

## Getting Started With S12X BDM

This document has a step-by-step procedure for installing, configuring and demonstrating Seehau HC12 and Nohau's Freescale S12X family support. It is intended to help you avoid problems in setup and introduce you to Seehau HC12 for the S12X.

If you do encounter problems, please call your Nohau support or send Email to [support@icetech.com](mailto:support@icetech.com). If you can include the information about which step first did not produce the expected results, a picture of the screen at that point, and the file: "c:\Nohau\SeehauHC12\Macro\startup.bas", if there is one, it will help us give you a quick, effective response.

For documentation of the Freescale MC9S12X... parts see the Freescale web site.

Together give the documentation for the MC9S12XDP512 and other members of the MC9S12X families.

We recommend reading the "S12XCPUV1 Reference Manual" and these sections of the MC9S12XDP512 Data Sheet:

Chapter 1	Device Overview (MC9S12XDP512V2)
Chapter 4	Port Integration Module (S12XDP512PIMV2)
Chapter 9	XGATE (S12XGATEV2)
Chapter 21	Interrupt (S12XINTV1)
Chapter 23	Memory Mapping Control (S12XMMCV2)

of the "MC9S12XDP512 Data Sheet" as an introduction to the S12X family.

We don't know anyone who has understood all 1500 pages of this documentation on the first reading. Expect to have to read some parts of those chapters several times to understand them.

It is helpful to use Adobe Acrobat Reader's search feature to look up bit names and register names that you don't understand.

Be warned that the S12X introduces two completely new memory address mappings, the global addresses, and the XGate addresses. They do provide some real advantages, but I haven't seen a simple way to explain them. The "Data Sheet" has diagrams and examples of them that are worth studying carefully.

Our attempt to explain the different address mappings is at the end of this document in the section titled "Understanding the S12X Family".

## Requirements

BDM emulators assume that there is a “target” system with a Freescale / Motorola standard 6 pin BDM connector on it. (The Freescale / Motorola “evaluation boards” and most other commercial boards have this connector.)

A PC running Windows 98, 2000, ME or XP.

A display of 1024 by 768 pixels is recommended so that you can view the important windows of both the S12X and the XGATE co-processor at the same time. An even larger display is very helpful when debugging the interaction of S12X and XGate programs.

## Installing Seehau HC12

Installing the software before the hardware is required for USB connections and causes no problems with other connections.

As installation requires installing Windows device drivers, it should be done from an “administrator” account if you are using Windows 2000, ME or XP.

Install Seehau HC12 from a CD or from the Nohau web site, [www.icetech.com](http://www.icetech.com). The installation follows normal Windows conventions. To keep things simple at first, install with all the default settings. Later you can un-install Seehau HC12 and re-install with any custom settings you desire.

At the end of installation, you will be asked if you want to view Read\_Me.txt and Launch Seehau HC12 Configuration. Check both checkboxes and click on “Finish”.

After reading “Read\_Me” you will get the Seehau configuration screen.

## Configuring Seehau HC12

Go through these configuration steps in order.

Select the Connection type by clicking on the appropriate picture. Click on “Next”.

If you have an “EMUL-PC/EPC” select which parallel port you will be using. In any case, click on “Next”.

Select the processor. Select “BDM-S12X-Family” for all MC9S12X... parts. (Seehau HC12 will read the PARTID register from the target S12X part to determine which part of the S12X family is connected.) Click on “Next”

The trace support will default to “On chip trace feature only”. If you are using a BDM pod with the on chip trace support (EMUL12-PC/BDM-S12X-TRC) the on chip trace will be decoded, but if you are using the BDM pod without trace support (EMUL12-PC/BDM-S12X), the trace window can be opened but no trace information will be displayed.

## Getting Started With S12X BDM

You should now have a window that looks something like this:

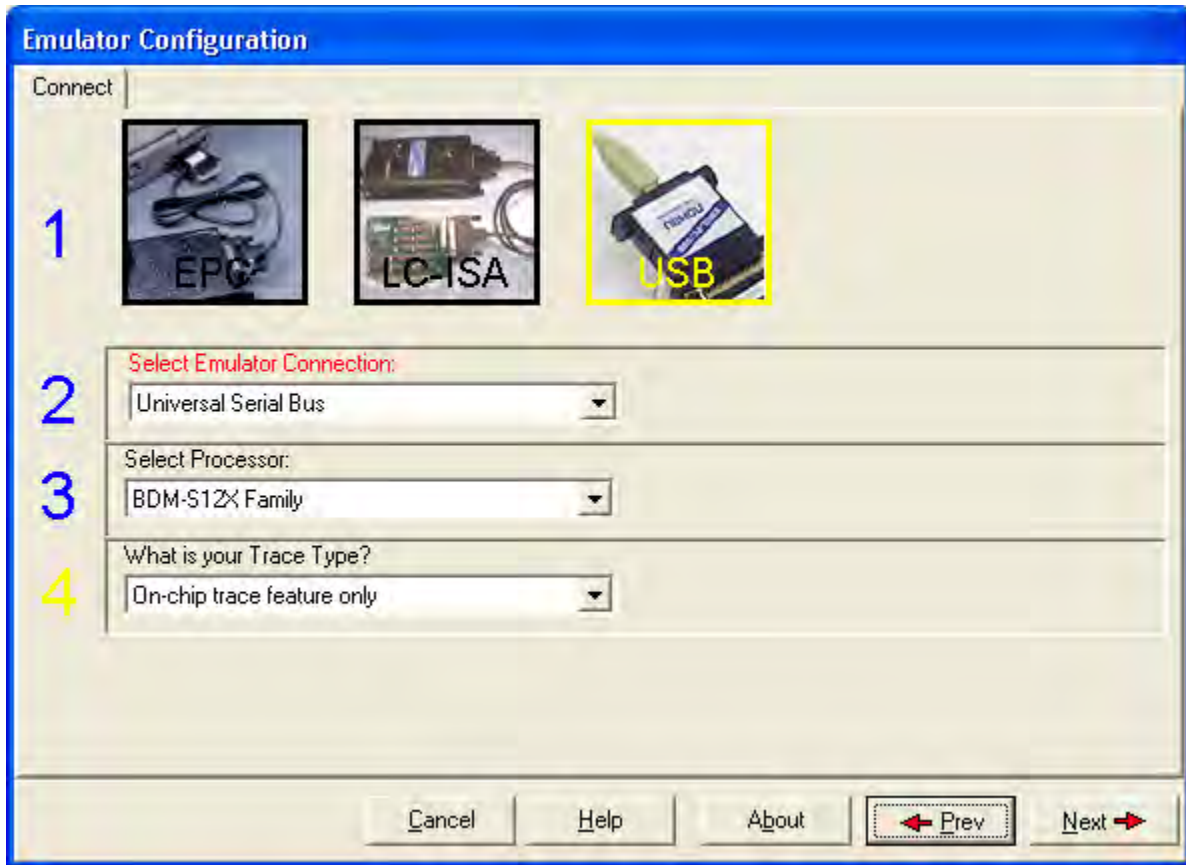


Figure 3 – First Configuration Window

Click on "Next".

The "Emulator Configuration" window will appear. (All of these settings except the processor selection and the Clock frequency may be easily changed later.) Enter the frequency of the crystal or external oscillator connected to the S12X EXTAL pin. In the "Clock(MHz)" field in the middle of the window.

## Getting Started With S12X BDM

You should now have a window that looks like this, but with the Clock(MHz) field set to your target's external clock speed:

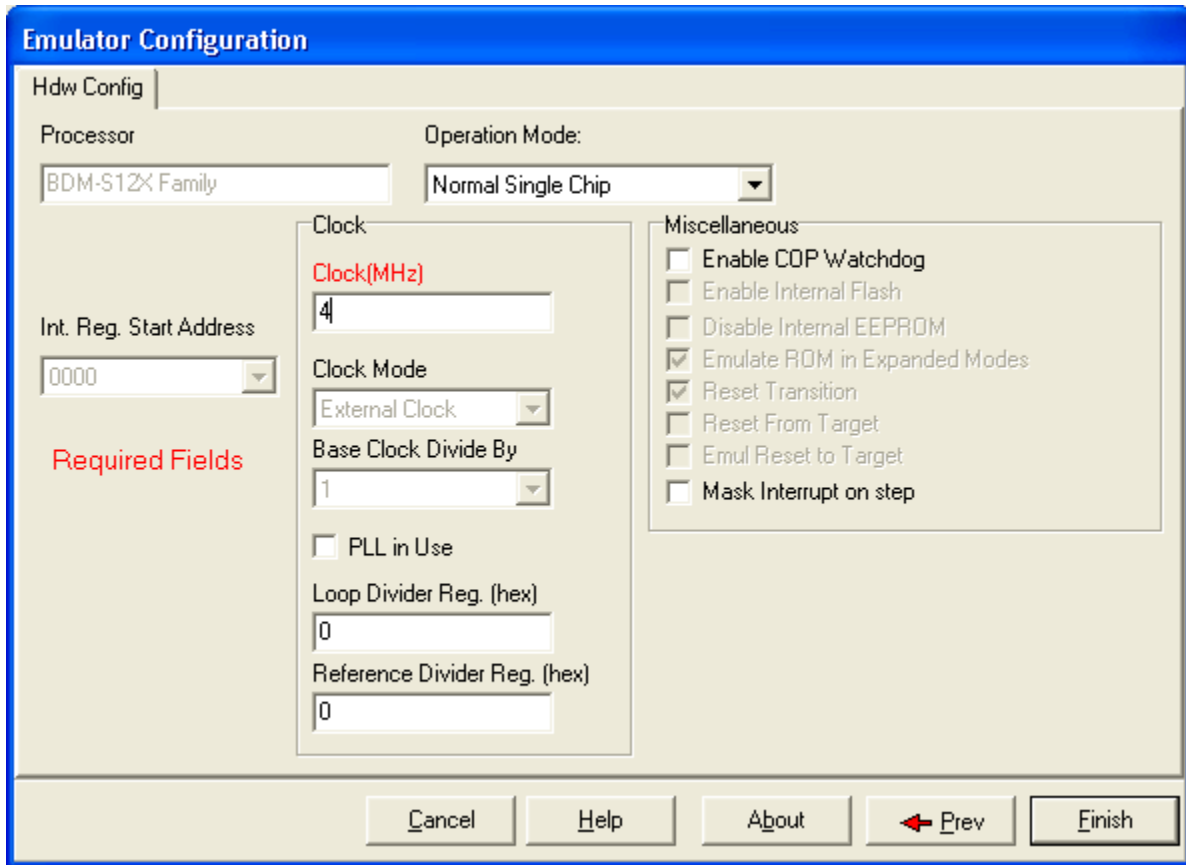


Figure 4 – Second Configuration window

Click on “Finish”.

Your Seehau HC12 configuration is now written to the Seehau “Macro” directory as “Startup.bas”. This file is written in Seehau’s powerful “Macro” language, which is very similar to Microsoft’s “Visual Basic for Applications” or VBA.

You will be asked if you want to “Start Emulator?”. Click on “No”, because the hardware has not yet been connected.

(After the hardware has been installed and tested, more configuration will be needed for the XGate co-processor. This is explained in detail later on in this document.)

## **Assembling The Nohau Hardware**

### ***The parts***

There are 4 main pieces in the Nohau S12X BDM emulators. Each one has Nohau part identification and a serial number on a label:

A “BDM pod” which has a cable with the Freescale / Motorola standard 6 position BDM connector socket on the end, and 2 single wires ending for forcing the MODA and MODB. This is labeled either “EMUL12-PC/BDM-S12X” or “EMUL12-PC/BDM-S12X\_TRC” if it includes trace support.

A power adapter that supplies power to the BDM, labeled “EMUL/EPC-ADP”.

A device for connecting to a PC, labeled “EMUL-PC/USB, “EMUL-PC/EPC” or “EMUL-PC/LC-ISA”.

A desktop power supply.

### ***Connecting to the Target***

Make sure power is off on the target and the Nohau power supply is unplugged.

Plug the 6-position socket at the end of the cable on the BDM pod onto the 6 pin BDM connector on the target. The red stripe on the cable should be on the same end as “pin 1” on the “BDM” header on the target.

Plug the power adapter into the 25 pin connector on the BDM pod.

Plug the coaxial connector from the Nohau power supply into the side of the power adapter.

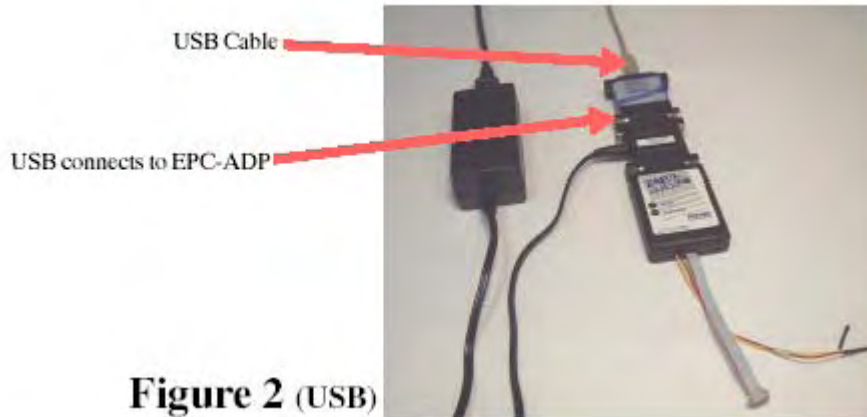
Do not turn on the power to the target or the Nohau power supply yet.

### ***Connecting to the PC***

The connection procedure depends on your connecting device.

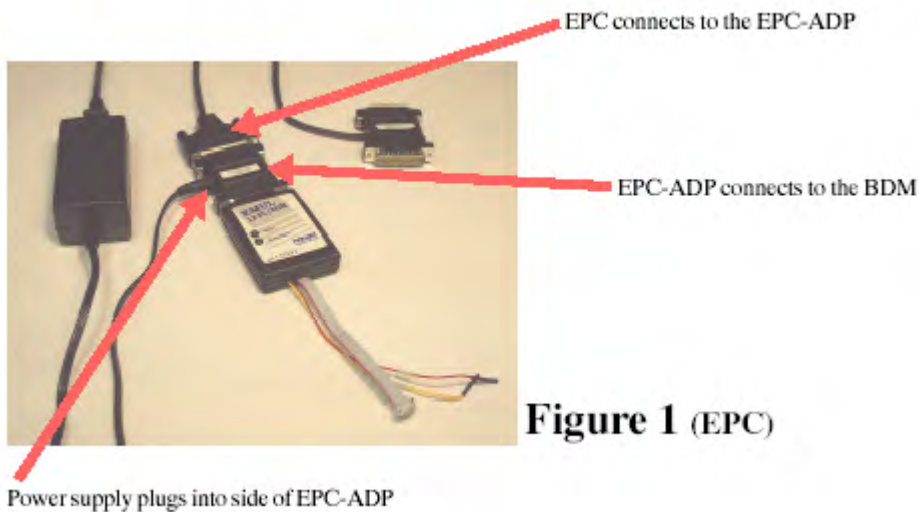
### For EMUL-PC/USB connection to the PC

Plug the connecting device into the remaining 25-pin connector on the power adapter. Plug one end of the USB cable into the EMUL-PC/USB and the other end into your PC or a USB hub connected to your PC. See figure 2 below:



### For EMUL-PC/EPC connection to the PC

Plug the single connector on the end of the cable into the power adapter. Plug the end of the cable with two connectors into the parallel or “printer” port on your PC. See figure 1 below:



### For EMUL-PC/LC-ISA connection to the PC

Shut down your computer, and install the EMUL-PC/LC-ISA card in a ISA (“Industry Standard Architecture”, now obsolescent) slot, and start up your PC. Next use the supplied cable to connect to the 25 pin connectors on the EMUL-PC/LC-ISA and the power adapter.

## ***Powering Up***

Turn on power to the target.

Plug in the Nohau power supply.

If you are using the USB connection, Windows may say it has found new hardware. Take the defaults as Windows finds the Nohau USB support.

## **Running the Example Program**

Click on the Windows “Start” button, and click on “Programs”, then on ”Seehau HC12” and then on “”Seehau HC12”.

(This is the normal way of starting Seehau. It will read your configuration file which has the settings that were last saved.)

