MicroConverter® Technical Note - uC002
Developing in C with the Keil uVision2 IDE.

INTRODUCTION

This technical note describes the use of the Keil uVision2 Integrated Development Environment in the development of a 'C' based, MicroConverter application.

The Keil 8052 compiler package includes uVision2 which is an Integrated Development Environment (IDE) along with all the utilities you may need to create embedded application programs for the MicroConverter family.

An evaluation copy of the Keil C51 developers kit (PK51) is available as part of MicroConverter tools chain or directly from the Keil web-site at http::\www.keil.com. This evaluation copy will create applications that are 2Kbytes and smaller. You may use it to create and test your own target programs and also to evaluate the Keil environment.

CREATING A PROJECT

Before writing any C-code, a project associated with our code needs to be created. This is done by first creating a new folder in the Keil directory in which your project will be saved. Next the Keil uV2 application can be launched and a new project is created. This is achieved by completing the following steps.

- Create a folder named Demo_ADuC in this path : c:\keil\c51\examples\
- Launch the uV2 application. Start -> Programs -> Keiluvision2
- Create a new project. From the main window, choose the 'Project' menu and select 'New project…'. A new window appears as shown below in Figure 1.
- Select the folder that you've created previously (Demo_ADuC) and on the bottom of the window type the name of your new project, eg. Demo_ADuC and press SAVE.

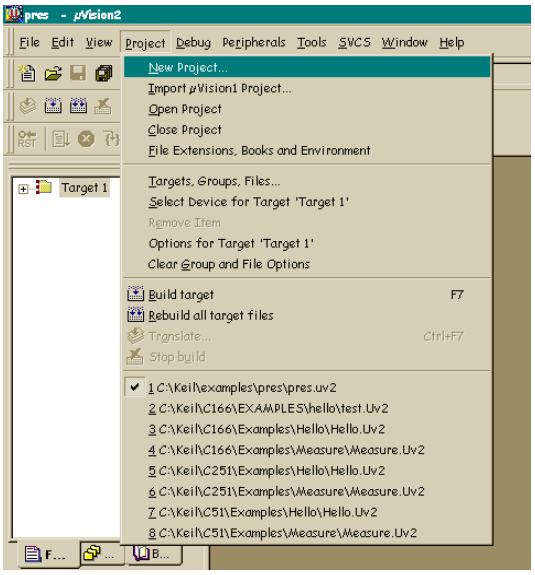

Figure 1.

A new window appears as shown in figure 2 below and you are now required to configure your setup to target the specific MicroConverter device you wish to use (in this example we will be using the ADuC834) and the output file format generated after the compilation stage. This is achieved by completing the following steps.

- Open the ADI folder.
- Select the MicroConverter on which you will be developing, in this tech note we will be using the ADuC834 as the target example.

