

Keil Toolset Workshop

Getting a leg up on ARM development

Kevin Bernhardt
Product Specialist



Today's Agenda

- Introduction and Overview
 - Keil and tools
- RealView Microcontroller Development Kit
 - Ease of use
 - Integrated Development μ Vision Environment
- Architecture overview
- Peripherals
- Interrupts and register banks



Keil: Tools for ARM-Powered Devices

ARM C/C++ Compiler

μ Vision
Project Manager, Editor & Debugger

RTX Real-Time Operating System

CAN Interface

File System

USB Host

USB Device

TCP/IP Networking Suite

GUI Library

Four Steps

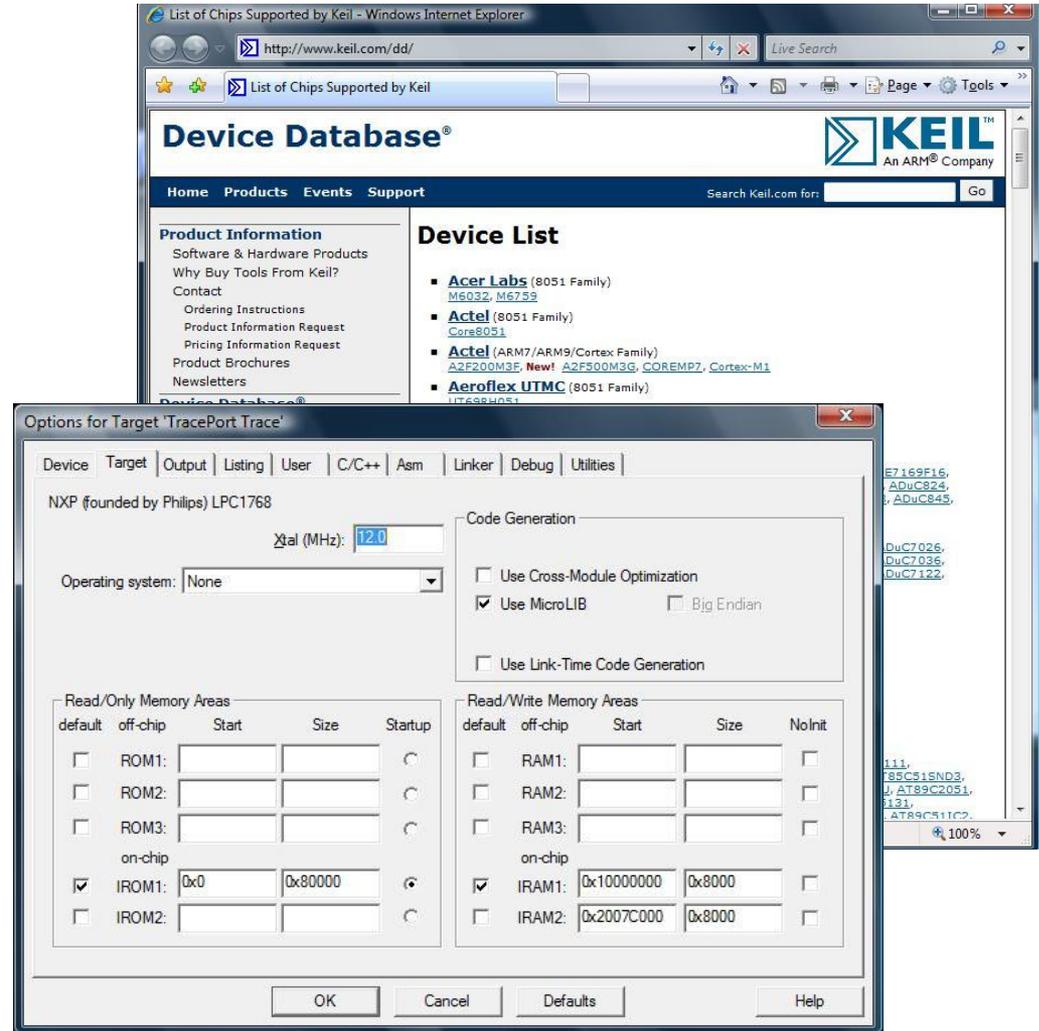
Move forward with the easy-to-use tools



Step 1: Create a Program

Select the device and specify the target hardware

- Find the perfect part with the website tools
- Jump-start your project with online device documentation