You should see a window something like this:

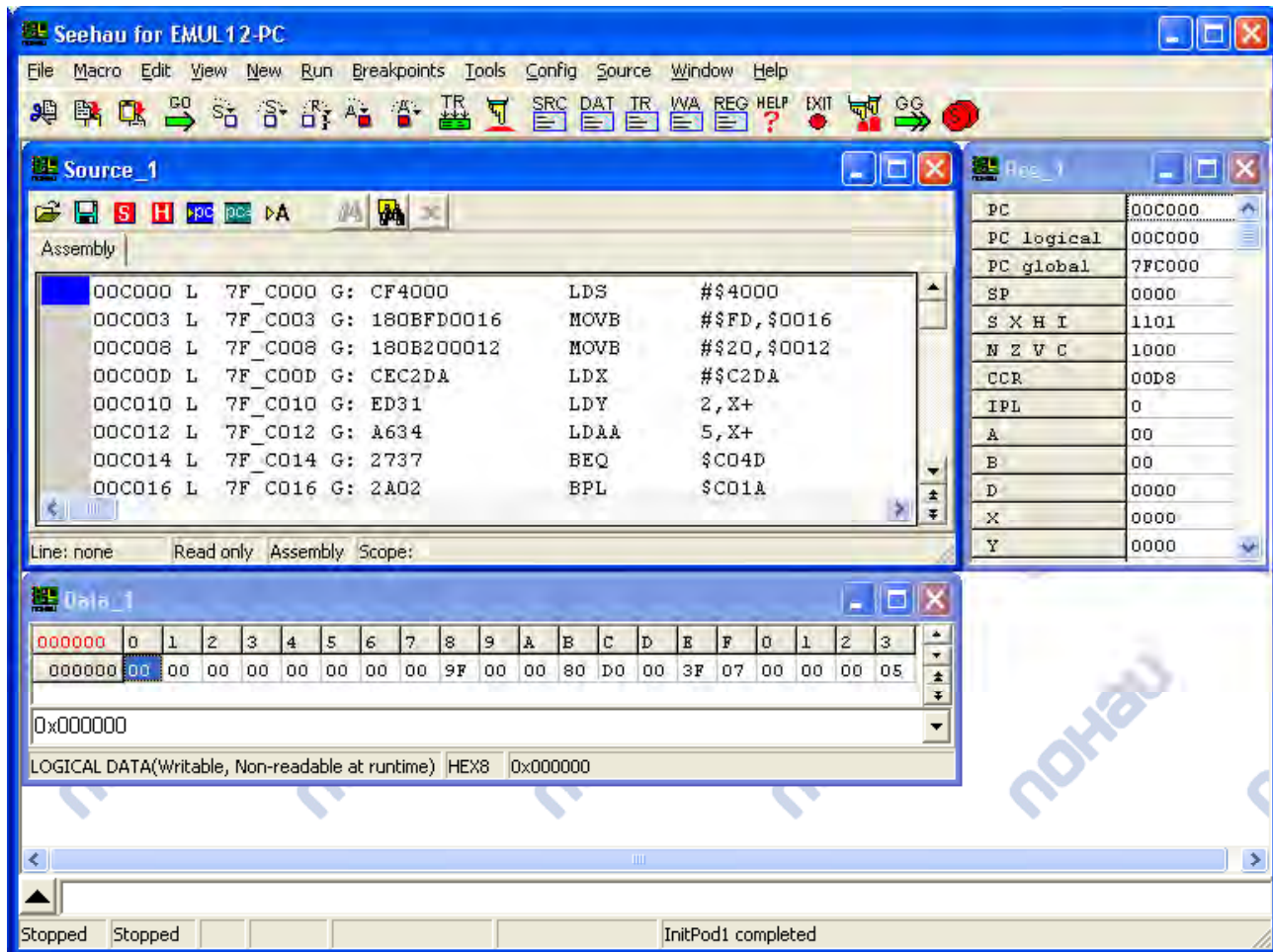


Figure 5 – Just Started SeeHau HC12

The details of the contents of the windows will vary, and some error indications may be present if your target has erased flash, which will give an invalid reset vector.

## Getting a Program into Flash Memory

The next step of this "Getting Started" guide will erase the on-chip flash memory and program a demonstration program into it. Please make sure you can re-create the contents of this memory if they are important before proceeding.

This program does not change any MCU pins from their reset settings, so it should leave the target devices external to the MCU in their power on reset state. It does assume that the target will not produce any external interrupts or resets. The Freescale / Motorola evaluation boards meet these requirements.



## Getting Started With S12X BDM

On the main menu "Tools" item, select "Prom Programmer ..." to program on-chip flash and EEPROM.

Check the "Program Flash" box and click on the "Browse" button. The file browser window should open in the "Examples" directory. Double click on " MC9S12XDP512", then "Timer - Cosmic" and finally on "Nohau Time.elf". The Prom Programmer window should now look like this:

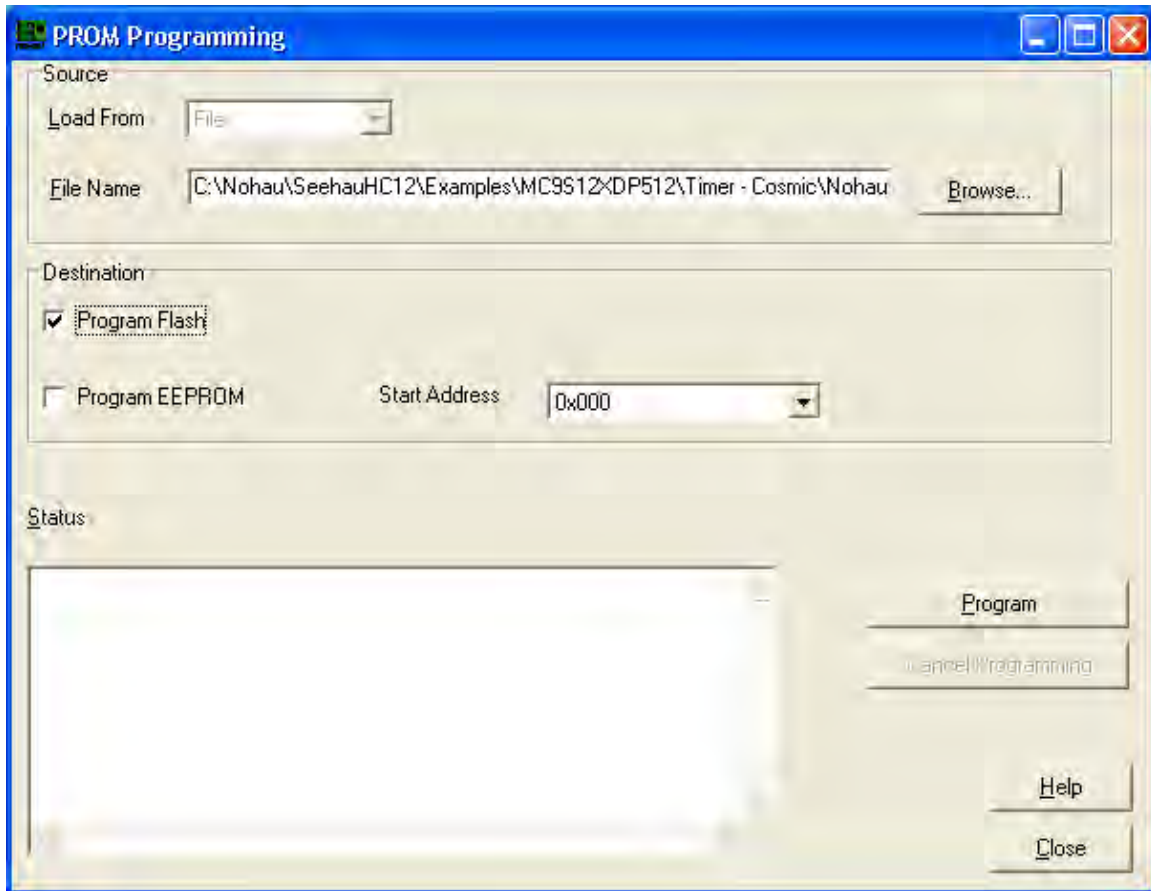


Figure 6 – PROM Programming Window

## Getting Started With S12X BDM

Click on the "Program" button. You should see various progress reports in the "Status" window and a final summary that looks like this:

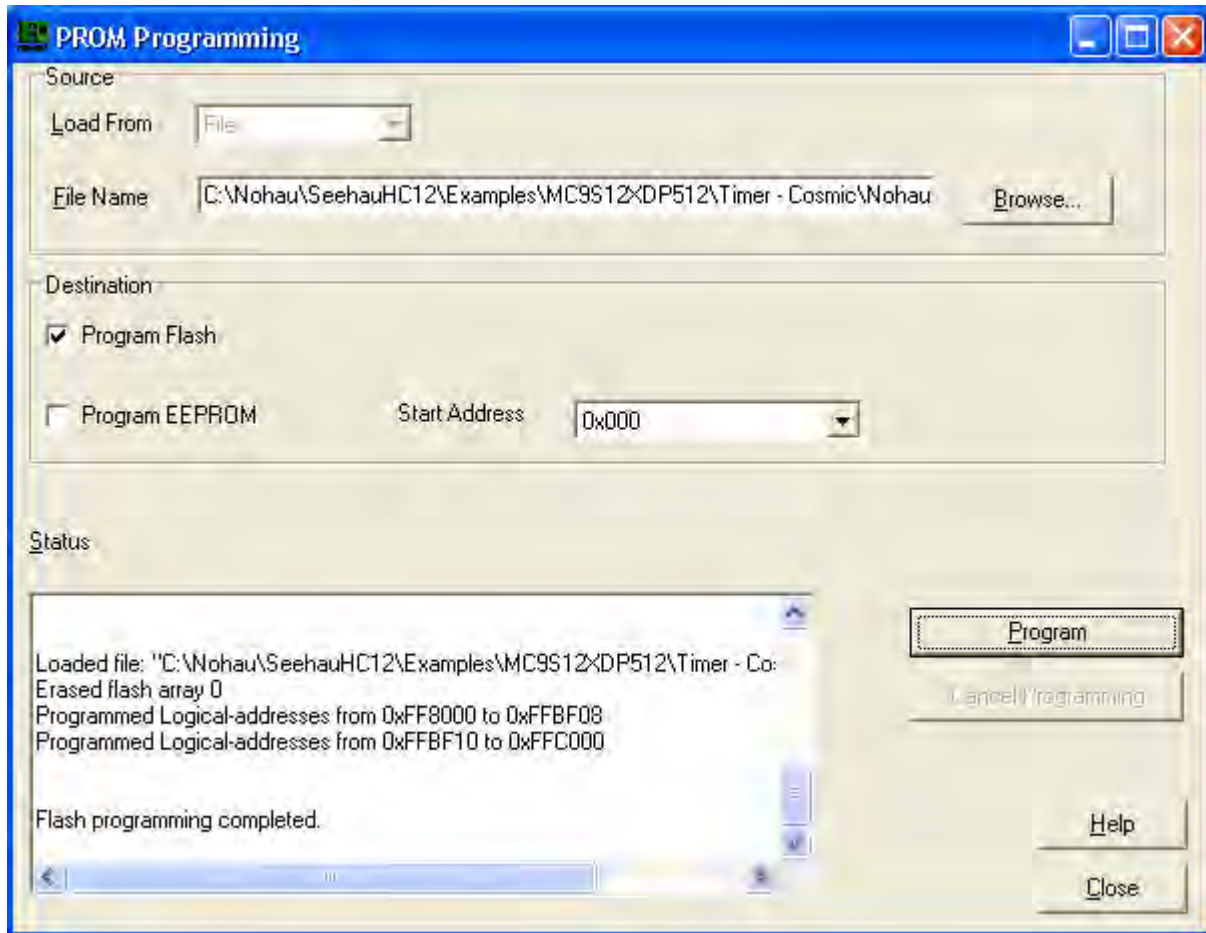


Figure 7- Flash Programming Completed

Click on the "Close" button.

The example program is now programmed in flash memory. The flash programming is verified automatically as it is written, so it is very unlikely that there are any undetected errors.

### ***Debugging a Program In Flash Memory***

The next step will verify that program was programmed correctly in flash, which demonstrates the Seehau load verify feature and demonstrates the flash programming was done correctly. Because flash programming is verified automatically during programming, it is unlikely that there are any programming errors. The verification step is very helpful if you are not certain that the load file matches the contents of the on-chip flash memory.

This step also gives Seehau the debugging information that the linker has put in the load file. This information enables symbolic and source-level debugging.

## Getting Started With S12X BDM

From the main menu, select "File - Load Code ...". Check "Verify" and "Load Symbols" and un-check "Load Code". The window should look like this:

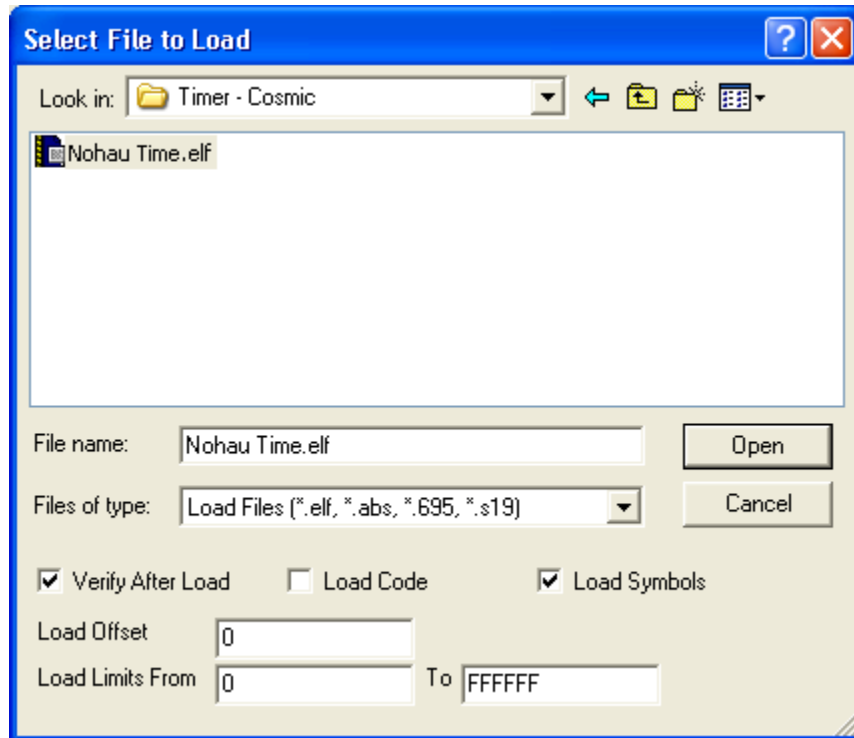


Figure 8 – File Load Window

Then select the "Nohau Time.elf" file as you did for flash programming, and click on "Open" if necessary.

## Getting Started With S12X BDM

You should get a short series of progress reports and no error messages. With the reset vector now programmed to point to the startup code and the symbols loaded the Source window should now look like this:

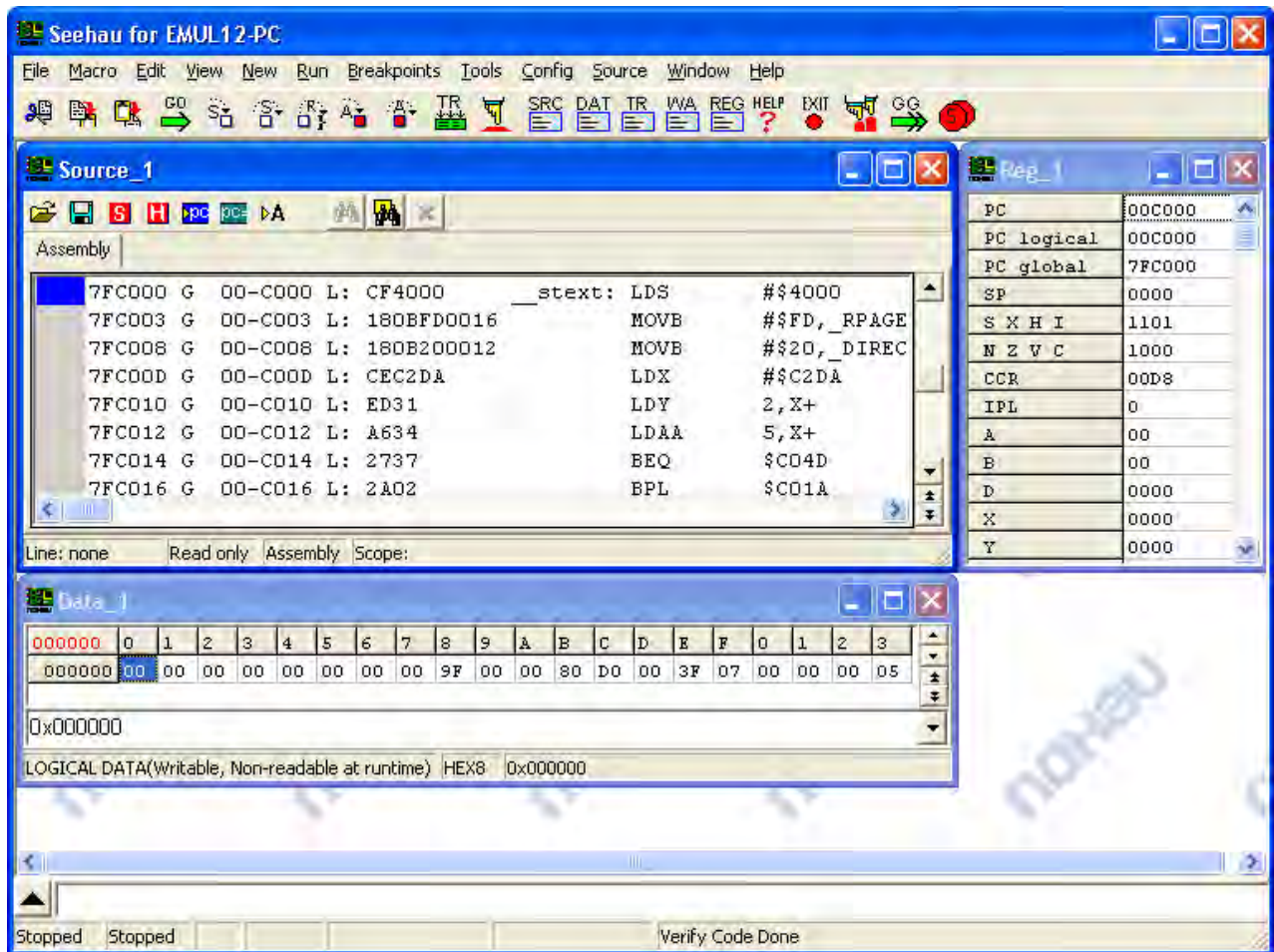


Figure 9 – After Verifying Flash and Loading Debug Information.