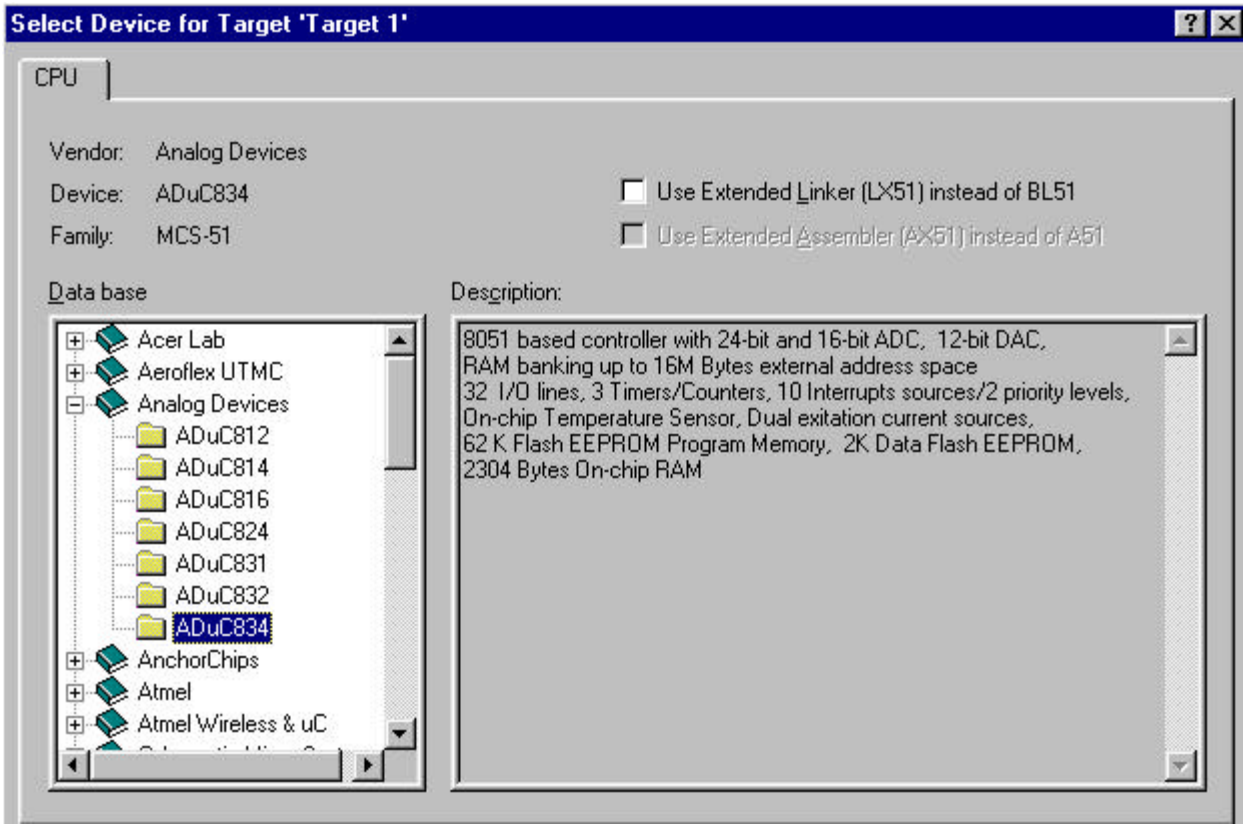

Figure 2.

Next, you need to configure your target output options. This is done by clicking on the “options for target” item located in the “project” pull-down menu. Alternatively you can select this option by clicking on the icon from the ‘compile’ toolbar situated at the top of the screen and shown in figure 3 below.

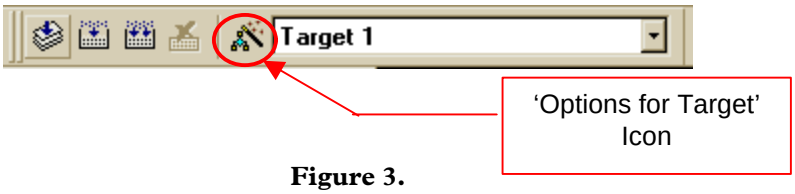

Figure 3.

A new window appears as shown in figure 4. Enable the option to Create Hex File by ticking the check-box and pressing ‘OK’ as shown circled in figure 4 below.

