BDM MultiLink interface. This has to be configured in the project settings and through the usage of several external files (debugger command files). The good thing for you is that I have already done this for you in the example project. You only have to verify one thing. The actual flash erasing and programming requires certain delay times on the microcontroller while performing these operations. You don't have to do anything for this. However, the time reference for this delaying is derived from the crystal clock frequency of the microcontroller. The crystal clock frequency of the evaluation board I used in this example is 16MHz. If yours is different, you have to modify this in file "erase_unsecure_hcs12.cmd", which you can find the ".\cmd"-subfolder of the example. You have to change the line with "define CLKDIV 0x49", by changing it to a value as described in the comments above this line in the same file. This only has to be done once. To start with the actual flash programming using CodeWarrior's debugger, all you have to do is click on the green play-button as illustrated in Figure 7².

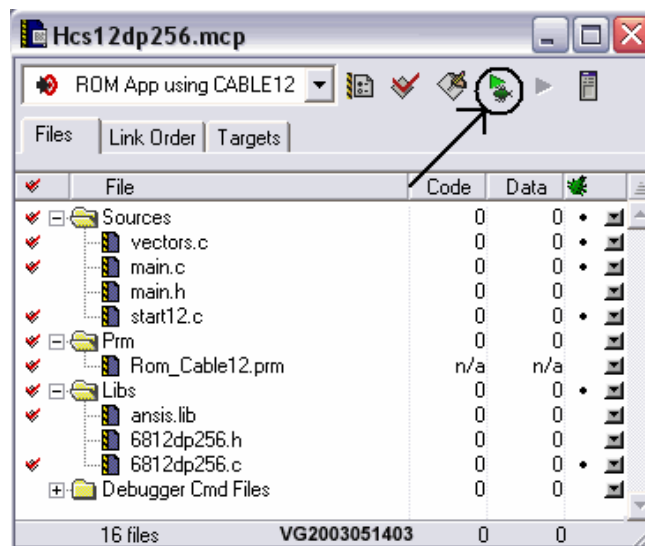


Figure 7 - Flash programming your application

3.3 Step 3 – Running and debugging the example application

At this point you should be in the debugger window and your software application is programmed into the flash memory of the microcontroller on your evaluation board. This means everything is ready to start running your software application. In order to do this, click the "RUN"-button on the debugger's toolbar (Figure 8).

² A word of caution before you start the flash programming. The entire flash memory will be erased. This means that if you have D-BUG12 programmed in the flash memory, this will be erased. If it turns out that for some reason you're having problems with flash programming your HCS12DP256 and the flash was already erased, this means you won't be able to use your microcontroller until you solve these problems.



Figure 8 - Running and stopping your software application

When using D-BUG12, the monitor program lost control of the software application once started. When using the debugger, this is luckily not the case. You can stop (and even start) your software as often as you want using the “STOP”-button.

If you want the debugger to automatically stop the software application for you when it’s at a certain point, you can use a breakpoint. For demonstration purposes, let’s assume that you want to stop the example software application every time the RTI cycle time elapses, which is every 4.096ms right before “rtiCnt” is incremented. This piece of code is in the “main.c” source file. If your software application is running, first stop it. If the source code shown in the source window of the debugger is not “main.c”, then use your mouse to right-click in the debugger’s source window and select “Open Source File...” from the popup menu and select “main.c”. Now use your mouse to right-click in the debugger’s source window on the line that says “rtiCnt++;”. This is where the code is located that increments the “rtiCnt” and this is where we want to set the breakpoint. From the popup menu select option “Set Breakpoint”. You should now see a red arrow in front of this line, which represents the breakpoint. In case you want to remove the breakpoint, you can follow the same steps and select “Delete Breakpoint” from the popup menu. Now hit the “RUN”-button and you will see that shortly after this, the debugger stopped the application. In the data window, the debugger automatically added a watch for the “rtiCnt”-variable. If you hit the F11-key on your keyboard, the debugger will do a Single Step operation, meaning that it will execute the source code on the line where to application currently is. Note that in the data window you can see that the value of the “rtiCnt”-variable incremented. These operations are the basics of debugging and will make your life as a software engineer a lot easier!