The assembler symbol "\_\_stext" has been identified as global address 7F\_C000 or logical address 00-C000 and is disassembled as "LDS # \$4000" from the hex values of CF4000 in C000 through C002.

(If the terms "global address" or "logical address" are new to you, you should learn about them. For more information consult section at the end of this document titled "Memory Addressing".)

## Using On Chip Hardware Breakpoints to debug code in FLASH

In order to debug the code while executing out of the internal FLASH or EEPROM of the target S12X, it is necessary to use the S12X internal hardware breakpoints. This option is selected by default when creating a fresh SeeHau configuration for the S12X BDM.

## **Getting Started With S12X BDM**

On "Run" menu, verify that a check mark is shown next to the "Force Hardware Step" item (about two thirds of the way down the menu). If for some reason "Force Hardware Step" is not checked, click on it now in order to check it.

The "Force Hardware Step" option, configures Seehau to use by default the four S12X internal hardware-breakpoints, when breakpoints are specified, or needed.

If the "Force Hardware Step" option is un-checked, Seehau will attempt setting breakpoints by writing a BGND instruction over an instruction where a breakpoint is needed. This is useful when debugging code that executes in RAM, but will fail in the usual BDM case when debugging code that executes in the internal FLASH or EEPROM.

### ***Executing S12X code***

Now click on the "Run - Step Into" menu item.

## Getting Started With S12X BDM

You should now have the source window positioned to the first source line of the "main" c function and have a display that looks like this:

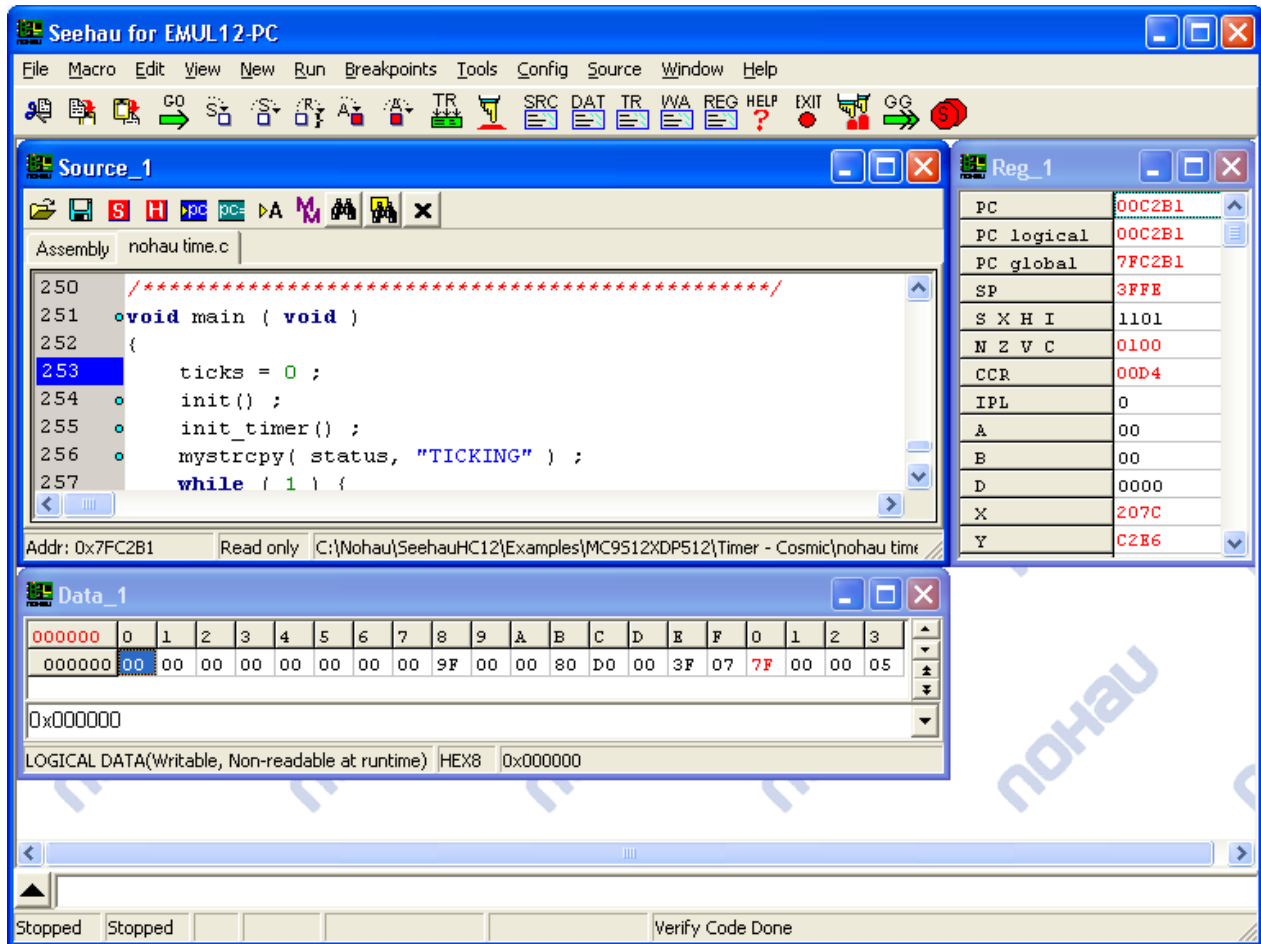


Figure 10 – After First Source Step Into

Just under the main menu is a "toolbar" with icons for various commonly used SeeHau functions. If you hold the cursor over one of these icons for a second, you will get "flyby help" giving the name of the function, and the shortcut key if any. About 5 icons from the left, just after the "GO" icon is an icon with an arrow. Put the cursor over it and see that it is "Source step into (F7)" indicating that the F7 key is the shortcut for this function.

Click on the "Source step into" icon on the toolbar.

The blue bar in the left margin of the source window should move down one statement.

## Examining variables and their content

Select the source window, point the cursor over the "ticks" symbol in the source line "ticks = 0;". After a second you should information pops-up like this

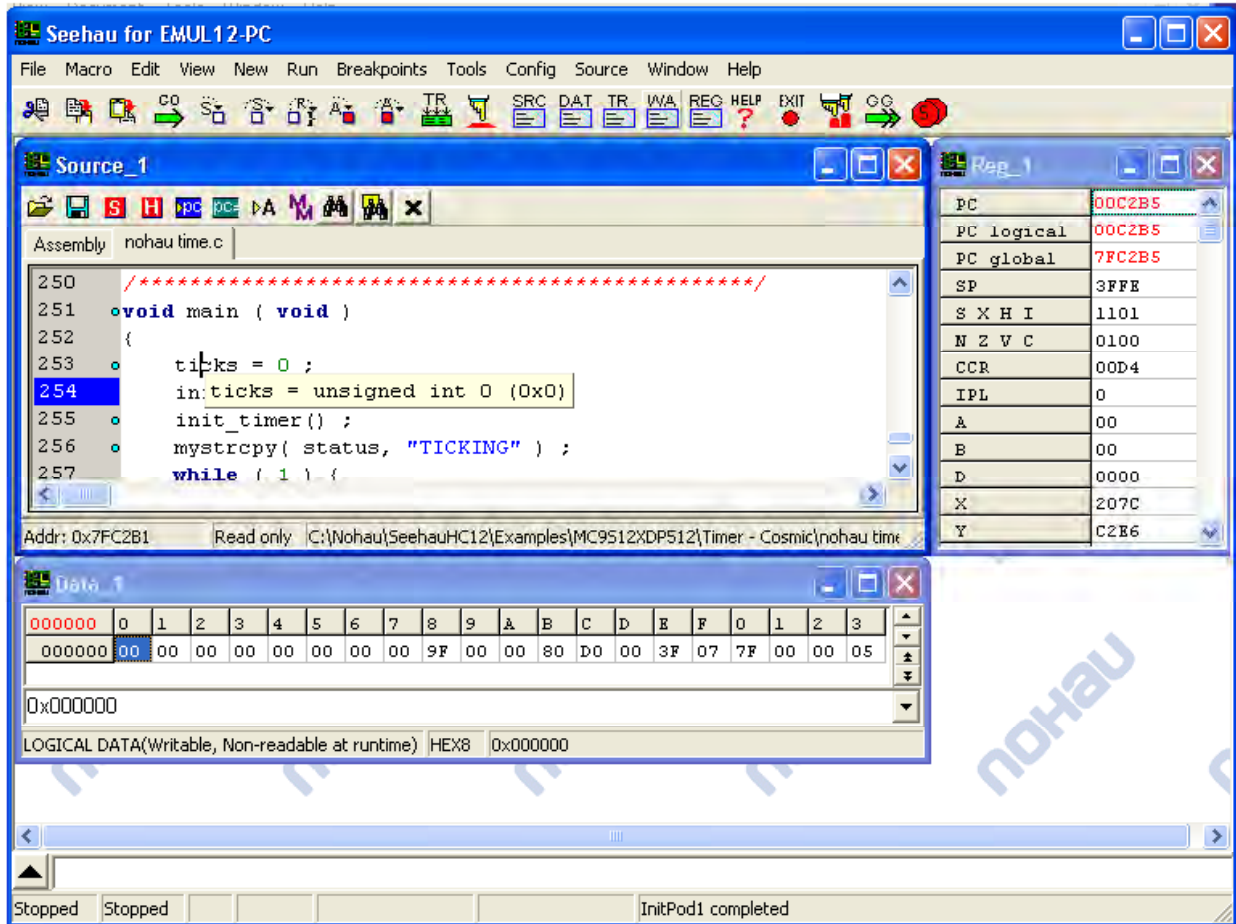


Figure 11 – “Flyby” Inspect of C Variable.

The pop-up information "ticks = unsigned int 0 (0x0)", gives you the type information about the variable "ticks" and its current value in decimal and hex.

With the cursor still over "ticks" in the source window, right click. In the right click menu, click on "Debug Windows ..." and select "Add to inspect (Ctl-I)".

## Getting Started With S12X BDM

The “Inspect/Watch” window will open, and you should see:

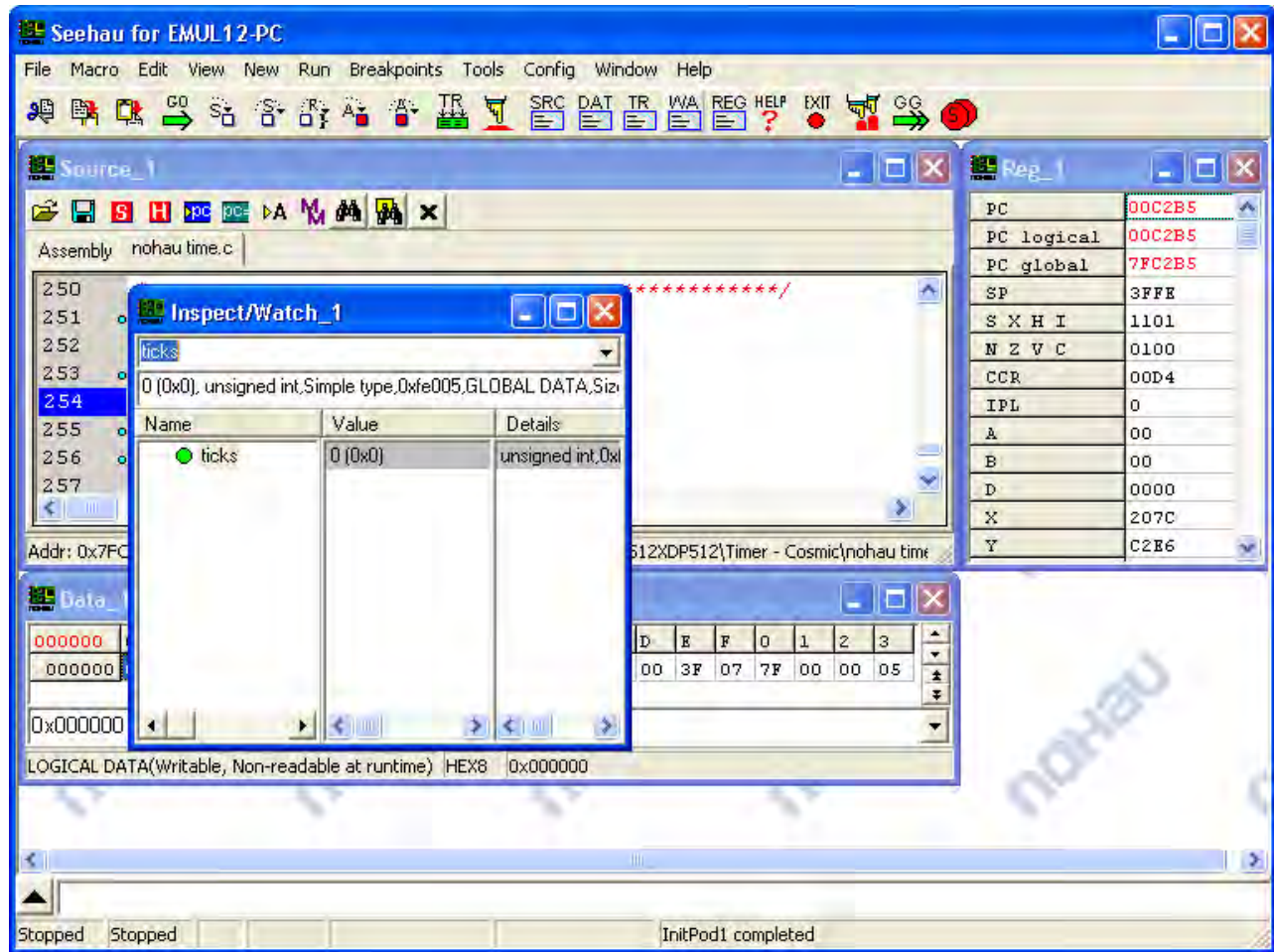


Figure 12 – Ticks Added To Watch Window

An "Inspect/Watch" window has been opened and the variable "ticks" has been added to it.

You can move the window around and re-size it in the standard windows ways.

To adjust a division between columns put the cursor over the division in the legend bar at the top of the column. When the cursor changes to a vertical bar with left and right arrows, hold down the left mouse button and drag the dividing line.



## Getting Started With S12X BDM

I re-sized my windows to look like this:

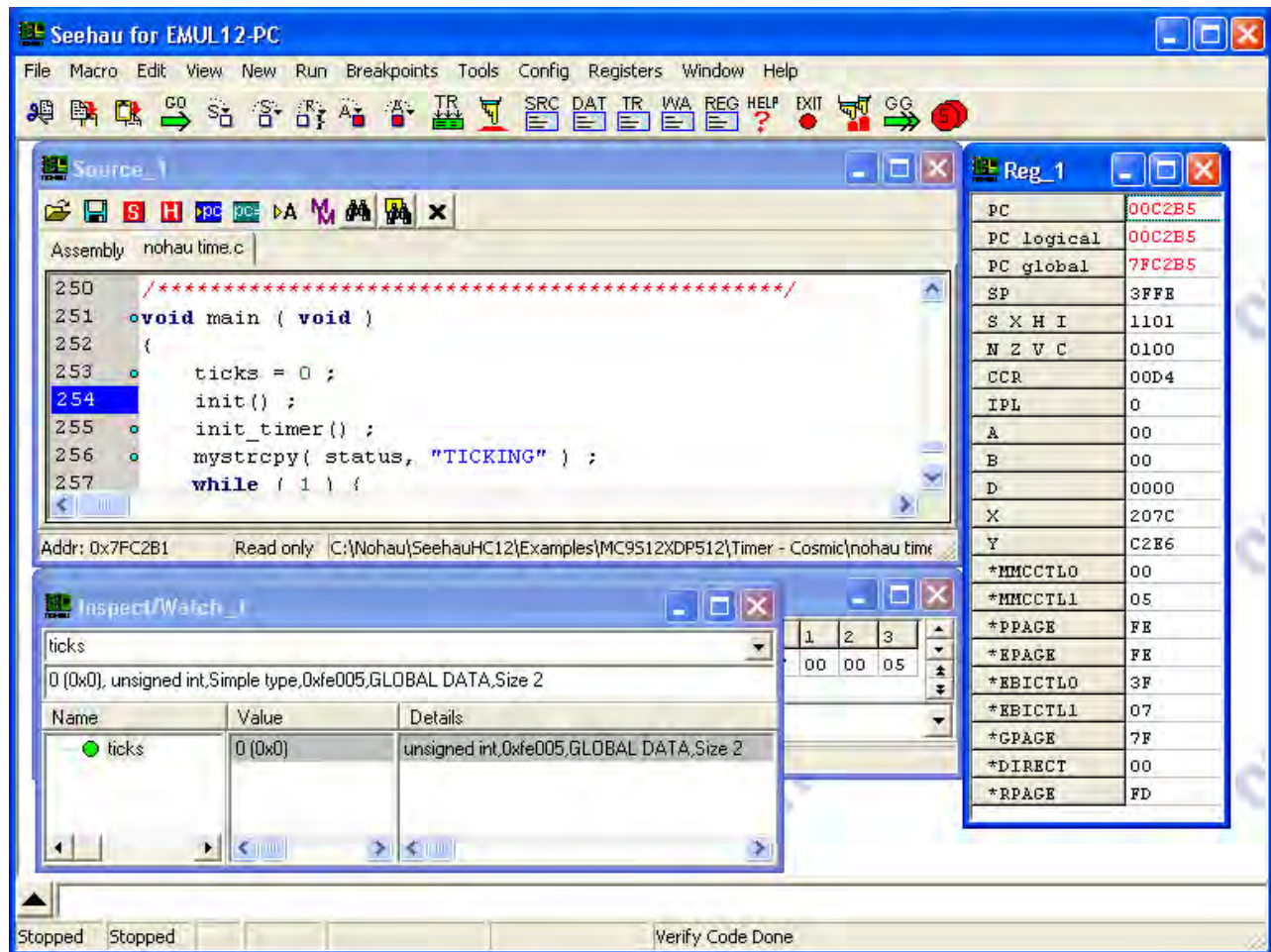


Figure 13 – After Resizing and Source Step

Now in the "Source\_1" window double click on the "status" variable in the line:

```
mystrcpy( status, "TICKING" );
```

Note the contents of the status array show on the screen.