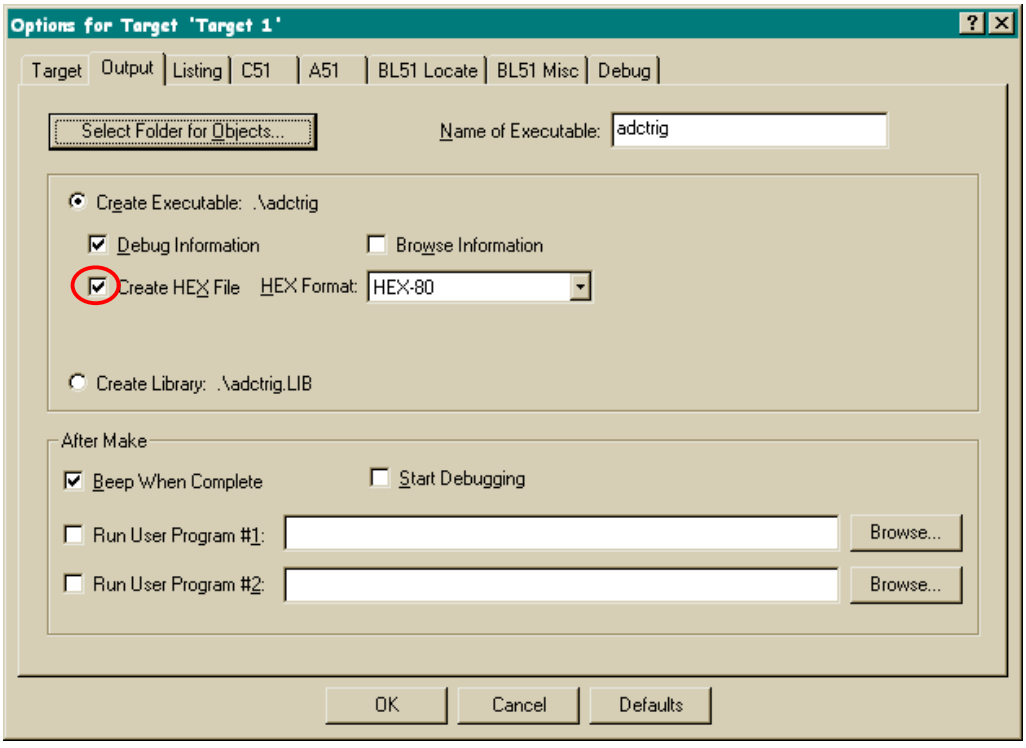

Figure 4.

CREATING A C PROGRAM

Now you can begin writing your C program. In the main window, choose the ***File*** pull-down menu and select ***New***. A new window named <text1> will appear on the screen. Type the C source code that is included in Appendix A of this tech note into the <text1> window.

Once you've typed all the code, again choose the ***File*** pull-down menu and select ***Save***. A new Save dialog window appears. Save your new file as Demo_ADuC.c in the Demo_ADuC folder you had created earlier.

At this stage, before compiling the C-program, we need to include it in our project. To do this you must click with the right mouse button on 'Source Group 1'and select Add Files to Group 'Source Group 1' as shown in figure 5. Select the Demo_ADuC.c file that is in the Demo_ADuC folder and click on Add and then on Close.