Step 2: Program Setup

Configure the device and create application code

➤ Use Keil's pre-made startup files and chip specific examples

The screenshot displays the Keil uVision IDE. On the left, a text editor window shows assembly code for stack and heap configuration. On the right, the Configuration Wizard dialog is open, showing various system options.

```
058 Stack_Top      EQU      Stack_Mem + Stack_Size
059
060
061 /// <h> Heap Configuration
062 /// <o> Heap Size (in Bytes) <0x0-0xFFFFFFFF>
063 /// </h>
064
065 Heap_Size      EQU      0x00000000
066
067
068 Heap
069
070
071 ///
072 EBI_
073
074 ///
075 ///
```

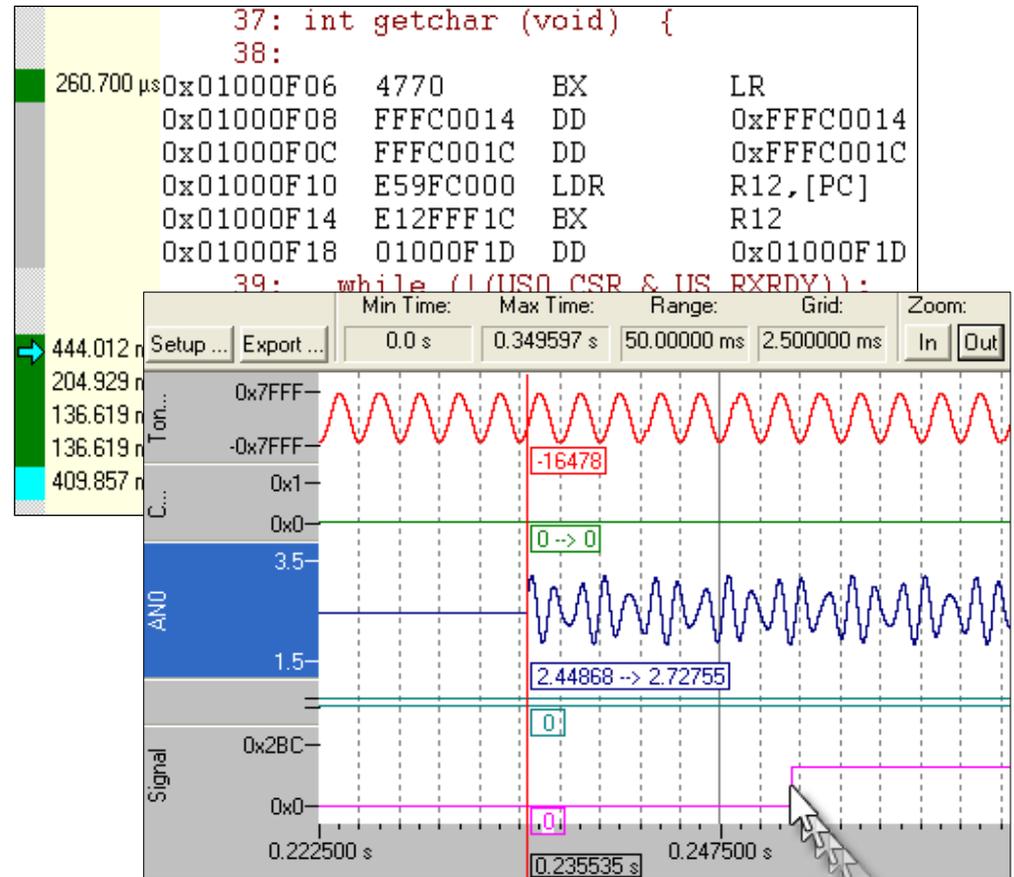
The Configuration Wizard dialog has the following options:

Option	Value
<input type="checkbox"/> Stack Configuration (Stack Sizes in Bytes)	
... Undefined Mode	0x0000 0000
... Supervisor Mode	0x0000 0000
... Abort Mode	0x0000 0000
... Fast Interrupt Mode	0x0000 0000
... Interrupt Mode	0x0000 0040
... User/System Mode	0x0000 0200
<input type="checkbox"/> Heap Configuration	
... Heap Size (in Bytes)	0x0000 0000
<input checked="" type="checkbox"/> External Bus Interface (EBI)	
<input type="checkbox"/> AT91M42800A PMC Clock Setup	
<input checked="" type="checkbox"/> AT91M55800A APMC Clock Setup	