Click on "status" variable in this line, and type Control I. This will add the "status" array to your Watch window.

## Placing Breakpoints and Running

Now go down in the "Source\_1" window to line number 256:

```
if ( ( TFLG1 & 0x80 ) == 0x80 ) {
```

## Getting Started With S12X BDM

Notice the little colored dot to the right of the line number in the left margin. This indicates that SeeHau has sufficient information to place a breakpoint on this source line.

Click anywhere in the left margin of line 256.

A red bar with cross hatching appears to indicate that a breakpoint has been placed on this line.

Now click on the "GO" icon on the toolbar.

The emulator will start the target, which will execute the program at full MCU speed. At the breakpoint you just set it will stop.

You should now have a display that looks like this:

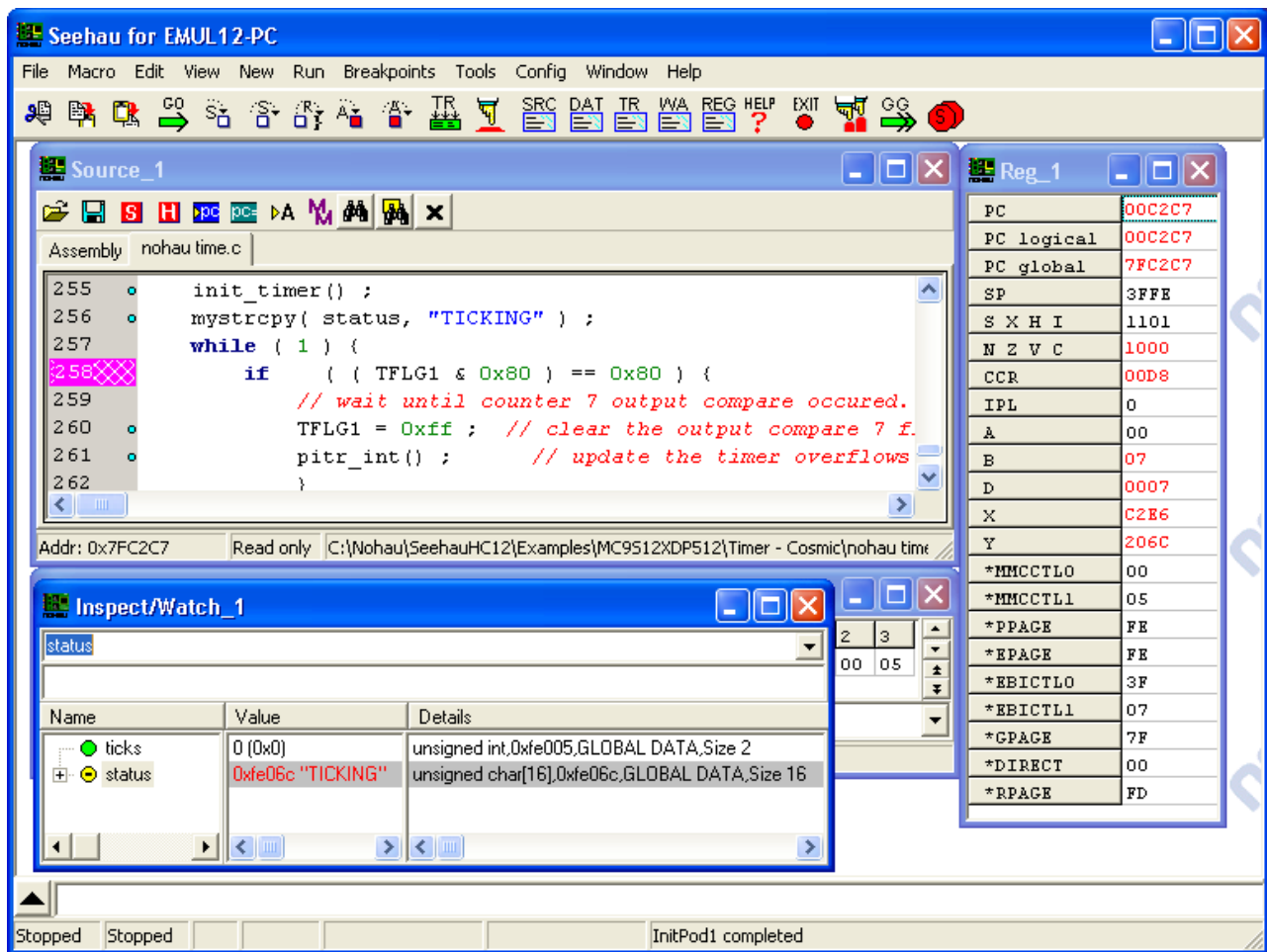


Figure 14 – After Adding Status to Watch Window

There are several changes highlighted to help you see what has changed.

The breakpoint indication in the left margin of line 258 has changed color to indicate that it is set and that the program is currently stopped just before executing the statement on line 258.

## Getting Started With S12X BDM

The "status" display in the "Inspect/Watch-1" window now displays the new value of "TICKING" in red to indicate that it has changed.

Many of the CPU registers in the "Reg\_1" window are also displayed in red because they have changed.

Now click on the source step into icon on the toolbar.

You may have to scroll around in the "Source\_1" window a little to see what is going on. The if statement the program was stopped on was executed, and the condition was false, so the statement after the if statement is now the next statement.

Click on the source step into icon again.

Now the source window displays the first line of the function "timer\_func".

Double click on the structure name "timer" on line 185. Type control I to place it in the watch window.

In the left panel of the watch window, click on the "+" sign in the little box to the left of "timer". This expands the structure and shows the current values of the elements of the structure.

## Getting Started With S12X BDM

Resize the window to get a display like this:

The screenshot displays a debugger interface with three main components:

- Code Window:** Shows C code with line numbers 181-188. Line 184 is highlighted. The code is:

```
181 {
182     // By checking change_min,
183     // check if the seconds have turned from 59 to 0
184     if ( change_min == 1 ) {
185         timer.min ++ ;
186         change_min = 0 ;
187     }
188     else {
```
- Register Window:** Located on the right, it lists various registers and their values:

PC global	7FC1B4
SP	3FFC
S X H I	1101
N Z V C	1000
CCR	00D8
IPL	0
A	00
B	07
D	0007
X	C2E6
Y	206C
*MMCTLO	00
*MMCTL1	05
*PPAGE	FE
*EPAGE	FE
*EBICTLO	3F
*EBICTL1	07
*GPAGE	7F
*DIRECT	00
*RPAGE	FD
- Inspect/Watch\_1 Window:** A window titled "Inspect/Watch\_1" with a search field containing "timer". It displays a tree view of the timer structure:

Name	Value	Details
ticks	0 (0x0)	unsigned int,0xfe005,GLOBAL DATA,Size 2
status	0xfe06c "TICKING"	unsigned char[16],0xfe06c,GLOBAL DATA,Size 16
timer	{...}	UnnamedStruct0,0xfe035,GLOBAL DATA,Size 23
hour	"\n" 10 (0xa)	unsigned char,0xfe035,GLOBAL DATA,Size 1
min	"\n" 17 (0x11)	unsigned char,0xfe036,GLOBAL DATA,Size 1
sec	"\n" 22 (0x16)	unsigned char,0xfe037,GLOBAL DATA,Size 1
am_pm	0xfe038 "PM"	unsigned char[10],0xfe038,GLOBAL DATA,Size 10
status	0xfe042 ""	unsigned char[10],0xfe042,GLOBAL DATA,Size 10

The status bar at the bottom shows "Stopped" and "InitPod1 completed".

Figure 15 – Timer Structure Expanded in Watch Window

Double click on "timer" in the top field of the "Inspect/Watch\_1" window, and type "show" to add the show array to the watch window.

## Getting Started With S12X BDM

Resize the window to get something that looks like this:

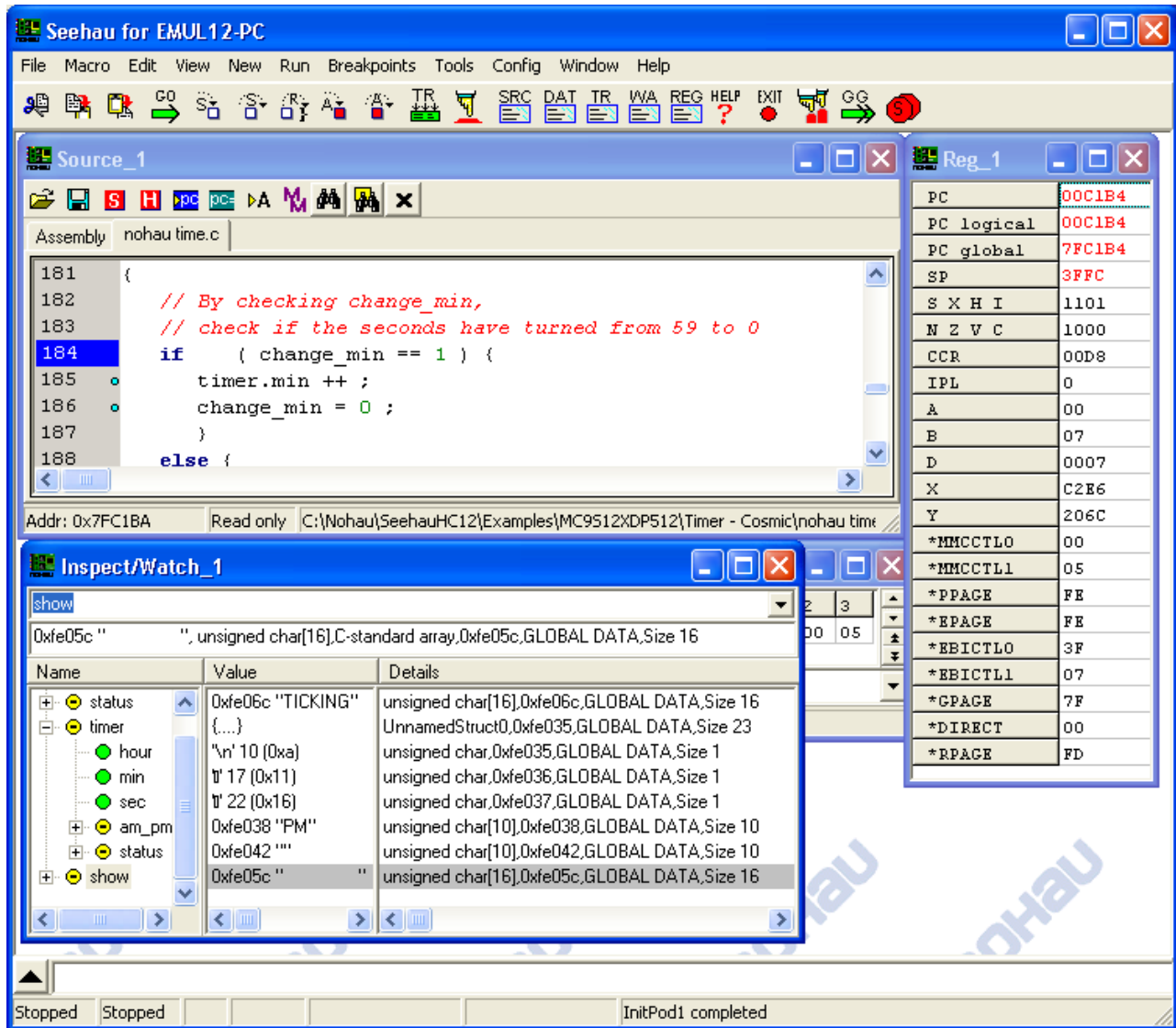


Figure 16 – Show Added to Watch

In the "Inspect/Watch\_1" window right click and check the "Update During Runtime" item.

Now click on the "Go" icon.

## Getting Started With S12X BDM

You should get something like this after re-sizing the watch window:

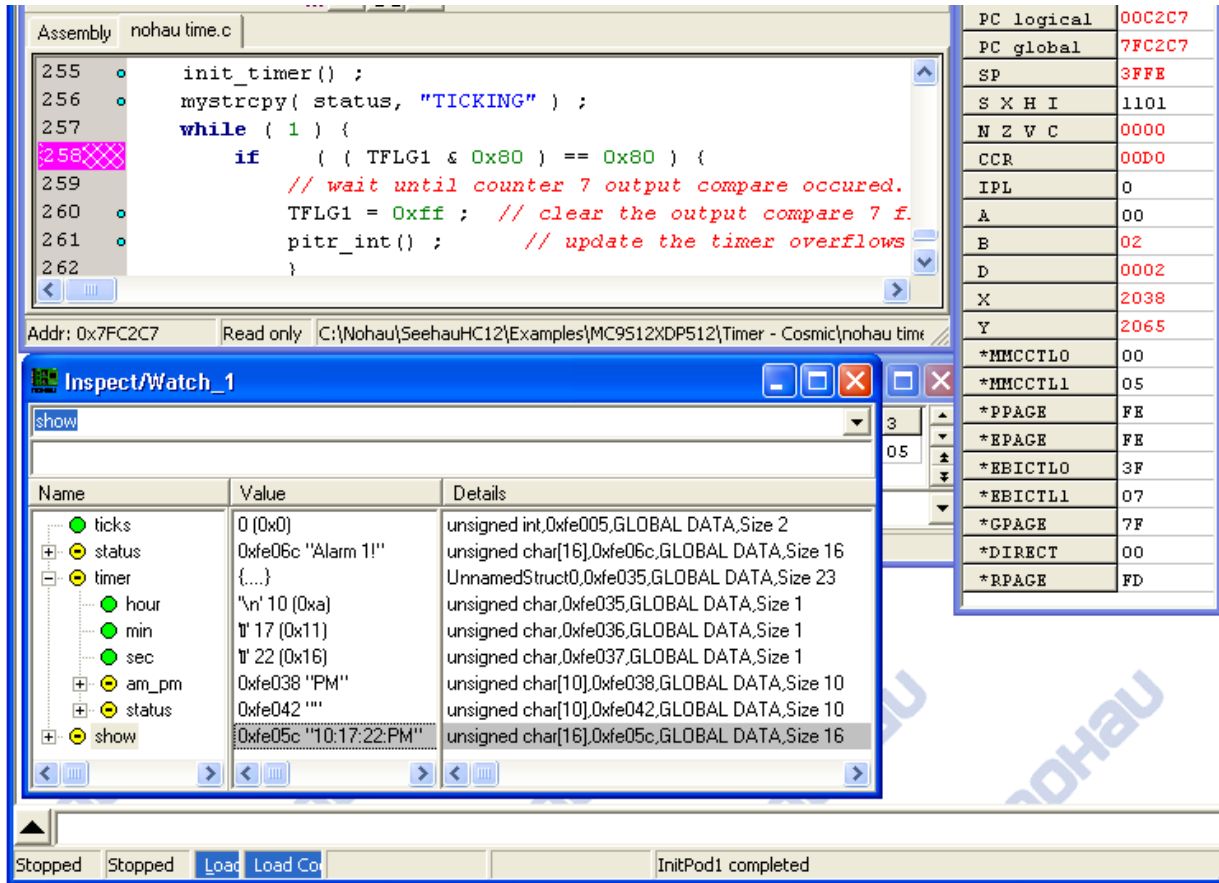


Figure 17 – After Go

Click on the breakpoint indication in the source window to remove the breakpoint on line 258. Click on "Go"

Watch the display for a minute or two to see the updating in of the inspect window while the target runs.

Note that the green "Go" icon has changed to a red "Stop" icon, indicating that the target is running.

Click on the red "Stop" icon.

### **Using the S12X internal PLL to ramp-up the bus speed**

The S12X internal PLL is used to change the bus speed. The default S12X bus-speed after Reset is typically fairly low, and is half the Crystal frequency (half the frequency supplied externally to the S12X - EXTAL pin).

## Getting Started With S12X BDM

In this example, we will setup the internal PLL as appropriate for the default settings of the Freescale / Motorola evaluation board. If you are using a different S12X target board, the PLL settings may be different.

The S12X internal PLL requires a network of two capacitors and one resistor to be connected to the external XFC pin. With no such network being connected to the XFC pin it is impossible to engage the PLL and lock it. The Freescale / Motorola evaluation board for example, comes with such network already installed: A 470pF capacitor between XFC and VDDPLL, in parallel with a 5.6Kohm resistor and a 4.7nF capacitor connected in series.

*Note: If your target board does not include a network of 2 capacitors and one resistor on the XFC pin, and you are unable to install them, please go now to the next chapter, and skip engaging the S12X internal PLL.*

*Note: Suggested values for the XFC components and suitable settings can most easily be gotten from the program "S12X\_PLL\_Filter\_Calculator.exe" available on the Freescale web site.*

*Note: The reference divisor and the loop divisor have a number of confusing traditions around them as a result of historical evolution.*

*The assembler symbol for the reference divisor is called "REFDV", and the assembler symbol for the loop divisor is called "SYNR".*

*The actual divisors are the number in the register plus 1.*

*The Freescale documentation lists the loop divider register first, because it has a lower memory address.*

*The Freescale S12X PLL Filter Calculator lists the reference divider register first, and produces the values for the registers in decimal.*

*The Seehau HC12 Emulator Configuration Window lists the loop divider register first, and expects hex values as would be used when debugging assembler code for MCU startup after a reset.*

The S12X bus speed out of Reset in the case of the Freescale / Motorola evaluation board is 2 MHz (half the EXTAL frequency). In this example we will engage the PLL to make the bus speed 40 MHz – the maximum specified frequency for the MC9S12XDP512 device.