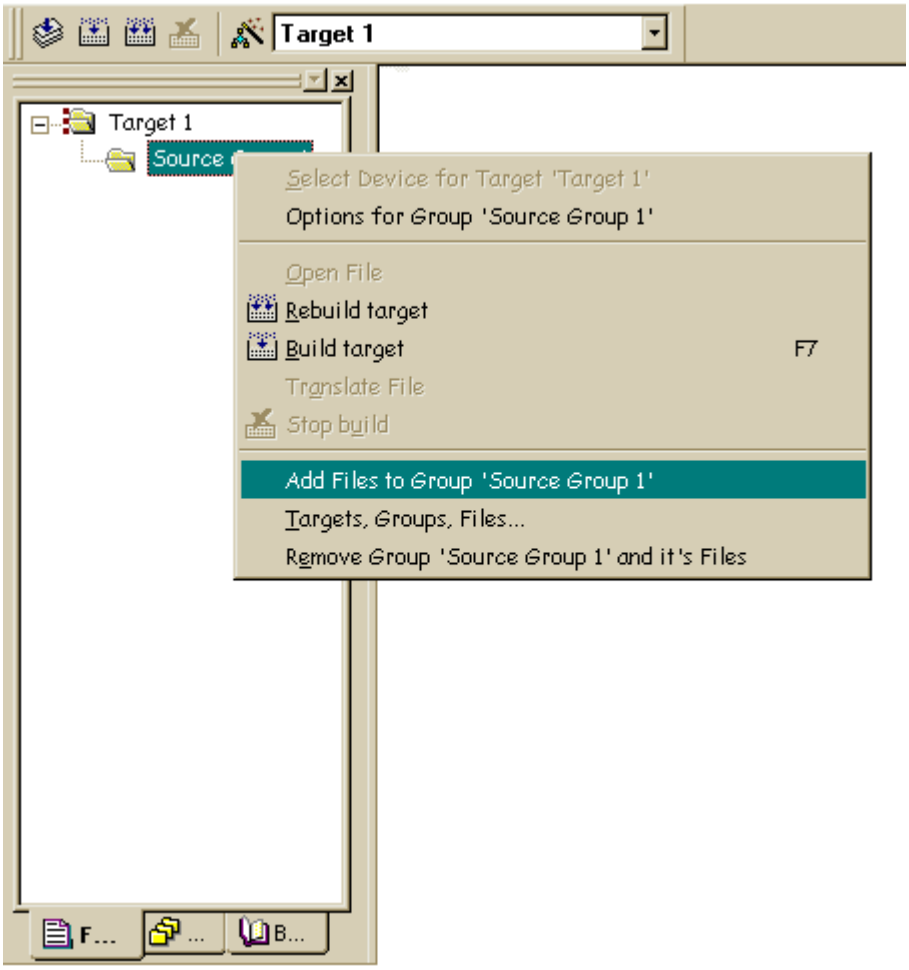

Figure 5.

COMPILING A C PROGRAM

Our C code is compiled by selecting 'Build Target' from the Project pull-down menu on the main tool-bar. The compile option can also be initiated by selecting the 'F7' special function key or by clicking on the 'Build Target' icon in the compile toolbar as shown in figure 6 below.

Figure 6.

If the compilation completes successfully a message indicating that the compile job has completed with 0 errors will appear in the build dialog screen. Sometimes warnings may be generated for information purposes to indicate multiple function calls etc. At this time, all the files that you need have been created in the Demo_ADuC folder. Chief among these files are :

Demo_ADuC.hex : The Intel-Standard, hex-file used when downloading code to the part in-circuit via the serial port

Demo_ADuC : This output file, generated without a suffix is used in C-Source hardware debug sessions.

If the resultant compilation message indicates that there were 1 or more errors, then the output files will not be created. In this case, the file has been entered incorrectly and clicking directly on any error message forces uVision2 to highlight where in the code the error has occurred.

SIMULATING YOUR C-SOURCE CODE

Another powerful feature of the uVision2 IDE is that it allows you to run your code in a MicroConverter specific simulation environment. To start a simulation session you simply click on the on 'Start/Stop Debug Session' option available from the 'Debug' pull-down menu. Alternatively you can press <Cntrl+F7> or the 'Debug' icon available in the 'File' toolbar as shown in figure 7.

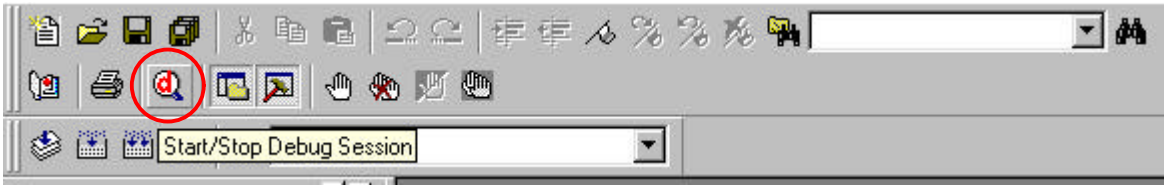

Figure 7.

CREATING THE SIMULATION ENVIRONMENT

The following steps will allow you to create a generic simulation environment that you may want to further customize to your own requirements. From the top Debug toolbar, click on the icon buttons that are shown in figure 8 below which open a Serial I/O peripheral window as well as a code Performance Analyzer window. Note: If the 'Debug' toolbar does not appear at the top of the screen, simply select the 'Debug Toolbar' option available from the 'View' pull-down menu.

Figure 8.

The various MicroConverter peripherals supported in the uVision2 simulation environment can be selected from the 'Peripherals' pull-down menu. The Port3 peripheral is selected by clicking on Port3 from the 'Peripherals' -> 'I/O Ports' option. In this example we will be using Port 3 to simulate external interrupts.

