

Code Coverage on Nohau's HCS12 and HC12 Full-Featured Emulator

By: Doron Fael – Nohau Corporation
September 1, 2004

This document briefly shows how code coverage is achieved using the Nohau full-featured emulator for the Motorola/Freescale HCS12 and HC12 family.

Code-Coverage is the emulator's ability to analyze which source lines and code lines are executed, which lines are not executed and which lines are partially executed. Code coverage is needed in applications that require very high reliability, and need to be verified so that every source line has been successfully exercised at least once, in order to comply with international standards of software integrity and reliability.

Code-coverage information is collected by the trace-unit of the full-emulator. The trace monitors the bus activity and decodes the instruction pipeline of the target HCS12 or HC12 processor as it executes the application under test. The trace analyzes which instructions are executed and which are not in run-time and writes this information to a dedicated memory in the trace in run-time. The analysis is aware of the HCS12 PPAGE register and can accurately analyze both non-paged and paged application up to 1Mbyte in size. (On the S12X Family it can analyze applications up to 8Mbyte in size)

When the collection of the code-coverage information is completed the user can stop the code coverage. The code-coverage information is then automatically read from the trace memory to the PC, and the user is asked to input a file-name to save the results of this iteration of code-coverage information.

The code coverage information is then displayed to the user as shown in the screen-shots in the following pages:

- Green source lines have been fully executed
- Red source lines have not been executed at all
- Yellow source lines have been partially executed
- White source lines did not produce any assembly code and therefore are not analyzed.

The Seehau software also allows Code-Coverage information from several iterations to be merged together, to create a complete code-coverage report of many code-coverage iterations.

In the following screen shot we can see fully executed source lines (Green), non-executed source lines (Red), partially executed source lines (Yellow), and source line with no assembly code (White).

As can be seen below, the portion of code that should take care of a change in the minutes has not been executed in this case, since the code-coverage was ran only for a few seconds before a transition of the minutes took place. If the user would run the code-coverage for a longer period, we would see these red (unexecuted) lines becoming green (executed).

The screenshot shows the 'SeeHau for EMUL12-PC - [Code Coverage]' application window. At the top, there is a menu bar with options: File, Macro, Edit, View, New, Run, Breakpoints, Tools, Config, Code Coverage, Window, Help. Below the menu bar is a toolbar with various icons for navigation and execution. The main window is divided into several sections:

- Addresses** tab: A table showing code coverage statistics for 'BarTime.c'.
- Source** tab: A code editor showing the source code for 'BarTime.c' with lines color-coded based on execution status.
- Summary** tab: A status bar at the bottom showing 'Code Coverage Started' and 'Coverage file: C:\nc12\Macro\temp1.txt'.

File name	Executed lines	Not executed lines	Part Executed lines	Total lines
BarTime.c	80(80.00%)	19(19.00%)	1(1.00%)	100(100.00%)
Total in program	80(80.00%)	19(19.00%)	1(1.00%)	100

```
181
182+  ticks++ ;
183
184+  if ( ticks == 250 ) ( // count 250 times 4 mSec = 1 SEC
185+    ticks = 0 ;
186+    if ( timer.sec != 59 ) (
187+      timer.sec++ ;
188%   change_min = 0 ;
189    )
190  else {
191-   timer.sec = 0 ;
192-   change_min = 1 ;
193  }
194
195
196  // second step of resetting the COP.
197+  ArmCOP = 0xaa ;
198+
199
```

The following screen-shot displays the code-coverage results in the source-window in mixed-mode display where both the C source lines and the assembly instructions are intermixed together.

The yellow source line “change_min = 0 ;” is partially executed, as only the first of the two assembly instructions that create this source line has been executed, and the second assembly instruction was not executed (in the particular case shown here because of a breakpoint).

```
002131: FD22AB          LDY     ticks
002134: 02             INY
002135: 7D22AB          STY     ticks
183
184  *   if ( ticks == 250 ) { // count 150 times 4 mSec = 1 SEC
002138: 8D00FA          CPY     #00FA
00213B: 261C           BNE     $2159
185  *   ticks = 0 ;
00213D: 87             CLRA
00213E: C7             CLR    B
00213F: 7C22AB          STD     ticks
186  *   if ( timer.sec != 59 ) {
002142: F623A2          LDAB   $23A2
002145: C13B           CMPB   #03B
002147: 2708           BEQ     $2151
187  *   timer.sec++ ;
002149: 7223A2          INC    $23A2
188  *   change_min = 0 ;
00214C: 7922AA          CLR    change_min
00214F: 2008           BRA     $2159
189  *   }
190  *   else {
191  *   timer.sec = 0 ;
002151: 7923A2          CLR    $23A2
192  *   change_min = 1 ;
002154: C601           LDAB   #01
002156: 7B22AA          STAB   change_min
193  *   }
194  *   }
195  *   // second step of resetting the COP.
196  *   ArmCOP = 0xaa ;
002159: C6AA          LDAB   #0AA
00215B: 5B3F          STAB   ArmCOP
198  *   }
```

Line: 188 | Read only | C:\Nohau\SeeHauHC12\Examples\DP256\BarTime.c\nc12\Macro\temp1.txt | Scope: File: BarTime.c Module: bartime Function: pitr_int

Stopped | Stopped

The Code-Coverage information can also be displayed in raw mode as seen below listing the address ranges of the instruction that have been executed and supplying a summary as seen below. This information is also available in a text file, to allow further processing outside the Seehau software if needed (further processing can also be achieved using Seehau's built in macro language).