In order for the BDM to operate with an engaged S12X PLL, Seehau must know the S12X PLL settings. This is needed since the S12X PLL affects the BDM communication frequency, so Seehau needs to know the PLL settings in order to implement them, and select the PLL and switch the BDM communication frequency in a controlled manner. (this restriction does not exist on the Nohau S12X full-emulator, which detects PLL changes automatically and allows unlimited number of PLL frequency changes during run-time)

## Getting Started With S12X BDM

To input the S12X PLL settings into SeeHau, on the main menu Config" item, select "Emulator ...". The "Emulator Configuration" window will show.

To program 40 MHz bus speed in the case of the Freescale / Motorola evaluation board, do the following:

Input "9" in the "Loop Divider Reg. (hex)" field. This is the setting that will be programmed by SeeHau to the S12X - SYNCR register. The field is in Hex.

Input "0" in the "Reference Divider Reg. (hex)" field. This is the setting that will be programmed by SeeHau to the S12X - REFCDV register. This field is in Hex.

*Note: The SeeHau settings must agree with your code settings for the SYNCR and REFCDV registers, if your code sets these registers.*

Check the "PLL in Use", in order for SeeHau to engage and select the PLL.

The bus speed with the PLL engaged is calculated according to the formula:

$$F_{\text{bus-pll-mode}} = F_{\text{extal}} * (\text{SYNCR} + 1) / (\text{REFCDV} + 1)$$

You should see this:



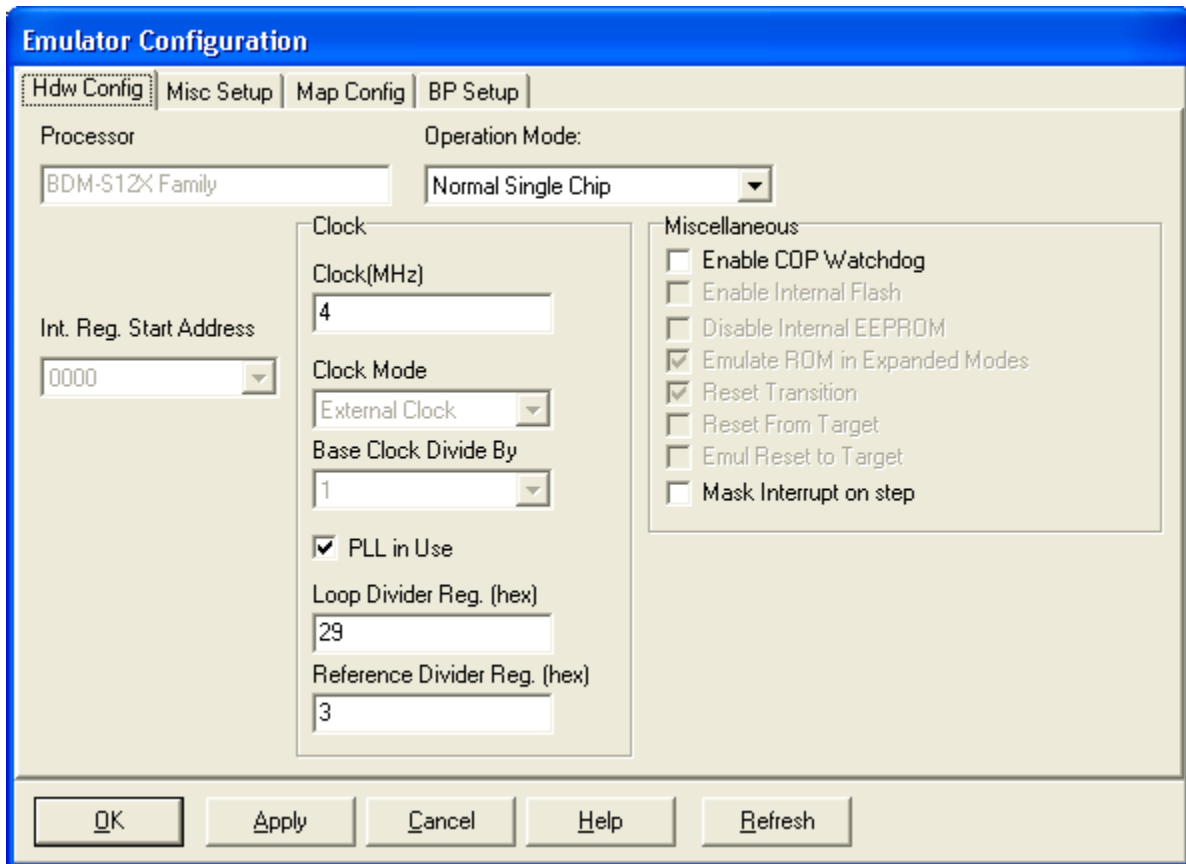


Figure 18 – PLL Configuration

Now click on the “OK” button. to run at the new PLL settings.

Seehau resets the target, programs and selects the S12X PLL. The S12X bus speed should now be 40 MHz.

*Note: Likely reasons to receiving a Seehau error message in this stage are:*

*Not having a proper network of capacitors and a resistors connected to the S12X - XFC pin.*

*Entering wrong values to the Loop Divider and Reference Divider field, that created too high or too low PLL frequency.*

To realize the new 40 MHz bus speed, click on the “GO” button, and observe that the clock now ticks at a much faster rate, since a higher bus rate is now selected.

In the case of the Freescale / Motorola evaluation board and the configuration given above, the S12X now runs 20 times faster than it was running before the PLL was engaged.

### ***Viewing memory content during Runtime.***

Seehau offers the ability to view the S12X memory content during run-time (when user code executes).

Open a new data window, but clicking on the “DAT” icon on the toolbar.

A new data window will open. Resize and move this new data window to your preferred location on the screen.

Right-click in the Data window, select “Address Space”, and then “GLOBAL RUN TIME”. This configures the data window to be updated during run-time.

Select the address field on the left bottom side of the data window, and enter “show”. This will position the data window to start at the variable “show” (any numeric address can be entered too).

Right-click in the Data window, select “Settings ...”, and then “Runtime timer for current window...”. This field shows the update rate for the data window in milli-seconds. Enter “100” to update every 100mSEC and click “OK”.

Start running the S12X by clicking on the “GO” button.

The Data window will update as the S12X runs, and show the S12X memory changes.

Right-click in the Data window, select “Display As ...”, and then select “ASCII” and click “OK”. The data window will now show the numeric clock display updating.

## Getting Started With S12X BDM

You should see something like this:

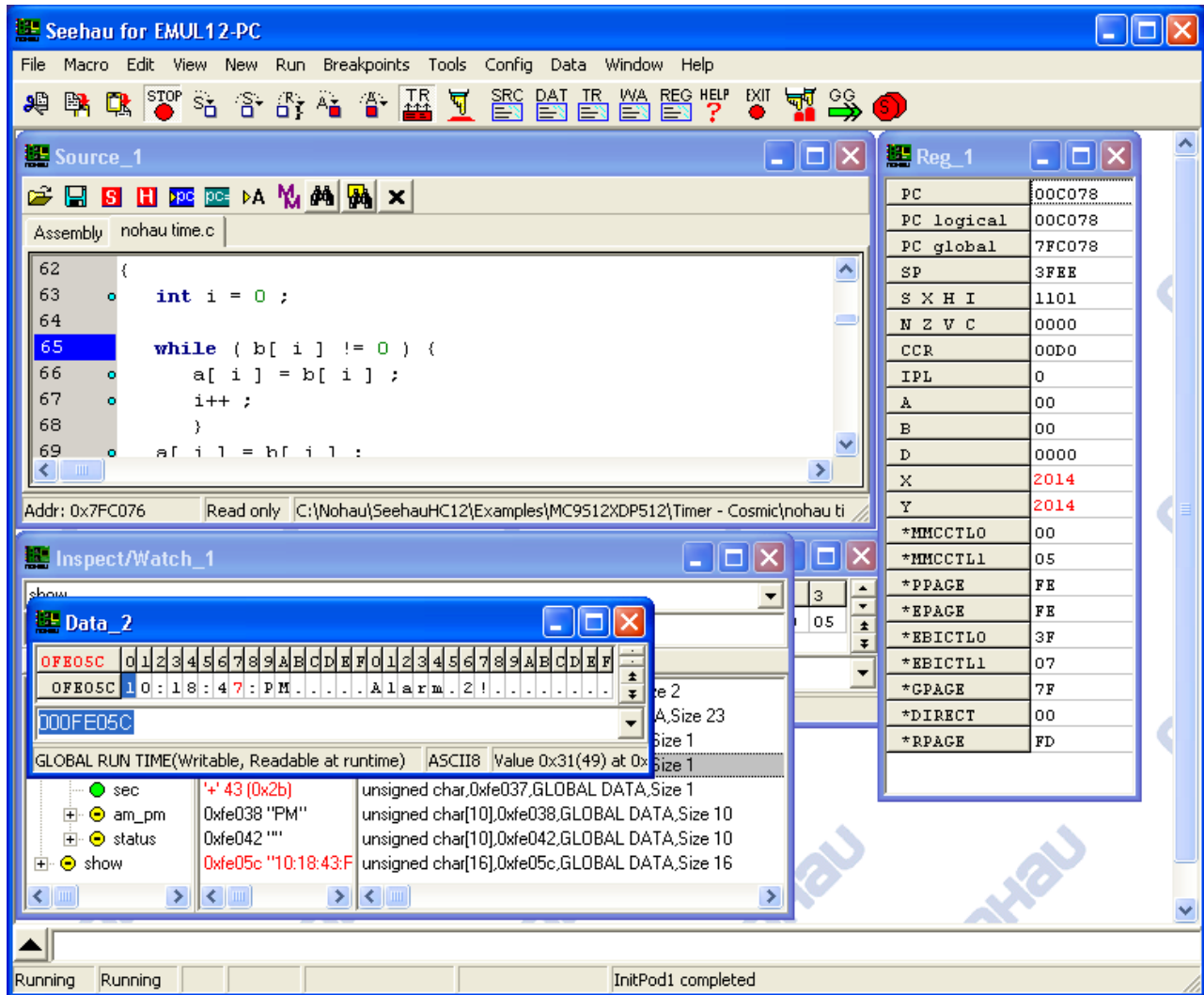


Figure 19 ASCII Data Window;

## Closing SeeHau and saving configurations

Click on the Close box with an "X" in the top right corner of the SeeHau HC12 main window.

You should get this window:

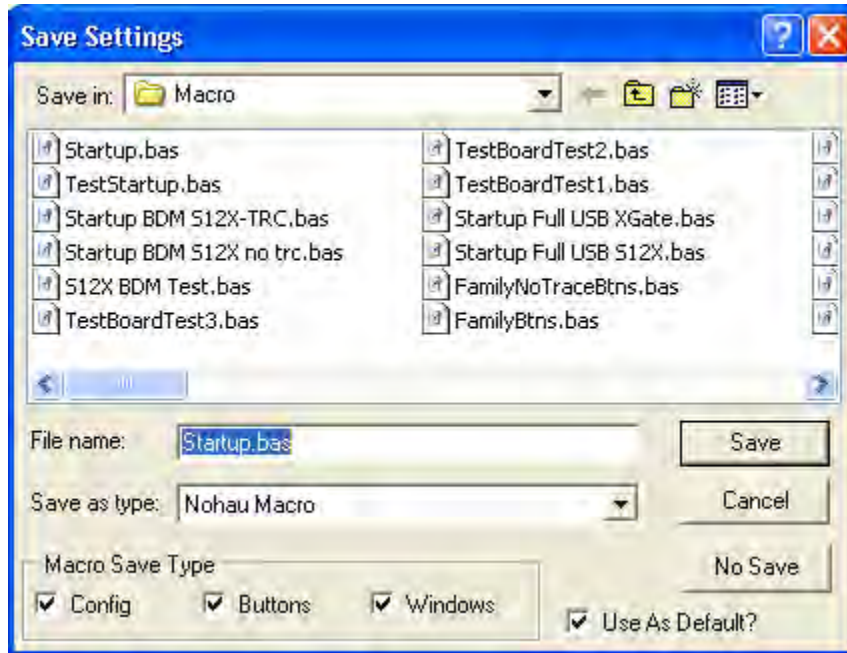


Figure 20 – Save Settings Dialog

Just click on "Save". Because "Startup.bas" already exists, you will get this window:

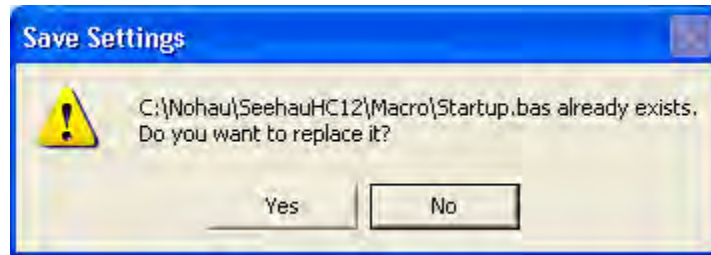


Figure 21 – Save Settings Message

Click on "Yes" and Seehau save your settings and exit.

You can start Seehau HC12 again, and your settings will be re-created, and you can continue from where you left off.

## ***Configuring for XGate debugging***

### **“Multi Core” – How Seehau Helps You Debug a Multi-Processor Target**

Seehau HC12 has two major parts, The visible part that deals with displayed windows, your mouse and your keyboard is called the “GUI”, for “Graphical User Interface”. The invisible part that deals with the target hardware and its state is called the “Core”.

Since the S12X and the XGate are nearly independent each one has its own “Core” to keep track of its registers, debug information and running state.

The windows that Seehau HC12 displays are associated with a specific core, and the core’s name is displayed as part of the window caption. The attention of the “GUI“ is automatically focused on the core that owns the active window. When you click on a source window, the core that owns the window is made the active core.

When Seehau makes a window active because of some target action, the core that owns the window is automatically selected. For example, when the target program gets to a breakpoint, the source window for that core is activated and positioned to the breakpoint.

You can select the core that is active either by clicking on any window that belongs to that core, or by using the main menu “Multi Core” item.

### **Configuring for Multi Core**

So far, we have just covered S12X debugging. For debugging XGate co-processor programs and their interaction with the S12X processor, you will need to do some more configurations.

In the instructions below the file paths are given as the Seehau defaults for simplicity. If you have used different paths, please substitute your install directory for “c:\Nohau\SeehauHC12”.

### **Rename a Good S12X Configuration**

Get a Seehau configuration you are happy with and exit Seehau, saving settings. Open the directory “C:\Nohau\SeehauHC12\Macro”. Rename the file “Startup.bas” to “Startup S12X.bas”.

### **Make a XGate Configuration**

From the Windows Start button select, the menu items “Programs”, “SeehauHC12” and “Config” in that order. Select connection type as you did for the original configuration. For the processor, select “BDM-S12X XGATE”. Take the default settings for the rest of the configuration, except for the clock frequency. Set the clock frequency to the frequency of the S12X external crystal or clock. Click on “Finish” and select “No” for start emulator. Open the directory “C:\Nohau\SeehauHC12\Macro”. Rename the file “Startup.bas” to “Startup XGATE.bas”.