Re-arrange the active windows as shown in figure 9 below.

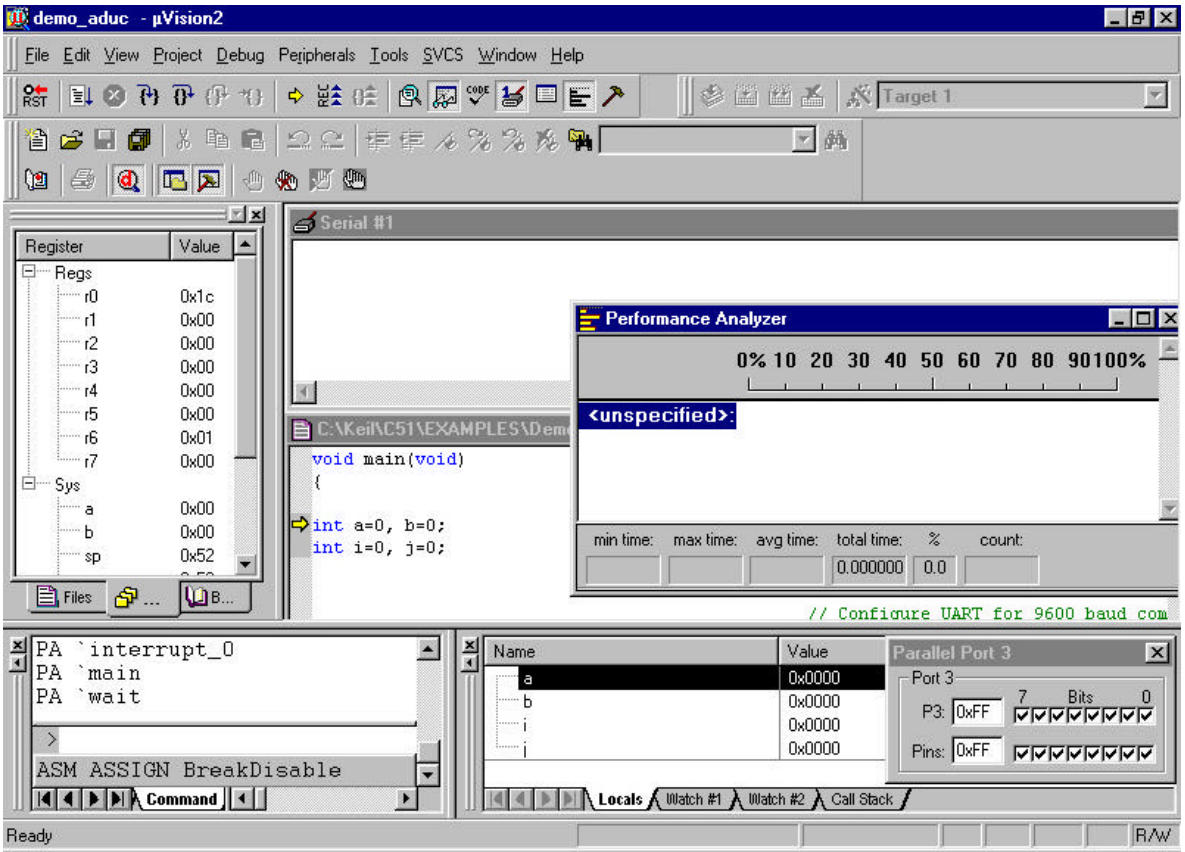

Figure 9.

CONFIGURING THE PERFORMANCE ANALYZER

Before we go any further we first need to configure the Performance Analyzer (PA). This is done simply by right clicking on the PA and selecting the 'Setup PA' option. In the resultant dialog box, double click on each function symbol and press 'Define' in turn. Once this operation is completed for each function symbol, press 'Close' and the PA window should now be configured as shown in figure 10.