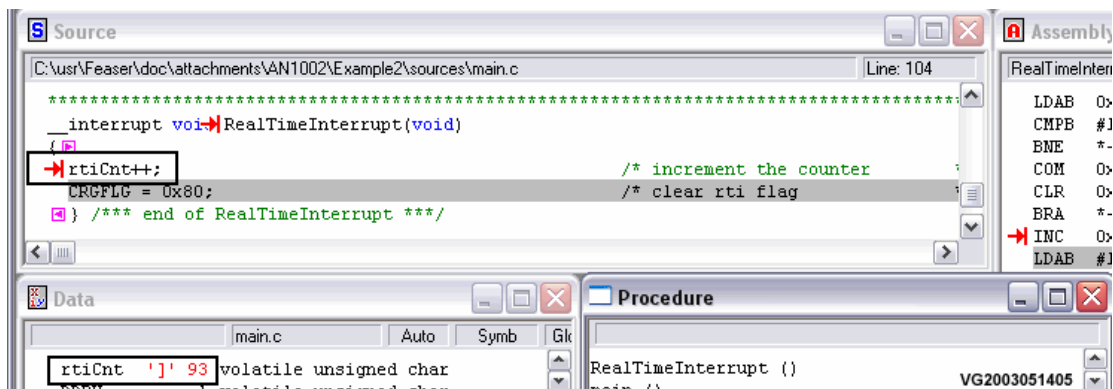


Figure 9 - Using breakpoints and variable watches

3.4 A closer look

3.4.1 Project files

The project files for this example are similar to the ones described in section 2.4.1, with the addition of the debugger command files. These files are used by the debugger to perform

microcontroller specific actions during the flash programming of your software application. In the example they are already setup for you. If you followed all the instructions in this chapter up to now, you won't have to touch these files.

The “erase_unsecure_hcs12.cmd” is by far the most important file. The flash of the HCS12 family microcontrollers can be secured and unsecured. When the flash is secured you won't be able to use BDM MultiLink to program the flash memory anymore. The flash is automatically secured when the entire flash is erased. In order to unsecure the flash memory again a certain value has to be written to a specific location in flash memory. This is automatically done for you by the “erase_unsecure_hcs12.cmd” file during the flash programming operation.