### Configure the Seehau “Multi-Core” Feature.

In the “C:\Nohau\Seehau HC12\” directory, double click on “MultiCoreConfig.exe”. Put 2 in the “Number of emulators” field. Your window should look like this:

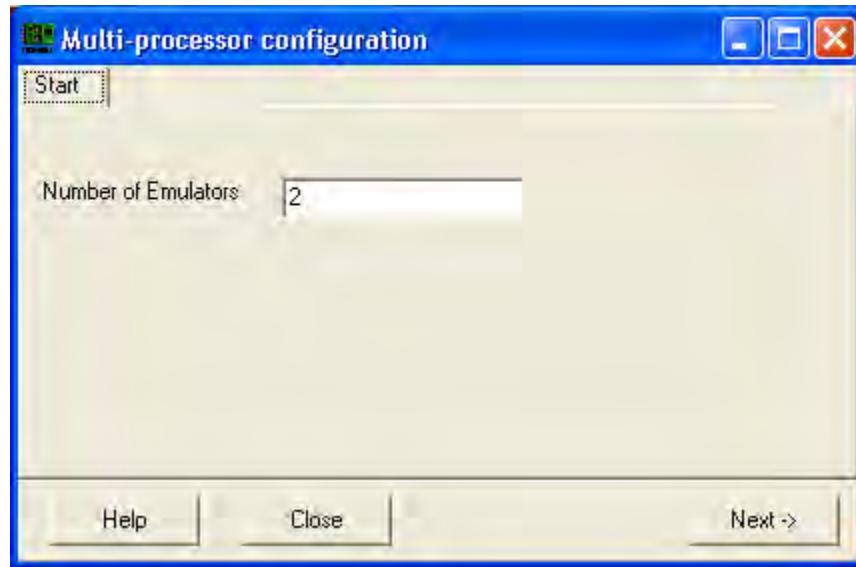


Figure 22 – First MultCoreConfig Window

Click on “Next” button. Click on the “...” button to the right of the “Startup Macro” field. Select the “Startup S12X.bas file that you just created. In the Core Name field enter “S12X”. Then window should look like this:

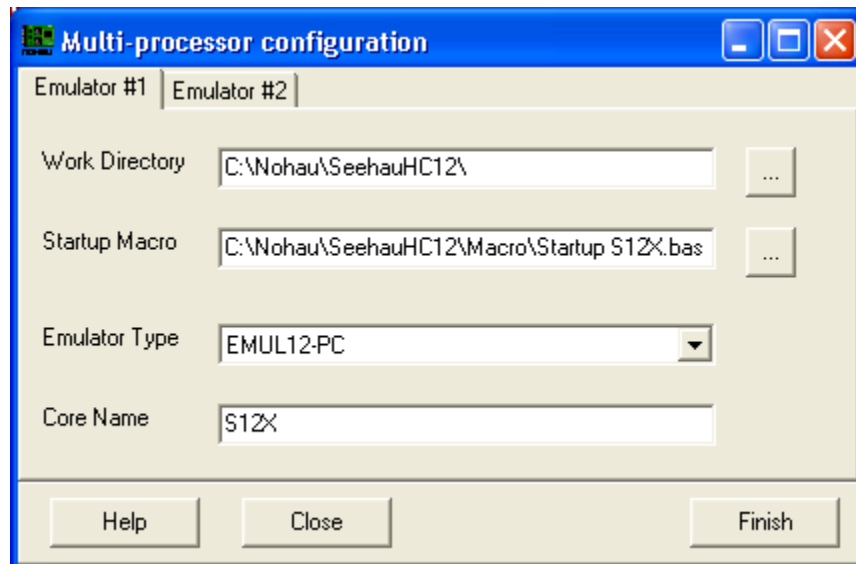


Figure 23 – Emulator #1 Tab

## Getting Started With S12X BDM

Click on the “Emulator #2” tab. As you did for the S12X, browse to the startup macro “Startup XGATE.bas”, and set the “Core Name” to “XGATE”. The window should look like this:

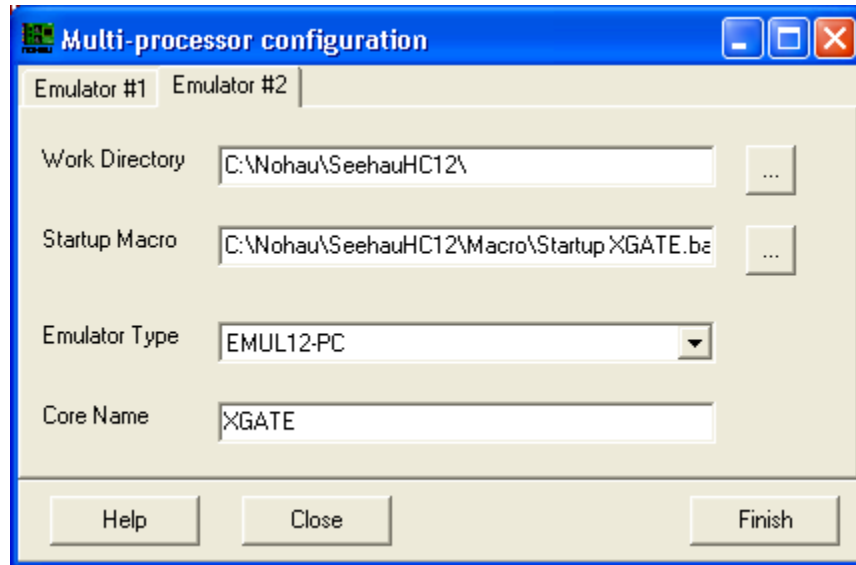


Figure 24 – Emulator #2 Tab

Click on the “Finish” button. You should see this information box:

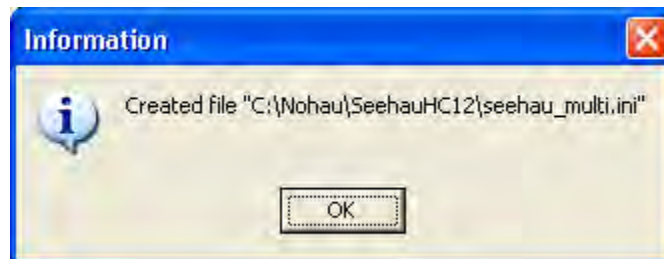


Figure 25 MultiCoreCfg Done

Click on “OK”.

## Make a Seehau HC12 Multi-Core Shortcut

Right Click on the Seehau HC12 program icon in the “Start – Programs –SeehauHC12” menu sequence. Select the menu item “Create Shortcut”. The Menu Item “Seehau HC12 (2)” will be created. Right-click on this new menu item. Add a space and “-multi” at the end of the “Target” field. The window should look like this:

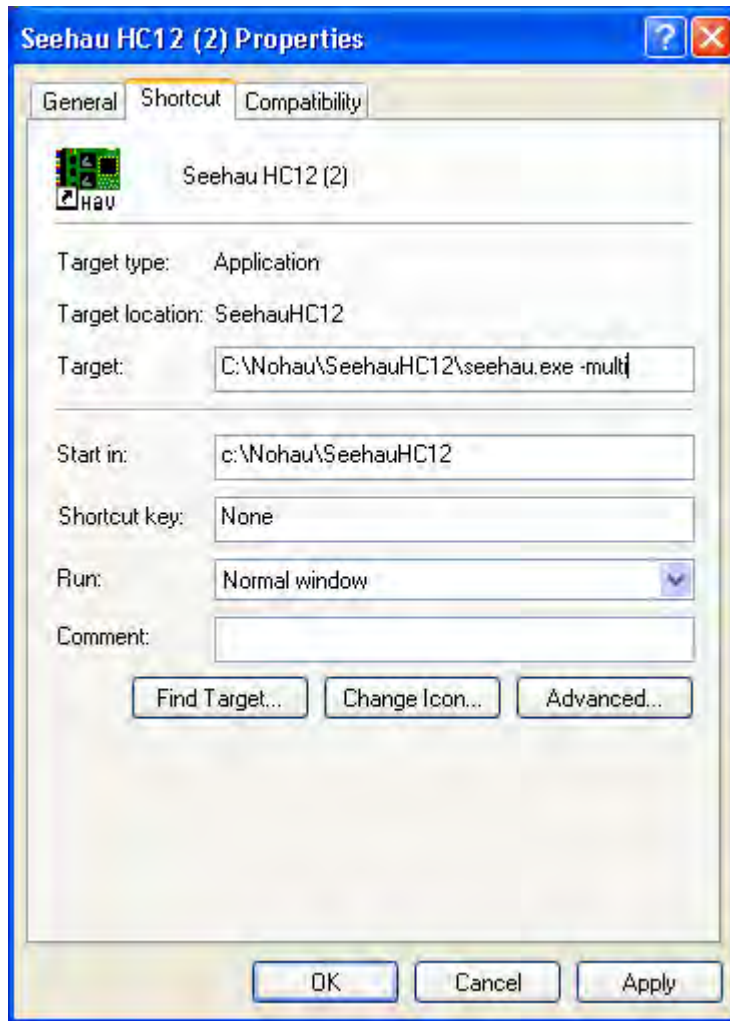


Figure 26 – Making a Shortcut

Click on “OK”.

Depending on what version of Windows you are using, you may receive a warning that “This change will affect all users, do you want to continue?”. If you get this message, click on “yes”.

Go through the “Start – Programs – SeehauHC12” menu sequence again and right click on the “Seehau HC12 (2)” item . This time select “Rename” and change the name to “Seehau S12X XGATE”



## Start “Seehau S12X XGate”

Double click on the “Seehau S12X XGATE” item in your “Start – Programs – Seehau HC12” menu sequence.

After a lot of action you should see something like this:

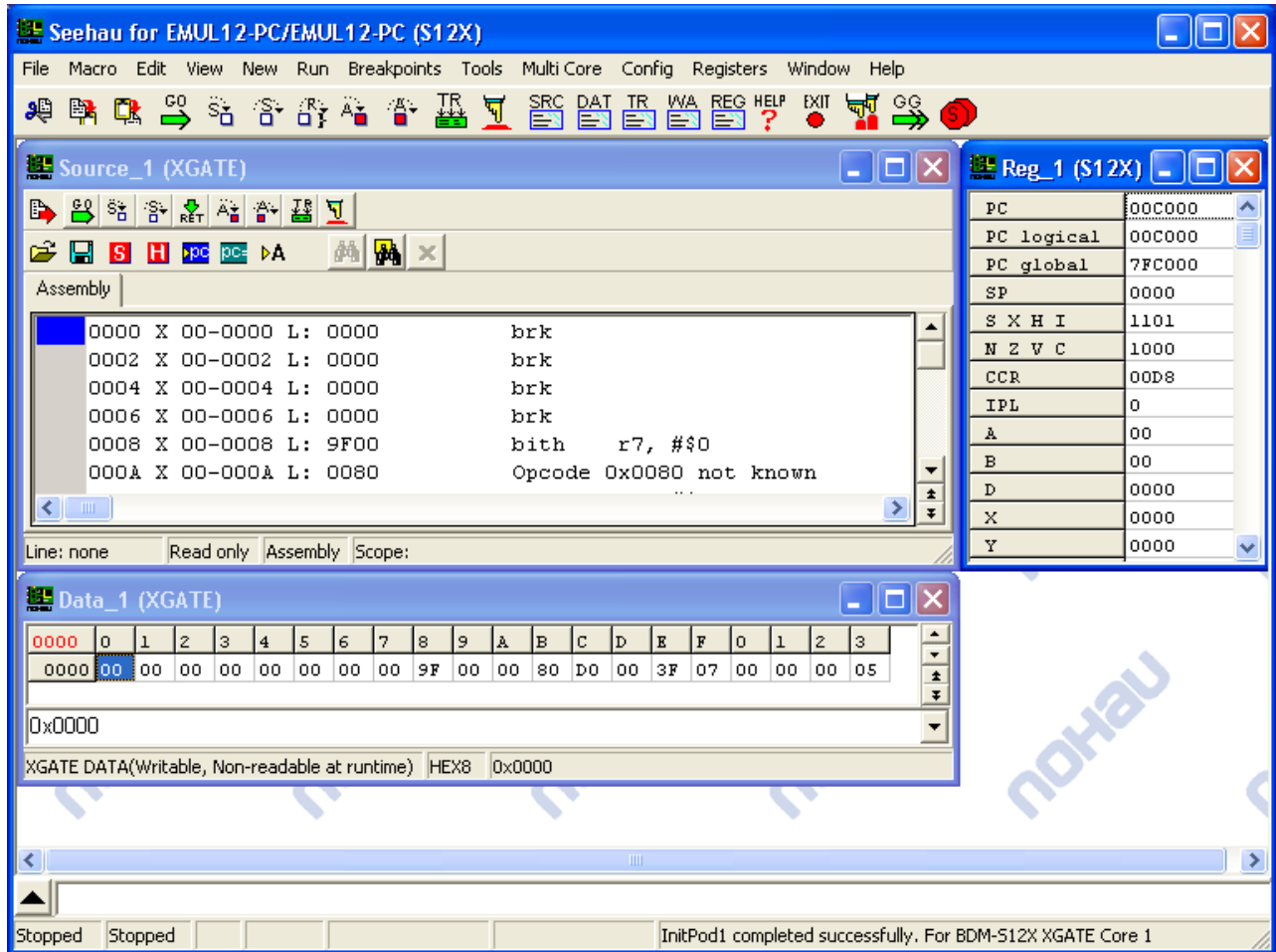


Figure 27 – Initial Multi-Core Windows

The window captions have the processor name in parenthesis.

Now Sort out the overlapping windows. As you can see the S12X and XGATE windows start out overlapping each other.

## Getting Started With S12X BDM

Here is one way of arranging the windows:

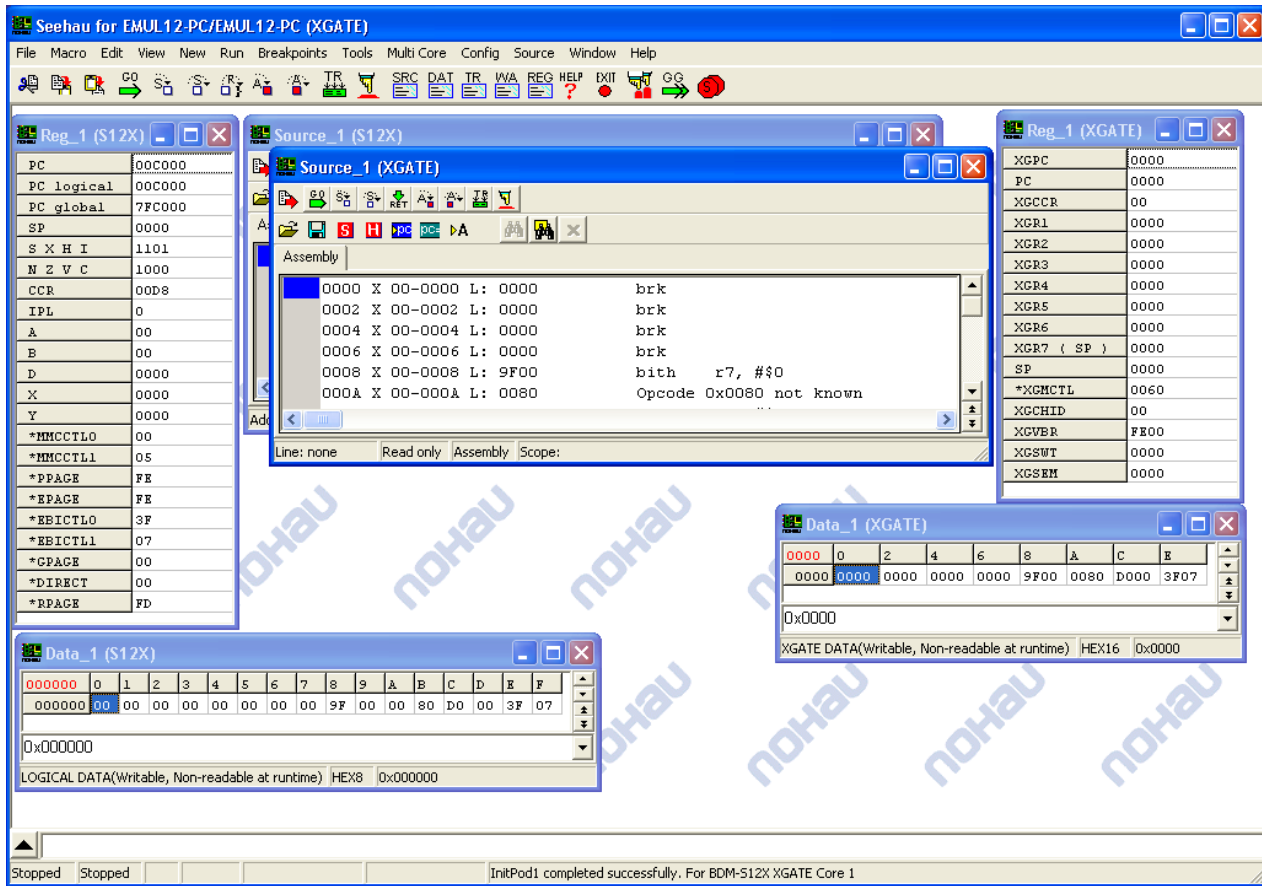


Figure 28 - Re-Arranged Windows.

## Save Settings

When you are satisfied with the layout, click on the close “x” in the upper right corner of the main window. You will be asked if you want to save settings twice, once for the S12X and once for the XGATE. Answer yes to both questions.

## Restart “Seehau S12X XGate”

Double click on the “Seehau S12X XGATE” item in your “Start – Programs – Seehau HC12” menu sequence.