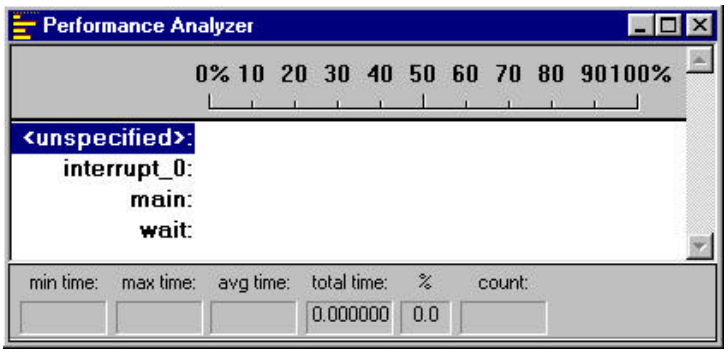

Figure 10.

BASIC CODE FLOW

The Demo_ADuC.C code continuously writes a linear ramp of 16 bytes to an internal RAM space on the MicroConverter. Each time a byte is written to internal RAM, P3.4 is toggled (P3.4 drives an external led on the MicroConverter Applications Board). If an external interrupt INT0 is simulated by toggling the P3.2 bit in the Port3 window, the code vectors to the interrupt_0 function ; the 16 values in the RAM are transmitted via the UART and the will not return to the main routine until P3.2 has been manually cleared. After each interrupt the delay between P3.2 toggles is increased because of the increment in the 'loop' variable.

STARTING THE SIMULATION SESSION

To start the simulation, click on the GO icon in the DEBUG toolbar as shown in figure 11.

Figure 11.

Immediately you will see the serial I/O window being updated with the data being transmitted by the MicroConverter UART in this simulation session and you should also note that the performance analyzer is being updated to reflect where code execution is spending most of its time (in this case, in the main routine) as shown in figure 12.