Step 3: Program Testing

Analyze the program with μ Vision device simulation

▶ Take advantage of the analysis tools in the simulator



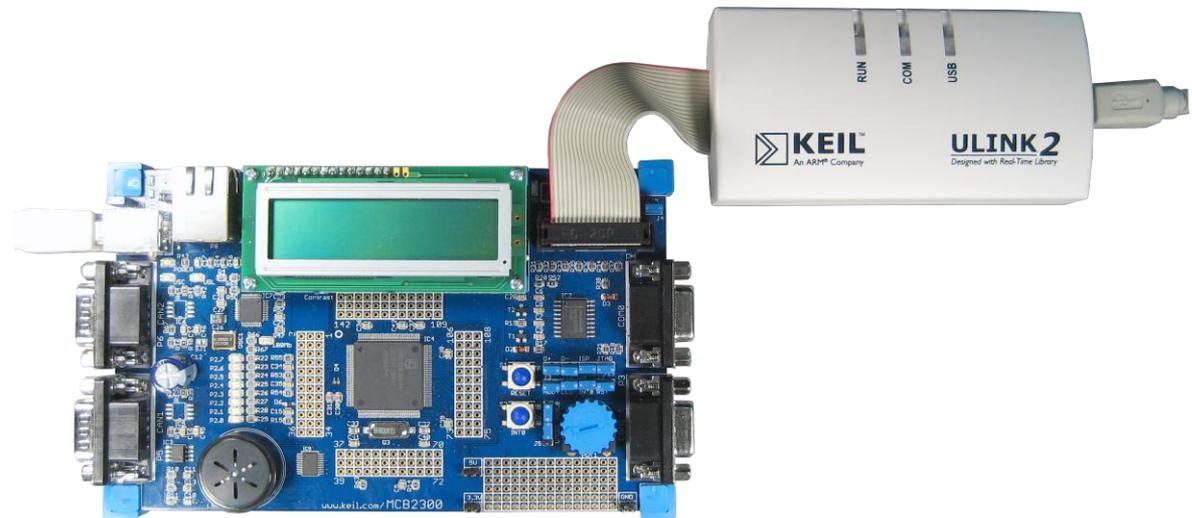
Step 4: Target Debugging

Flash download and final testing in target hardware

- ▶ Test the hardware, not the software
- ▶ More cost effective than an emulator

Target Debug

ULINK2 USB-JTAG Adapter
with Real-Time Agent



μ Vision4 Editor

The Keil Integrated Development Environment



µVision - Build and Edit Mode

The screenshot shows the Keil µVision4 IDE interface. The main window is titled "Blinky - µVision4" and contains several panes:

- Menu Bar:** Located at the top, containing File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help.
- Toolbars:** Located below the menu bar, containing various icons for file operations, simulation, and debugging.
- Target Selector Window:** Located on the left side, showing a tree view of the project structure. The "Simulator" target is selected.
- Project Workspace:** Located on the left side, showing a tree view of the project files, including System Files, Source Files, and Trace Configuration.
- Source Window:** Located in the center, showing the source code for "Blinky.c". The code includes a "MAIN function" and a "main" function.
- Configuration Window:** Located on the right side, showing a table of configuration options. The table has two columns: "Option" and "Value".
- Page Tabs:** Located at the bottom of the source window, showing tabs for "Text Editor" and "Configuration Wizard".
- Output Window:** Located at the bottom, showing the build output, including the linking process and program size information.
- Status Bar:** Located at the bottom, showing "ULINK Pro Cortex Debugger" and "CAP1".

Option	Value
Stack Configuration	
Stack Size (in Bytes)	0x0000
Heap Configuration	
Heap Size (in Bytes)	0x0000

```
108 }
109
110 /*-----
111  MAIN function
112  *-----
113 int main (void) {
114     short AD_value, AD_print;
115
116     SystemInit();
117     SysTick_Config(SystemFrequency/100);
118
119     LCD_init();
120     LED_init();
121     SER_init();
122     ADC_init();
123
124     while (1) {
125         if (AD_value != AD_last) {
126             AD_value = AD_last;
127             AD_dbg = AD_value;
128         }
129         AD_print = AD_value;
130
131         if (clock_1s) {
132             clock_1s = 0;
133         }
134     }
135 }
```

Build Output

```
linking...
Program Size: Code=2916 RO-data=268 RW-data=36 ZI-data=516
".\Obj\Blinky.axf" - 0 Error(s), 0 Warning(s).
```