## Getting Started With S12X BDM

### Select the S12X Core

On the main SeeHau HC12 menu, click on the “Multi Core” item, select “Select Active Core –“ and click on “0 – S12X” like this:

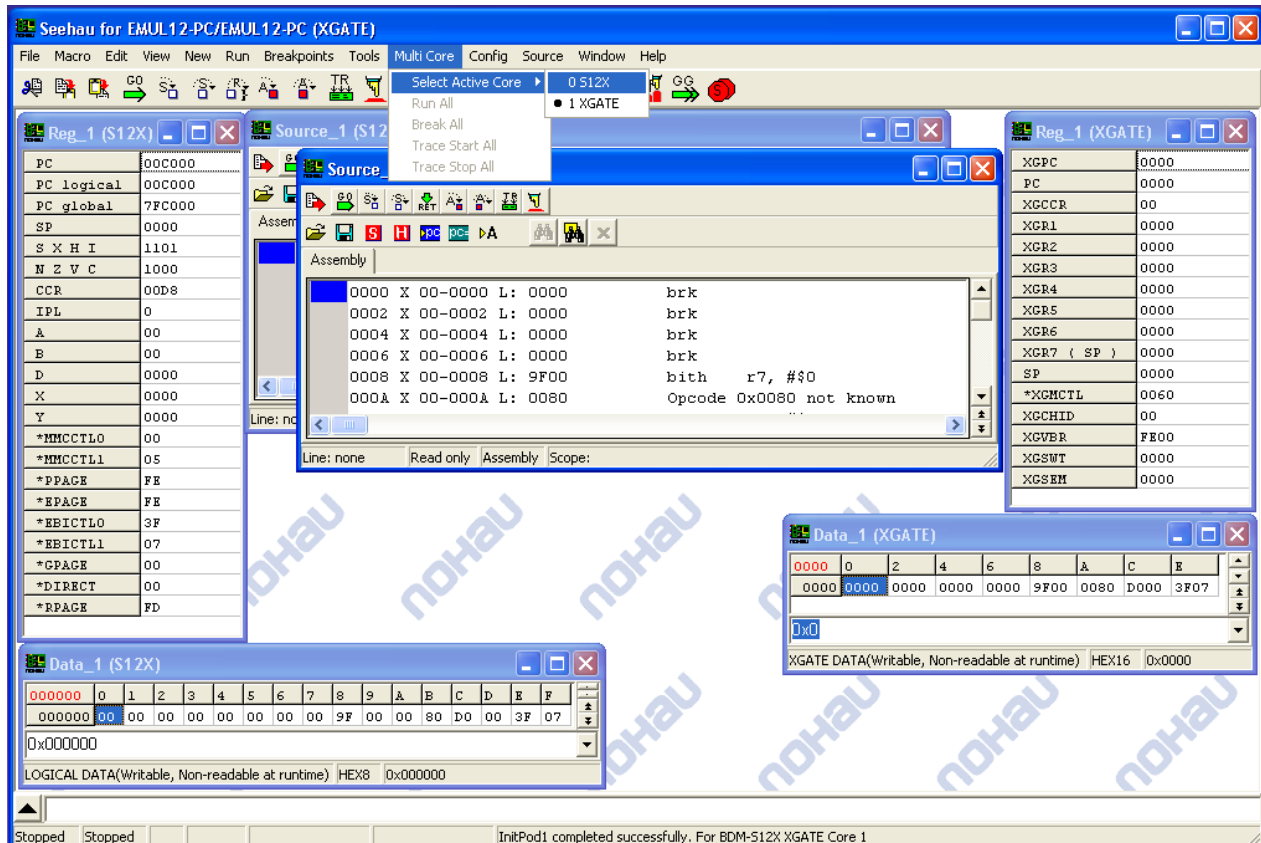


Figure 29 – Menu Selection of Active Core

### The Example Program

This example program has the S12X fill an array, and then has the XGate sum the values in the array. When the XGate has completed the sum, it returns the sum to the S12X program, and the program repeats using different numbers on each pass.

This program does not change the reset values of the IO registers and does not do any IO, so it should run on any MC9S12XDP512 target system without problems.

### Flash Programming

Flash programming works as it did for S12X single core, but it can be done only in the S12X core, due to the very limited XGate access to flash.

## Getting Started With S12X BDM

Make sure that the S12X is selected, and program the file “...\Examples\MC9S12XDP512\Cosmic S12X v2 Example 2\Cosmic S12X v2 Example 2.elf”.


### Loading Debug Information in a Multi Core Configuration

Just as for the S12X only, the debugging information needs to be read by Seehau. Select “File – Load Code” and the same file that was programmed into flash. Make sure that “Load Code” is not checked, and that “Load Symbols” is checked. “Verify Loaded Code” can be checked to be sure that the contents of flash match the debug information in the load file. Click on “Open”. This will load the S12X debugging information into the S12X core.

Now click on some XGate window to make the XGate the active core.

Again select “File – Load Code”, but this time make sure that both “Load Code” and “Verify Loaded Code” are NOT checked. “Load Symbols” should be checked. Click on “Open”. The XGate debugging information will be loaded into the XGate core.

### Resetting a Multi Core Configuration

The startup for the multi core configuration is different, because the S12X startup code needs to run to transfer the XGate program into RAM so that it can be seen by the XGate. The special “ResetReset” icon  on the tool bar resets the S12X and runs the S12X startup code. Click on it now.

## Getting Started With S12X BDM

You should see this:

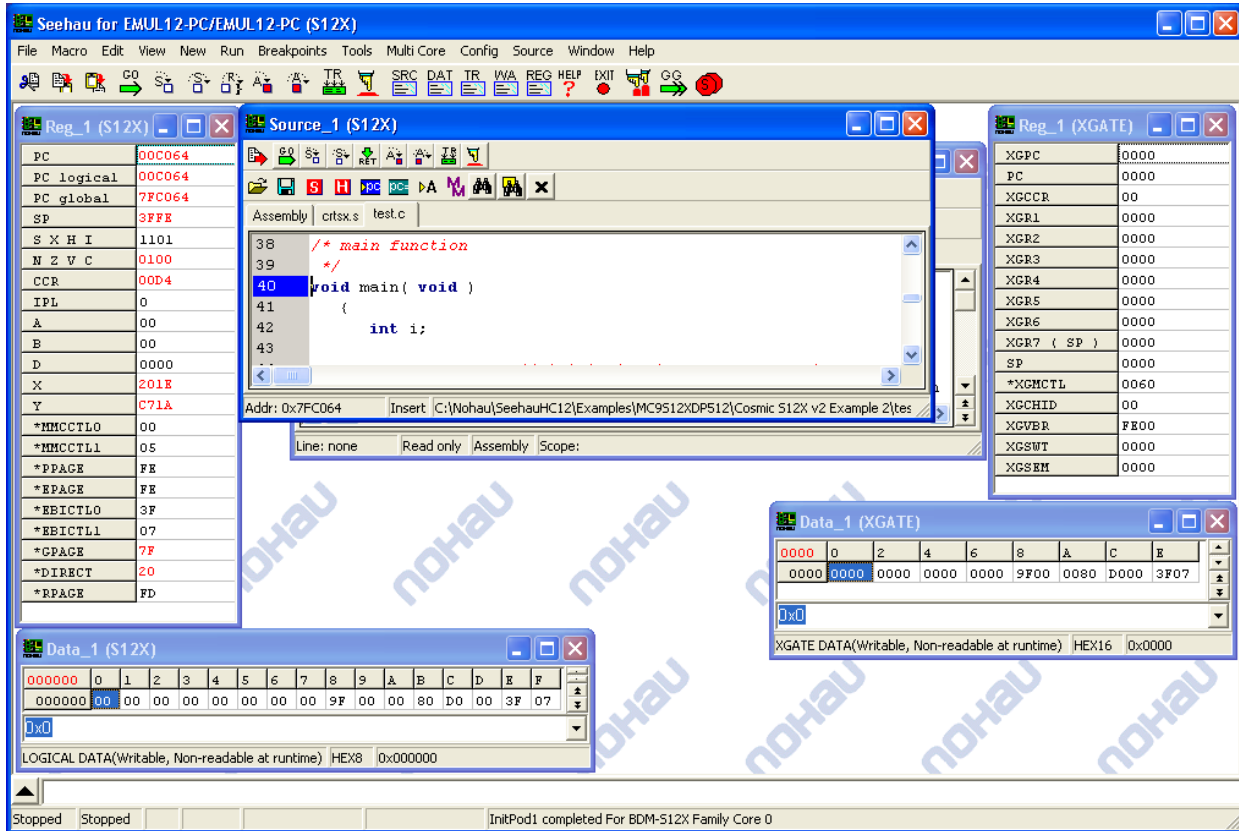



Figure 30 – After Multi Core Load and Reset

## Debugging the S12X Program

Now scroll down in the S12X Source window to the line `sync = 0;`. Click in the left margin of the source window to place an on-chip hardware breakpoint at that line. Now click on the “GoGo” icon  on the tool bar.

Emulation should stop at line 52.

Now double click on the symbol “tab” in line 51 and type Control I. This will open an S12X watch window with tab in it. Double click on + next to the tab in the left pane of the watch window to expand it.

## Getting Started With S12X BDM

Re-arrange the windows to get something like this:

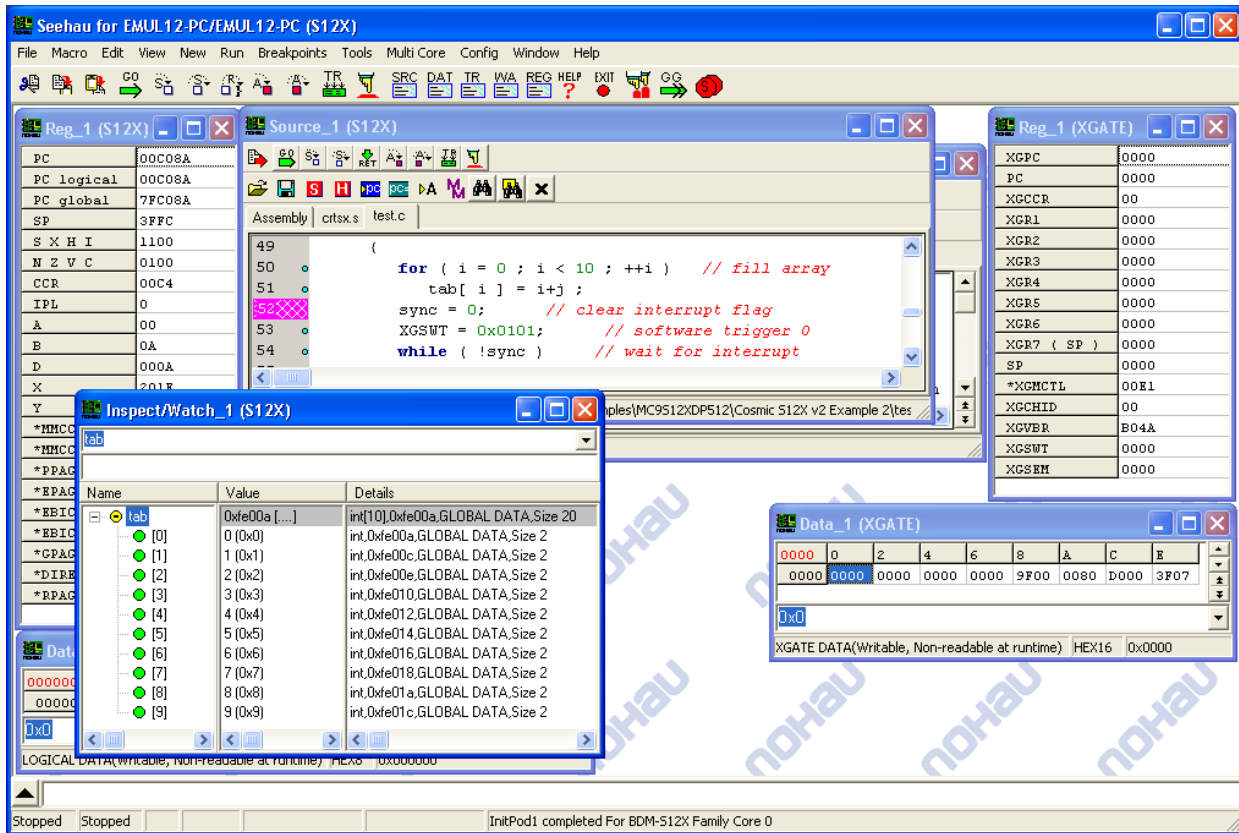


Figure 31 – Tab array expanded.

## How the S12X Starts the XGate Processing

This program illustrates one way of communicating between the S12X and the XGate.

The method used here is not necessarily the most efficient or the “best”. It has elements that appear in most methods, and it does not have any problems in this test program.

The “sync” variable is entirely under the control of the S12X. It is zeroed at this point in the code, and later will be set to 1 will be set to 1 by the code at line 67 when the S12X processes the XGate software interrupt that indicates that the XGate has finished summing the “tab” array. When “sync” is recognized as 1 in the while at line 54, the result will be read and the process will repeat with slightly different data.

The activation of the XGate is caused by line 53 “XGSWT = 0x0101” which sets software trigger 0 for the XGate.

## Looking at the XGate Code

Activate the XGate source window. It can always be activated from the “Window” item on the main menu.

## Getting Started With S12X BDM

Right click in the XGate source window, and select “File – Open File ...”. Open the file xtest.xc in the “...\Examples\MC9S12XDP512\Cosmic S12X v2 Example 2” directory.

Go to line 26 “Sum( tab, nb, &res );” and click in the left margin to place a software breakpoint there.

Because XGate code in RAM software breakpoints work and allow as many XGate RAM breakpoints as you want. Using software breakpoints allows all of the on chip hardware breakpoints to be used for debugging code in flash.

You should see something like this:

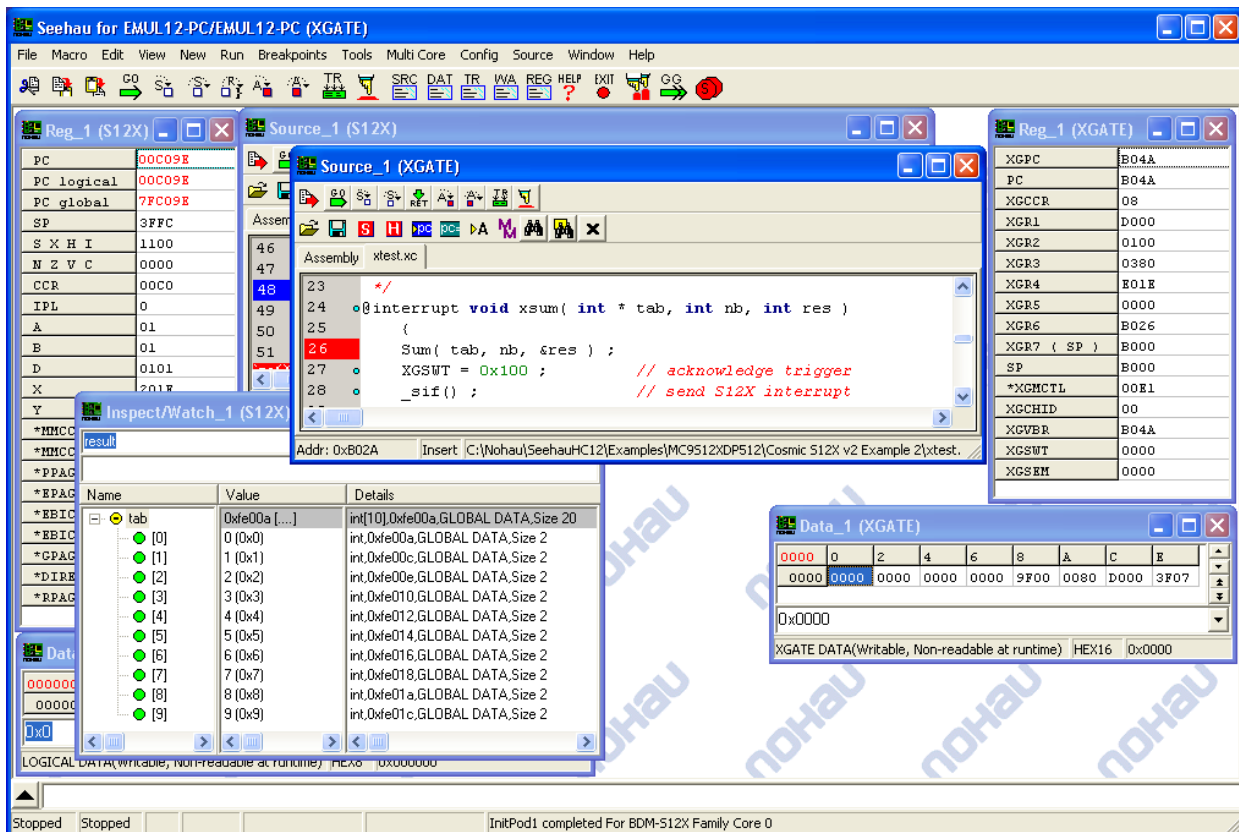


Figure 32 – XGate Software Breakpoint Placed

## Getting Started With S12X BDM

Scroll down to the end of `xtest.xc`, put your cursor in the symbol “`xargsum`” on line 41 and type Control I. Expand “`xargsum`” in the XGate watch window and re-size the window to get something like this:

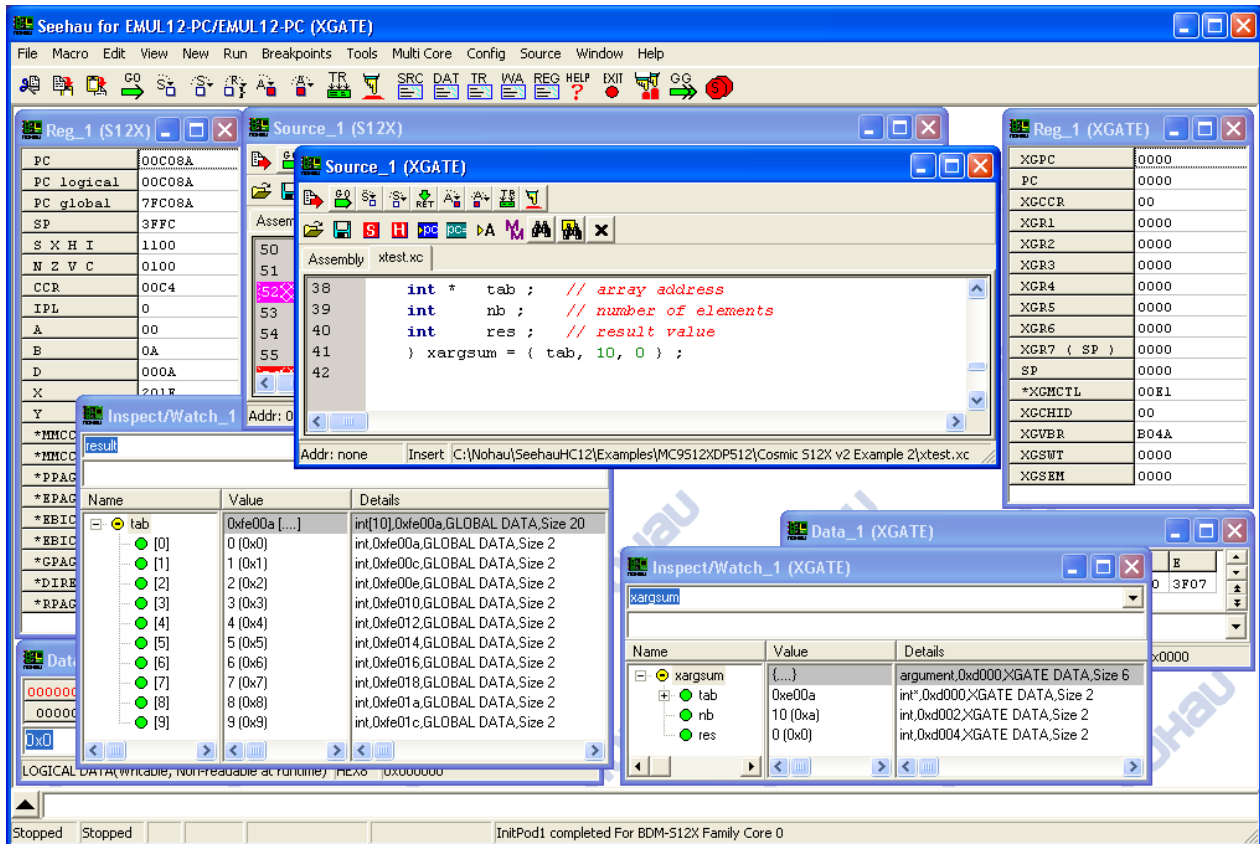






Figure 33 – XGate Watch Window - Structure for S12X and XGate Communication.