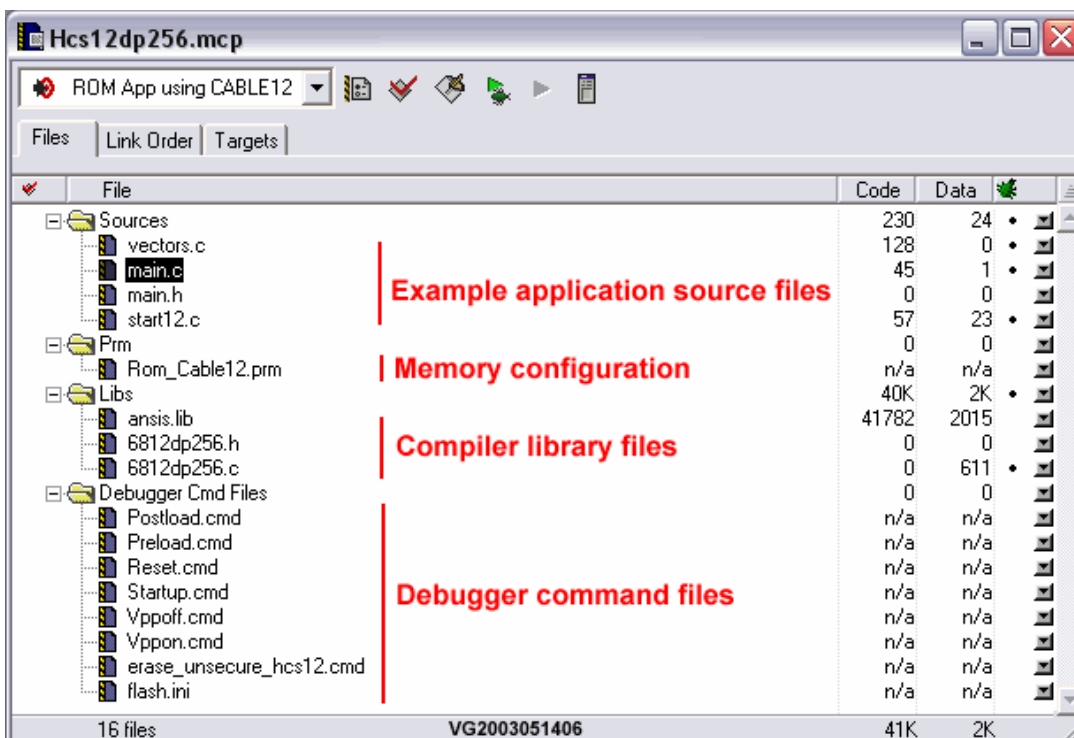


Figure 10 - Project files

3.4.2 Memory configuration

How the memory on your evaluation board is configured has to be explained to CodeWarrior, otherwise it would create an invalid executable. This configuration is done through the “Rom_Cable12.prm” file for this example. I configured it for you as show in the following table. I would recommend that if you are new to software development on the HCS12DP256, that you keep these settings for now. You can of course modify these to your liking, as long as you don't cross the boundaries of the RAM and the ROM.

Type	Start (hex)	End (hex)	Size (bytes)
RAM	0x1000	0x3EFF	12032
Stack	0x3F00	0x3FFF	256
ROM1	0x4000	0x7FFF	16384
ROM2	0xC000	0xFEFF	16128

Vectors ³	0xFF80	0xFFFF	128
----------------------	--------	--------	-----

Note that there is not just one ROM part, but two (ROM1 and ROM2). This is because I configured the example to use the unbanked flash pages on the HCS12DP256. There are two unbanked flash pages on the HCS12DP256, one starting at 0x4000 and one starting at 0xC000. The interrupt vectors are located at the end of the latter one.

3.4.3 Limitations

The approach described in this chapter has no limitations. However, the project configuration that I created for you in the example has one limitation. Only the unbanked flash pages can be used, meaning that your software application can't use more than 32 kilobytes of flash memory, even though the HCS12DP256 has 256 kilobytes of flash memory. I have done this on purpose, because it saves overhead of explaining the idea behind memory paging on the HCS12 family microcontrollers. This application note is also written from the point of view that the reader only has access to the Special Edition of Metrowerks CodeWarrior. This one has a ROM limitation of 32 kilobytes for your software application. Since this exactly fits on the 2 unbanked flash pages, there is no need to use memory paging. If you do prefer to use memory paging in your software application, you can still use this example as a starting point. Refer to CodeWarrior's documentation to find more information on how to use memory paging.

³ Configured in "vectors.c".

4 Software Development in RAM using P&E's BDM MultiLink

Using the configuration described in the previous chapter doesn't have any limitations. You can program the entire flash memory of the microcontroller. There is a downside to this. Programming the flash memory takes time. Using the configuration described in this chapter you'll develop your software application in RAM instead of flash memory, allowing you to skip the flash programming sequence every time and instead do a fast download to RAM. The limitation to this approach is that you're of course limited to the 12 kilobytes of available RAM memory on the HCS12DP256. Again, this should be plenty of memory for your first development projects.