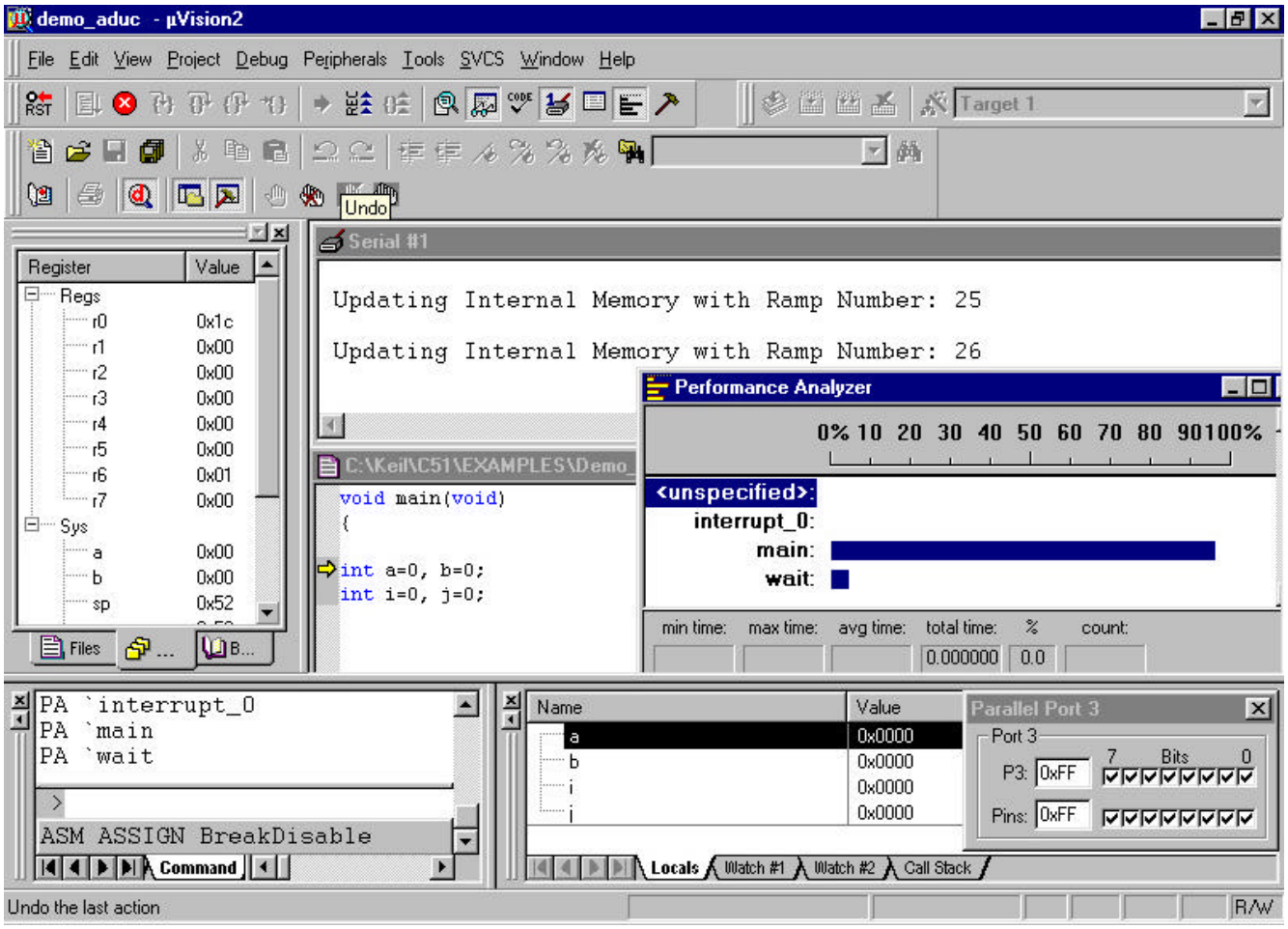

Figure 12.

SIMULATING AN EXTERNAL INTERRUPT

Next, an external interupt0 can be simulated by clearing the P3.2 bit in the Port window as shown in figure 13.

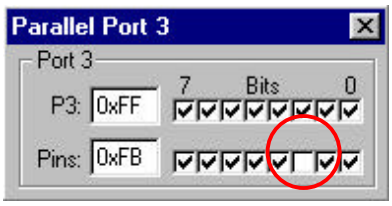

Figure 13.

In response to the simulated interrupt, the 16 values in the RAM are transmitted via the UART and code execution will not return from the Interrupt Service Routine (ISR) until you reset the P3.2. Again before you clear the P3.2 bit you should note that the performance analyzer changes to reflect how the code is now waiting in the INT0 ISR as shown in figure 14.

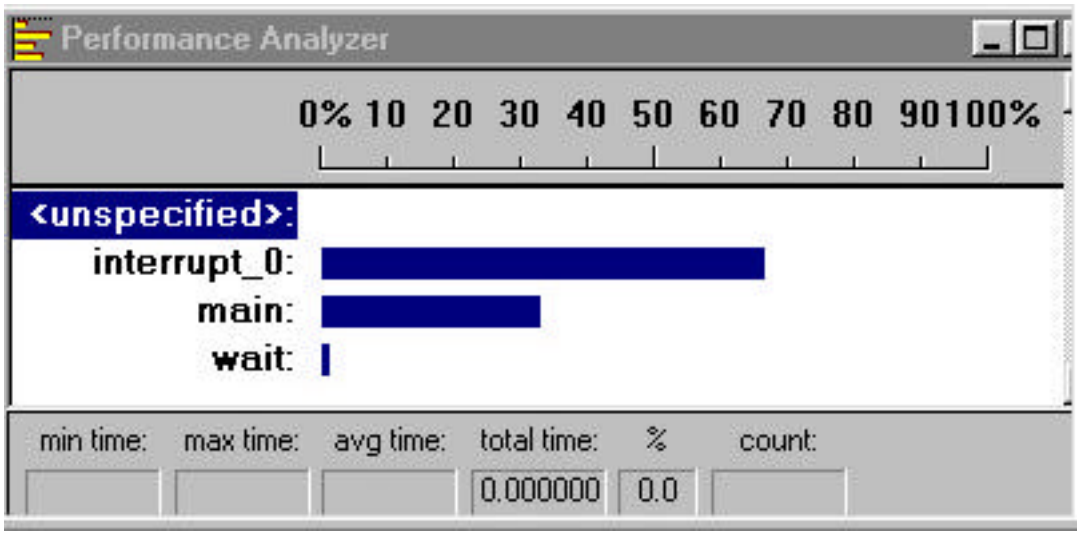

Figure 14.