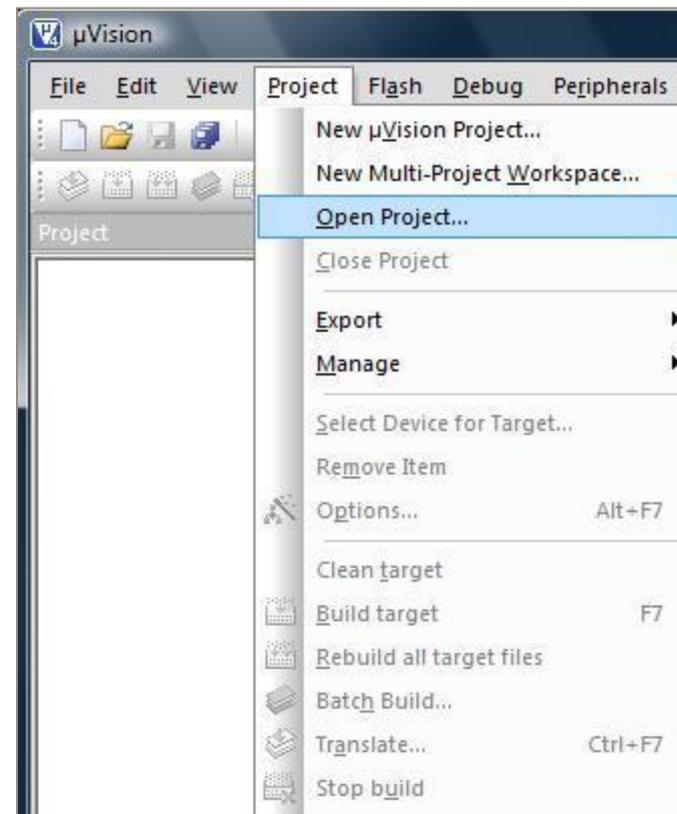
Example 1 - Open and Build a Project

➤ Go to Project → 'Open Project' and Open:

`..\ARM\Examples\Blinky.Uv2`

➤ Example Objectives:

- Build the project
- Hover the mouse pointer over an icon to display the tip box
- Debug the project with the simulator



µVision Build Toolbar



► Under the project menu:

-  **Translate:** Compiles the selected file in the Project Workspace window
-  **Build target:** Compiles, links any file changed since the last build
-  **Rebuild all target files:** Compiles, links all files in the project
-  **Stop Build:** Stops the build that is in progress
-  **Options For Target:** Opens target level configuration window

Build the Project

➤ Select the Simulator Target

 Click the **Rebuild all** button on the toolbar

 Note the information in the Output Window

```
Build Output
Build target 'LPC1768 Flash '
assembling startup_LPC17xx.s...
compiling system_LPC17xx.c...
compiling core_cm3.c...
compiling Blinky.c...
compiling IRQ.c...
compiling Serial.c...
compiling Retarget.c...
linking...
Program Size: Code=2856 R0-data=268 RW-data=28 ZI-data=516
".\Flash\Blinky.axf" - 0 Error(s), 0 Warning(s).
```

Switch to Debug Mode

- Hover the mouse pointer over the Debug button 
 - This will display the tip box



- Switch μ Vision into Debug mode
 - Go to **Debug – Start / Stop Debug Session**
 - The **Debug** Button will be selected in debug mode

µVision Debug Mode

The screenshot shows the Keil µVision IDE interface in Debug Mode. The main window displays the source code for 'BlinkySimple.c'. The 'Registers' window on the left shows the state of the processor registers. The 'System Tick Timer' dialog is open, showing control and status options. The 'Watch' window at the bottom shows the value of the 'NVIC_ACTIVE0' register. The 'Memory' window shows the contents of memory addresses. The 'Status Bar' at the bottom indicates the simulation is running.

Labels and their corresponding components in the image:

- Toolbars**: Located at the top left of the IDE window.
- Menu Bar**: Located at the top right of the IDE window.
- Source Window**: The main window displaying the source code.
- Project Workspace**: The left sidebar showing the project structure.
- Page Tabs**: Located below the Project Workspace, showing 'Project' and 'Registers' tabs.
- Output Window**: Located at the bottom left, showing the command prompt.
- Watch Window**: Located at the bottom center, showing the value of the 'NVIC_ACTIVE0' register.
- Memory Window**: Located at the bottom right, showing the contents of memory addresses.
- Status Bar**: Located at the very bottom, showing simulation status and timing information.
- Peripheral Dialog**: The 'System Tick Timer' dialog box on the right side.