Before using this configuration, you have to prepare the HCS12DP256 on your evaluation board. This preparation consists of programming an interrupt vector jump table into the flash memory once. This jump table redirects all interrupt vectors to a fixed location in RAM. Your software application can then use a special pseudo vector table that is downloaded to RAM together with your software application. More about the interrupt vector jump table can be found in the following section.

4.1 Flash programming the interrupt vector jump table using BDM MultiLink

First the example project should be opened up in CodeWarrior. To do this, find the file "Hcs12dp256.mcp" in the ".\Example3"-subfolder and double-click it. This should launch CodeWarrior and open up the example project. Before the example application can be downloaded to your board, the source files need to be compiled into object files and then the object files need to be linked together to create the executable of your software application. Follow the instructions in sections 3.1 and 3.2 and apply them to the "Example 3"-project to flash program⁴ the interrupt vector jump table.

4.2 Step 1 – Building the executable

First the example project should be opened up in CodeWarrior. To do this, find the file "Hcs12dp256.mcp" in the ".\Example4"-subfolder and double-click it. This should launch CodeWarrior and open up the example project. Before the example application can be downloaded to your board, the source files need to be compiled into object files and then the object files need to be linked together to create the executable of your software application. In CodeWarrior, you do this using the "Make" command. You can find this in the "Project"-menu in CodeWarrior or you can click the button as illustration in Figure 11.

After this you should see that a file called "Hcs12dp256.abs" appeared in the "bin"-subdirectory. This file is the executable with we will download into the RAM in the following step.

⁴ A word of caution before you start the flash programming. The entire flash memory will be erased. This means that if you have D-BUG12 programmed in the flash memory, this will be erased. If it turns out that for some reason you're having problems with flash programming your HCS12DP256 and the flash was already erased, this means you won't be able to use your microcontroller until you solve these problems.

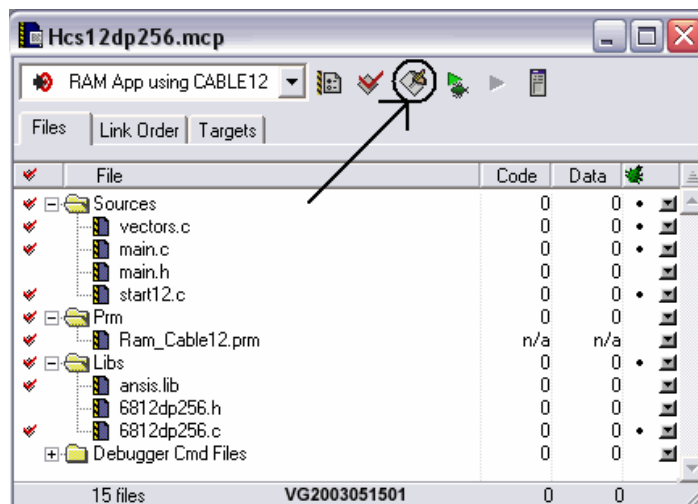


Figure 11 - Building the executable

4.3 Step 2 – Downloading using BDM MultiLink

The next step is to download your software application into the RAM of your HCS12DP256 evaluation board. The source level debugger that is part of the Metrowerks CodeWarrior environment includes functionality to do RAM downloading when using the P&E Micro BDM MultiLink interface. This has to be configured in the project settings and through the usage of several external files (debugger command files). The good thing for you is that I have already done this for you in the example project. To start with the actual downloading using CodeWarrior's debugger, all you have to do is click on the green play-button as illustrated in Figure 12.