CONCLUSION

This application note is a starting point in the use of the uVision2 IDE from Keil. uVision2 is a powerful means to develop your MicroConverter application in assembly or high level C. There are many aspects in the uvision2 tool suite; this application note illustrated some of them. The code compiled and simulated in this tech-note can be downloaded directly on the ADuC834 QuickStart™ evaluation board using the Windows Serial Downloader (WSD) application.

APPENDIX A : DEMO_ADuC CODE LISTING

```
//DEMO_ADuC.c -               This Program writes a linear ramp of 16 bytes to
//                            internal RAM on the MicroConverter. Each time a
//                            byte is written the P3.4 I/O pin is toggled.
//                            P3.4 drives an external LED on all MicroConverter
//                            Application Boards. In response to an external
//                            interrupt (INT0), the program transmits the internal
//                            RAM data bytes via the UART serial port at 9600 Baud
//
//                            Note this routine can be tailored to any
//                            MicroConverter device by simply changing the header
//                            file include directive at the beginning of the
//                            routine below.
//
//=====================================================================

#include <stdio.h>                          // Standard I/O Functions
#include <ADuC834.h>                        // ADuC834 Header File

int loop=0, c=0;                            // global loop and index variables
unsigned char memory[16];                   // memory array of chars

void interrupt_0 () interrupt 0             // INT0 ISR defined here
{
            for ( c = 0 ; c < 16; c++)      // Cycle through internal RAM
                  {
                        printf ("%02BD\n",memory[c]); // Transmit RAM via UART
                  }
            printf ("\n");                  // Transmit <cr>
            loop++;                         // Increment loop variable so the LED
                                            // will flash at a slower rate
            while(INT0 != 1);               // Wait until P3.2 pin is reset
}


void wait (void)  {                         // wait function
;                                           // only to delay for LED flashes
}


void main(void)
{

int a=0, b=0;                               // loop variables
int i=0, j=0;                               // memory index variables


                                            // Configure UART for 9600 baud comms
```

```
SCON = 0x52;                              // 8-Bit UART Mode
T2CON = 0x30;                             // enable T2 in reload mode for TX and
                                          // RX UART Clk
RCAP2H = 0xFF;                            // T2 hi-byte reload value
RCAP2L = 0xFB;                            // T2 lo-byte reload value
TH2 = 0xFF;                               // T2 hi-byte initial value
TL2 = 0xFB;                               // T2 lo-byte initial value
TR2 = 1;                                  // Run Timer 2

                                          // Configure INT0
TCON |= 0x01 ;                            // enable INTO interrupt as edge
                                          // triggered input
IE |= 0x80 ;                              // enable global interrupts, Set EA Bit
IE |= 0x01 ;                              // enable INTO, Set EX0 bit

      while(1)                            // Loop Forever
      {
            printf (" Updating Internal Memory with Ramp Number: %d\n", j);
                                          // Transmit Ramp indication update via
                                          // UART Serial Port

            for ( i = 0 ; i < 16; i++)    // Linear Ramp
                  {
                  for (a = 0; a < 1000; a++) // Delay for 1000 Counts
                        {
                        for (b = 0; b <= loop; b++)   // 2nd delay loop
                              {                       // delay increases everytime
                                                      // INTO ISR is executed
                                    wait();           // software delay
                              }
                        }
                        P3 ^= 0x10;       // toggle P3.4 which drives LED on Eval
                                          // Board
                        memory[i]=i+16;   // write internal RAM
                  }
            printf ("\n" ) ;              // transmit <CR> via UART
            j++;                          // increment ramp count
      }
 }
```