µVision Debug Toolbar



► Under the debug menu

-  **Run:** (execute) until the next active breakpoint
-  **Step:** Execute a single-step into a function
-  **Step Over:** Execute a single-step over a function
-  **Step out of Current function:** Execute a single-step out of function
-  **Run to Cursor Line:** Execute until the current cursor line is reached
-  **Stop Debugging:** Halt the program

Debugging the Code

➤ To Run the Program

-  Press the **Run** button to start execution
-  Press the **Stop** button to halt execution
-  A Yellow arrow points to the next instruction

➤ To run to a Breakpoint

-  Double click on the line number column, line 28, to set a breakpoint
-  Press **Run** to start execution
-  Execution will halt on your breakpoint

Breakpoints Options



- Under the Debug menu
 - Insert Remove Breakpoints
 - Kill All Breakpoints
 - Enable/Disable Breakpoints
 - Disable All Breakpoints

Simulator

- Powerful tool for rapid program development
- ‘Virtual Test Registers’
 - Allow stimulus to be added to simulator device
 - Help Standardize Test cases
 - creating test data input to the user target software
 - Will be discussed in detail later
- Simulates up to 16Mbytes of memory mapped as read, write or code execution
- Supports integrated peripherals
 - Can be displayed or controlled from dialog boxes

Example 2 - Correct Errors in Code

➤ Open the project 'badcode.uv2' on your desktop

➤ Example Objectives:

- Build the project
- Use the editor, compiler to correct the errors

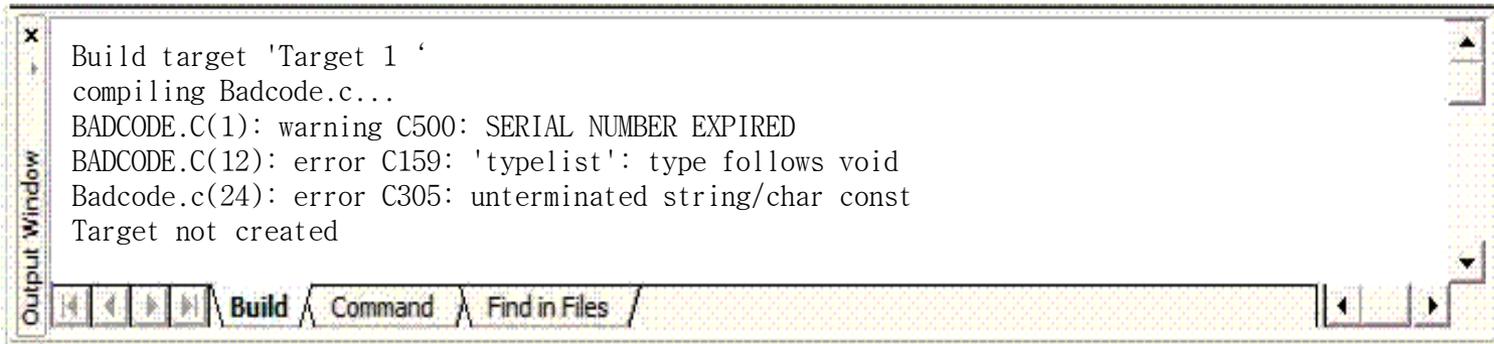
➤ Review:

- Color syntax
- Error messages
- Hot keys

```
/*-----  
BADCODE.C  
  
Copyright 1995-1999 Keil Software,  
  
This source file is full of errors.  
correct errors in this file.  
-----  
  
#include <stdio.h>  
  
void main (void, void)  
{  
  unsigned i;  
  long fellow;  
  fellow = 0;  
  
  fer (i = 0; i < 1000; i++)  
  {  
    printf ("I is %u\n", i);  
  
    fellow += i;  
    printf ("Fellow = %ld\n", fellow);  
    printf ("End of loop\n")  
  }  
}
```

Finding Compiler Errors

 Build the program, view the output window



The screenshot shows a window titled 'Output Window' with a close button (X) in the top-left corner. The window contains the following text:
Build target 'Target 1 '
compiling Badcode.c...
BADCODE.C(1): warning C500: SERIAL NUMBER EXPIRED
BADCODE.C(12): error C159: 'typelist': type follows void
Badcode.c(24): error C305: unterminated string/char const
Target not created
At the bottom of the window, there is a toolbar with several icons and three tabs: 'Build', 'Command', and 'Find in Files'. The 'Build' tab is currently selected.