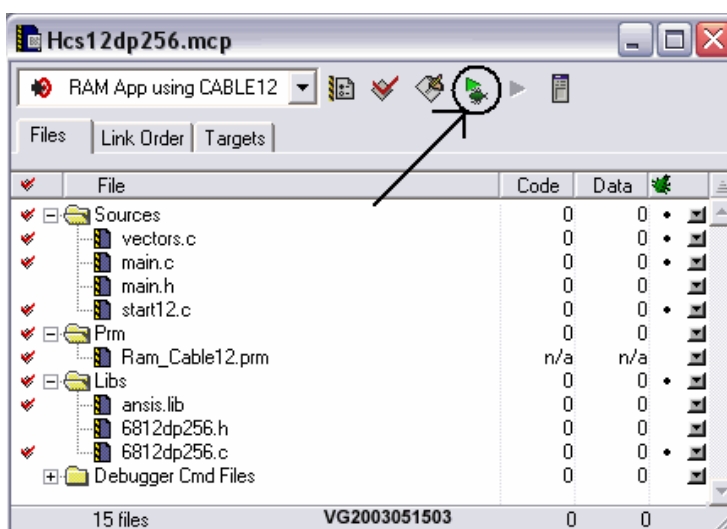


Figure 12 - Downloading your application to RAM

4.4 Step 3 – Running and debugging the example application

Once your application is downloaded to the internal RAM of your microcontroller using CodeWarrior's debugger, the actual running and debugging is exactly the same as described in the previous chapter. Refer to section 3.3 for more information.

4.5 A closer look

4.5.1 Project files

The project files for this example are similar to the ones described in section 3.4.1. There is only one difference, which can be found in the debugger command files. Since this configuration only downloads your software application to RAM, everything related to erasing, programming, and unsecuring the flash memory is stripped out. In the example the remaining debugger command files are already setup for you.

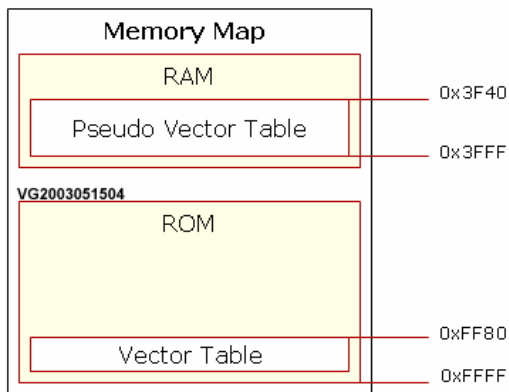
4.5.2 Memory configuration

How the memory on your evaluation board is configured has to be explained to CodeWarrior, otherwise it would create an invalid executable. This configuration is done through the “Ram_Cable12.prm” file for this example. I configured it for you as show in the table below. I would recommend that if you are new to software development on the HCS12DP256, that you keep these settings for now. You can of course modify these to your liking, as long as you don’t cross the physical boundaries of the RAM on the HCS12DP256.

Type	Start (hex)	End (hex)	Size (bytes)
RAM	0x1000	0x1EFF	3840
Stack	0x1F00	0x1FFF	256
ROM	0x2000	0x3F3F	8000
Vectors ⁵	0x3F40	0x3FFF	192

4.5.3 Vector tables

The figure on the right graphically illustrates how the interrupt vector tables are used in this configuration. Normally, only one interrupt vector table is present and it’s located at the bottom of the memory map, from address 0xFF80 to 0xFFFF. This is predetermined by the hardware of the HCS12DP256. If this wasn’t the case you could simply change some sort of setting on the microcontroller and move it to wherever we want, which is somewhere in RAM for this configuration. This is not possible and therefore a workaround has to be used in order to develop your software application from RAM without having to program the vector table into the flash memory every time.



The workaround I created involves the usage of a second vector table, which I refer to as the pseudo vector table. I located it at the end of the RAM. Let’s look at what happens when an interrupt occurs. The microcontroller will read the 2-byte address associated with the interrupt that occurred. For demonstration purposes, assume that the interrupt that occurred was an RTI interrupt. The vector for the RTI interrupt is located at address 0xFFFF0. The microcontroller would then read the 2-byte address of the interrupt service routine (ISR) function from 0xFFFF0 to 0xFFFF1 and would jump to this address automatically. In our configuration, we don’t know yet if we want to use the RTI interrupt and in case we chose to, it shouldn’t be necessary to re-flash the

⁵ Configured in “vectors.c”.

vector table just for this. Using the workaround, instead of jumping to the ISR directly, a jump is performed to a predetermined address in the pseudo vector table. At this location in the pseudo vector table, information is stored about where the actual ISR for the RTI is located, if used. Because the addresses in the pseudo vector table are predetermined, we only have to flash the actual vector table once into flash memory. Whenever we want to change the usage of an ISR we can do this in the pseudo vector table, which is part of the software application in RAM. Problem solved!