For this program “`xargsum`” is the structure that is used for communicating between the S12X and the XGate. You can see the `nb` element is set to 10, the number of bytes to sum.

Now click on the “GoGo” icon  on the tool bar. The emulation should run for a short time, and the both S12X and XGate will stop.

Activate the XGate Source window. The XGate will be stopped at the breakpoint you placed on line 26.

## Single Stepping with Multi Core

It is possible to step either the XGate code or the S12X code individually. This is done with the step control icons  and  in the toolbar at the top of the SOURCE window. Step over that call to the function `Sum`, by clicking on the “Step Over” icon .



## Getting Started With S12X BDM

You should see something like this:

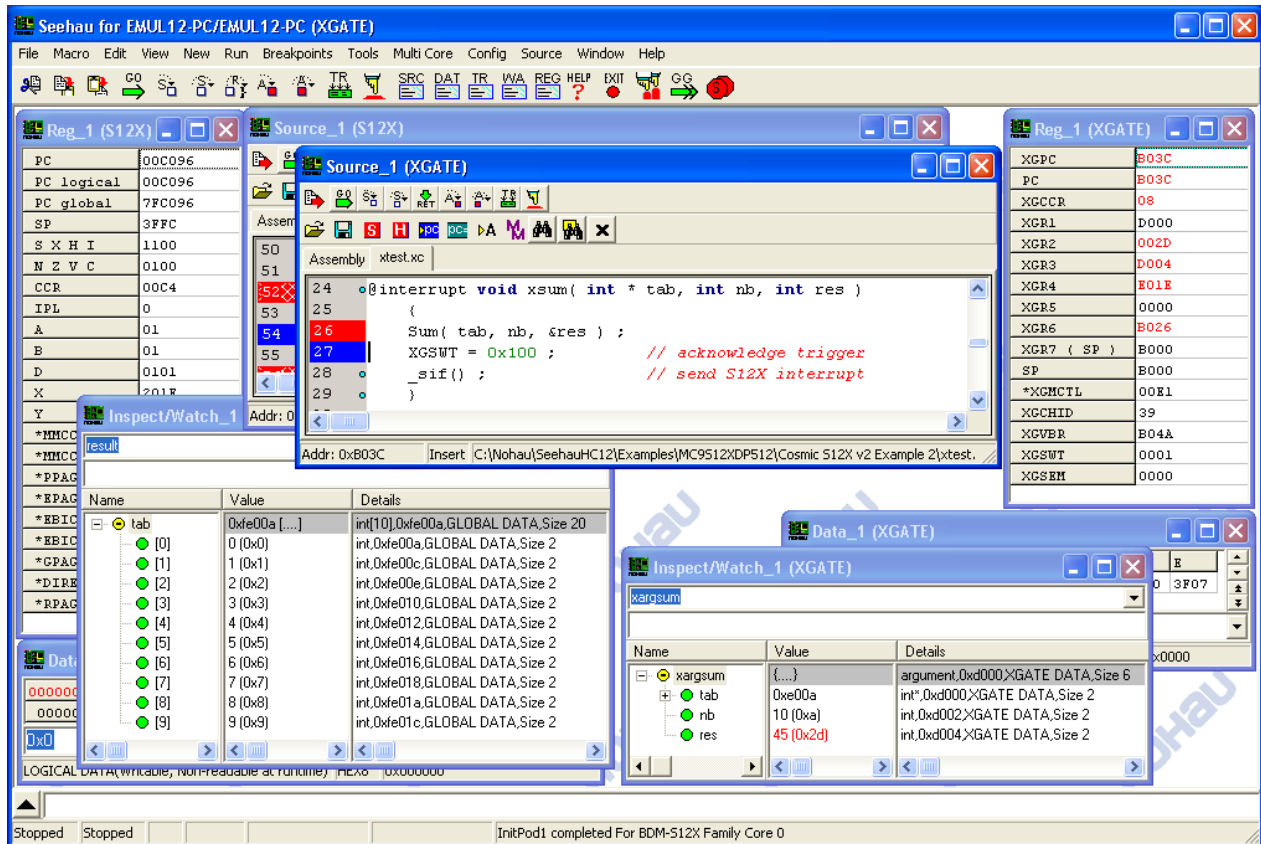




Figure 34 – After XGate Step Over.

Notice that the “res” element of the “xargsum” function has changed to 45, which is the sum of the “tab[]” array.

Now activate the S12X Source window, and place a breakpoint on line 56 “result = xargsum.res;”, add “result” to the S12X watch window and click on the the “GoGo” icon  on the tool bar. After the emulation stops on line 56, click on the “Step Over” icon  in the S12X source window toolbar.

## Getting Started With S12X BDM

You should see something like this:

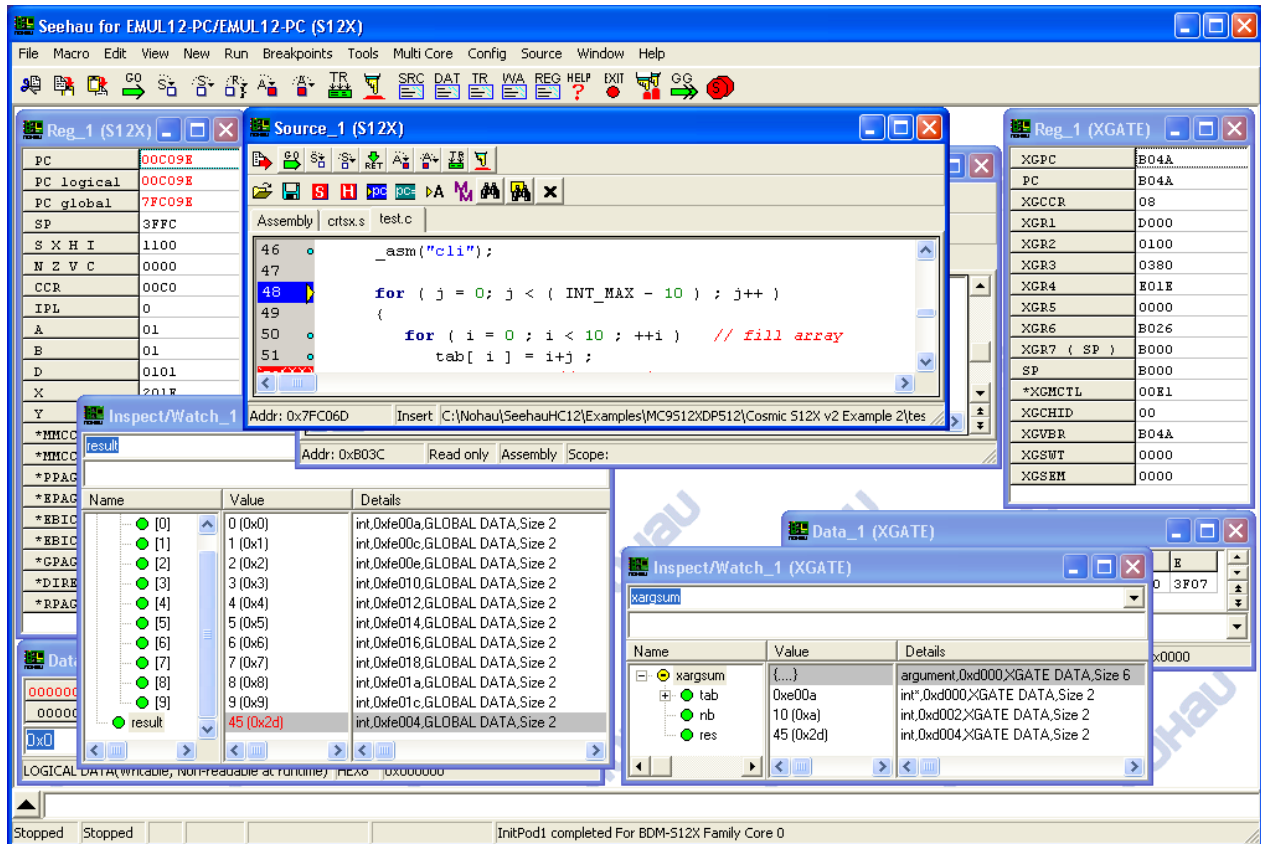





Figure 35 – After S12X Source Step

Notice that result in the S12X watch window has changed to 45.

## Summary of Multi Core operation

You have just run, and source stepped code in both the S12X and XGate processors and examined C variables and data structures for both the S12X and XGate C code.

An important point to remember is that the run controls for multi core operation are different from the single core ones. They are the “ResetReset” icon , the “GoGo” icon  and the “StopStop” icon  on the main SeeHau toolbar for resetting, starting and stopping both processors, and the buttons on the source window toolbar for stepping the processors individually.

## Using the On Chip Trace Feature

The EMUL12-PC/BDM-S12X-TRC pod supports the S12X on chip trace feature. The trace windows are owned by either the S12X or the XGate, and show the available trace information for their processor.

(More material will be added here in future releases.)

## Understanding the S12X Family

### *The S12X Hardware Features*

#### **Memory Addressing**

This subject is complicated and confusing. The complication is a side effect of having a memory efficient, tried-and-true processor design that can be produced in high volume at low cost. We will try to reduce the confusion in this section.

Nohau has over 15 years of experience dealing with complicated addressing in micro controllers and gives you tools to help you see through the confusion of addressing systems.

Each of the different addressing systems is a different way of relating a range of numerical addresses to a range of bytes in a piece of hardware. Each was developed because it was much more convenient for some hardware or software function.

Some addresses in an addressing system may have no bytes associated with some addresses. A particular hardware byte in some addressing systems may be addressed with several different numerical addresses.

The S12X and the XGate co-processor operate on somewhat different 16 bit addresses.

The S12X on chip breakpoint registers use global addressing.

As of December 2005 the Metrowerks S12X reported debugging information in banked addresses and the Cosmic S12X support reported debugging information in global addresses.

The S12X memory mapping design provides for considerably larger devices than are currently feasible, so some of the addresses are not available on current S12X parts.

Some of these addresses can be used to address external memory-like devices, and some will be used in the future to address larger on-chip memories.

#### **S12X CPU 16 bit Addressing**

The S12X CPU has 16 bit address fields in instructions and operates on 16 bit addresses. To allow access to more than 65 k bytes of memory, some ranges of these 16 bit addresses are “memory windows” into a larger section of memory.

Exactly what larger section of memory is determined by some number of the least significant bits of the address within the memory window, and by the setting of a “page” register associated with the particular memory window.

In particular:

## Getting Started With S12X BDM

S12X CPU16 bit address range	S12X Page Register	Window Name
0x0800 through 0x0BFF	EPAGE	EEPROM Window
0x2000 through 0x3FFF	RPAGE	RAM Window
0x8000 through 0xBFFF	PPAGE	Flash Window

There are some conventions that the Freescale S12X design follows in all of these paged areas:

- The page registers have 8 bits.
- The on-chip pages start with 0xFF and extend to consecutive lower numbers from that page.
- Some of the highest numbered pages are also visible in fixed 16-bit addresses outside the paged window and independent of the related page register setting. These pages can also be selected in the related paged window when the related page register is set to the proper page number. When this occurs, the same byte can be addressed both by a 16 bit address in the fixed area and by a different 16-bit address in the paged window.
- The fixed pages are next to the page window. This means that all of a particular type of memory that can be addressed with 16-bit addresses is in a single range of 16 bit addresses..

There are some global addresses that do not have an equivalent 16-bit address and page register setting. They can only be addressed with the combination of the GPAGE register and the S12X new gld\* and gst\* instructions.

The different memory mappings available are documented in the MC9S12XDP512 documentation mentioned at the beginning of this guide.

## XGate 16 bit Addressing

The XGate co-processor sees a 16-bit address space that includes RAM, the CPU registers and a part of flash memory. (The first S12X silicon, mask set 0L40V, cannot access flash memory.)

Because the XGate can execute out of RAM at more twice the speed it can execute from flash, the preferred location for XGate code is RAM.

On a reset, the startup code in flash memory executes on the S12X CPU. This code configures the S12X and the XGate.

This S12X startup code must copy the necessary XGate code from flash into RAM. The startup code can also set which parts of RAM are available only to the S12X, which parts are available only to the XGate co-processor, and which parts are shared by both. See the chapter titled “Memory Mapping Control” in the MC9S12XDP512 Data Sheet. In particular see the functions of the RAMWPC, RAMXGU, RAMSHL and RAMSHU registers.

See the Freescale S12X documentation for the XGATE for an explanation of addressing in the XGate co-processor hardware. As of December 2005 this is in the Freescale document “MC9S12XDP512\_V2”.of October 2005 in section 21.4.2.4.1, titled “Expansion of the XGATE Local Address Map”

### Banked or “Logical” Addressing

This addressing is one of the addressing methods traditionally used with HC-12 and HCS-12 parts with more than 64 k bytes of flash memory.

The banked address has 24 bits, divided into 2 sections. The least significant 16 bits are the 16-bit address that the CPU instructions use for accessing the location. The most significant 8 bits are the value of the relevant page register.

If the 16-bit address is not in a paged window, then the most significant 8 bits are ignored and can have any value. If the 16-bit address is not in a paged window all possible page values will refer to the same memory location. When an address is not in a paged window, Seehau HC12 will set the most significant 8 bits to zero.

Since some sections of the 16 bit address space are “fixed” windows that access flash, EEPROM or RAM. These windows are not “paged” windows, but they access “fixed” pages, such as page 0xFF. This means that the locations in these pages can be accessed by two different addresses, one in the “fixed” window and one in the corresponding “paged” window. This is another way that two different banked or “logical” addresses reference the same memory location.

Some Examples:

- A. Logical address 0xFA8000 accesses internal paged Flash in the program window, by the HCS12 CPU instruction specifying address 0x8000, and the PPAGE register (which is the relevant paging register in this case) being set to 0xFA.
- B. Logical address 0x FC1800 accesses internal paged RAM in the Ram window, by the HCS12 CPU instruction specifying address 0x 1800, and the RPAGE register (which is the relevant paging register in this case) being set to 0xFC.
- C. Logical address 0x FE0A00 accesses internal paged EEPROM in the EEPROM page window, by the HCS12 CPU instruction specifying address 0x0A00, and the EPAGE register (which is the relevant paging register in this case) being set to 0xFE.
- D. Logical address 0x003000 specifies the first location of RAM page 0xFF in an un-paged window. This location can also be accessed by logical address 0xFF1000, which is RPAGE value 0xFF and the first location of the RAM window. Since the 16-bit address part of 0x003000 is 0x3000, which is the “fixed” RAM page, Addresses 0x013000 and 0xFF3000 will also refer to this same memory location.
- E. Logical address 0xFE1FFF specifies the last address of RAM page 0xFE in the RAM window. This location can also be accessed at logical address 0x002FFF.

## Getting Started With S12X BDM

- F. 16-bit address 0xFFFFE is the “Reset Vector”. It is in the range of 0xC000 through 0xFFFF, which can also be accessed by setting the PPAGE register to 0xFF and reading the 16-bit address 0xBFFE.

This location can have the logical address 0x00FFFE or 0xFFBFFE. If you open a “LOGICAL DATA” windows in SeeHau HC12 and look at these two addresses, you will see the same data, because both “logical” addresses refer to the same location.

All logical addresses have also the equivalent Global addresses that can be used to access these memory locations using the GPAGE register and the S12X new GLDx and GSTx instructions.

## Global Addressing

Global addressing is used on the internal memory bus in S12X processors.

The S12X CPU accesses global memory by setting the GPAGE register and then executing the new gld\* or gst\* instructions. This allows access to some memory provided for by the S12X hardware that cannot be accessed by other instructions, and cannot be accessed with 16 bit addressing or banked addressing.

A global address has 23 bits. The least significant 16 bits are those in the 16 bit address of a gld\* or gst\* instruction that references the location. The most significant 7 bits are the corresponding 7 bits from the GPAGE register setting that references that location.

The rules for mapping from global addresses to 16-bit or logical addresses are cleverly designed to make the hardware translation of addresses fast and effective, but they are not simple to explain in words.

For an explanation, the Freescale S12X documentation for the “Module Mapping Control”.

Printing the figure titled “Logical to Global Address Mapping” for reference will be helpful.