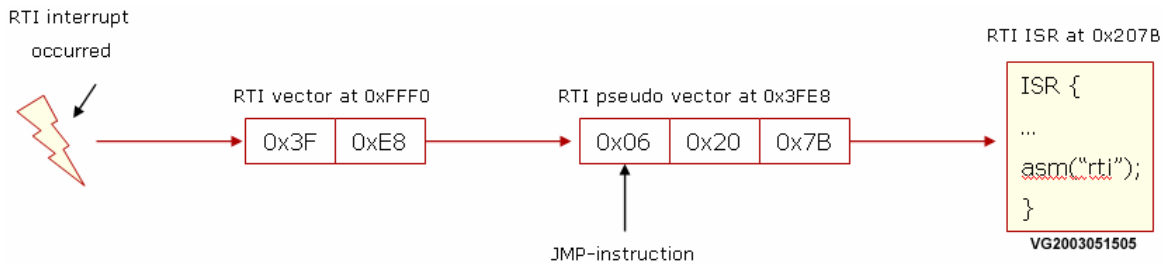


Figure 13 - Pseudo vector table functionality

Figure 13 graphically illustrated the functionality of the pseudo vector table as described for the RTI interrupt example. First the interrupt occurs, and the microcontroller will start executing code at the address that is stored at the RTI vector (0x3FE8). The code that the microcontroller finds there instructs it to jump to address 0x207B. This is the actual location of the ISR, which is of course ended with an “rti”-assembly instruction, indicating to the microcontroller that it is returning from an ISR and not a regular function. A little tricky, however, it is all preconfigured for you in the examples 3 and 4. When you need to use an ISR, all you do is add the function name of the ISR to the pseudo vector table in the “vectors.c”-file in Example 4.

4.5.4 Limitations

The only limitation is this configuration is that your software application has to be smaller than 12 kilobytes in total.

5 Conclusion

The information in this document provides a good overview of commonly used software development techniques for the Freescale HCS12 family of microcontrollers. I selected the Metrowerks CodeWarrior software development environment for usage in the examples because that way I could give you a more hands on explanation on how to get started. In addition to that it's a high quality development environment that's available for free (Special Edition).

Configuration	Cost	Ease of use	Memory restrictions	Speed
RAM using DBUG12	☺	☺	☺	☺
ROM using BDM MultiLink	☹	☺	☺	☺
RAM using BDM MultiLink	☹	☺	☺	☺

The comparison table summarizes the pros and cons of each configuration. The one that works best for you depends on your personal preference and balance regarding cost, ease of use, memory restrictions, and speed. Knowing that, you can use this table to pick the configuration that fits your needs. My personal preference is the last one ("RAM using BDM MultiLink"). If you have financial resources available to invest in the BDM interface and the 12 kilobytes of RAM are plenty for your software application, I can highly recommend this one to you.

6 Contact

You can visit our website at <http://www.feaser.com> for additional information and updates, or contact us directly:

Feaser LLC
3430 East Jefferson Av, #143
Detroit MI, 48207
United States
Ph. +1 (877) 293-1452
Fx. +1 (877) 852-0523
Em. info@feaser.com

References

G. Doughman

“Reference Guide For D-Bug12 Version 4.x.x”

April 5, 2002

Motorola Semiconductor

Motorola

“MC9S12DP256 – Advance Information”

December 1, 2000

MC9S12DP256/D

Motorola Semiconductor