 Double Click the first error in the build tab

 A cyan pointer indicates where the error was detected

Correct Errors in Code With Color Syntax

- Note color syntax for Keywords, Comments, Literals and Text
- Colors can be changed
- User keywords can be added

```
/*-----  
BADCODE.C  
  
Copyright 1995-1999 Keil Software,  
  
This source file is full of errors.  
correct errors in this file.  
-----  
  
#include <stdio.h>  
  
void main (void)  
{  
  unsigned i;  
  long fellow;  
  fellow = 0;  
  
  for (i = 0; i < 1000; i++)  
  {  
    printf ("I is %u\n", i);  
  
    fellow += i;  
    printf ("Fellow = %ld\n" , fellow);  
    printf ("End of loop\n");  
    while (1);  
  }  
}
```

μVision - Correcting Errors

- ▶ Correct the error—did the Keyword change color?
- ▶ Select the second error. In embedded systems usually ‘Main’ cannot accept args or return
 - In ARM tools, Main() must be type ‘void’

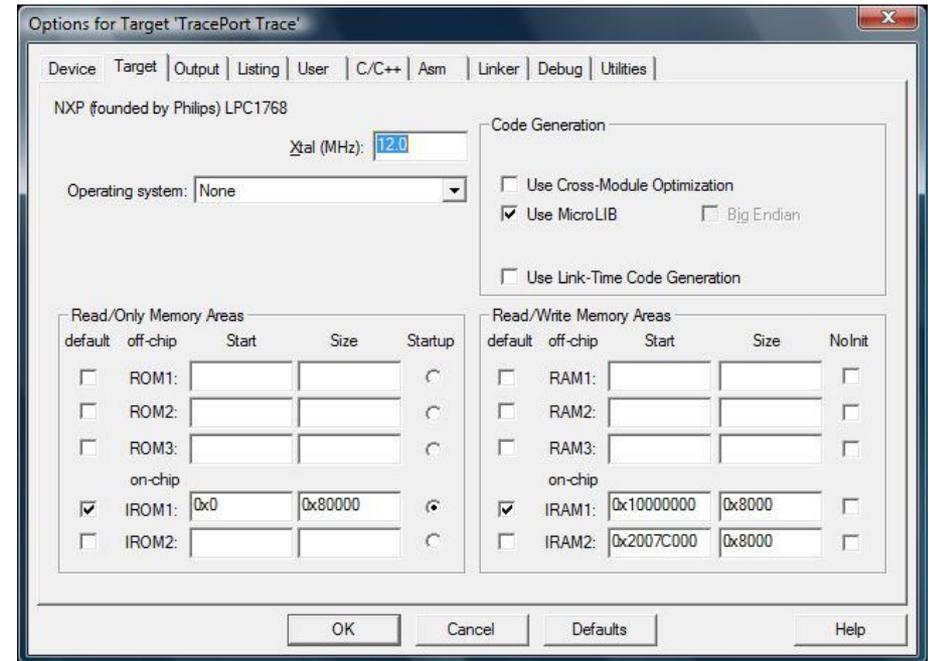
µVision - Finish Fixing the Errors

- ▶ Select the third error. Now hit the F1 key for context sensitive help
- ▶ Help may not be available for all errors. (Spelling Errors)
- ▶ Rebuild the target after each error fix to update the build tab

Memory - Target Tab for ARM

Select Options for Target

- Set off chip / on chip memory
- Set where the startup code will go
- Set the default memory section to place code
- Choose to not initialize a RW section, if needed



μVision Debug Options on the Toolbar



➤ Under the Debug menu

- **Current Statement:** Opens the file where the program counter is currently at
-  **Enable Trace Recording**
-  **View Trace Records**

Trace record

- Debug - View Trace Records displays a history of executed instructions.
- Debug - Enable/Disable Trace toggles the trace feature on/off. Trace must be enabled and code executed before a trace history is recorded.

More Debug Tools

Get to Know Your Code



Tools on the Debug Toolbar



► Under the View menu:

-  **Disassembly:** Shows a mix of source code and machine code
-  **Watch Windows:** shows locals, selected globals, and call stack
-  **Memory windows:** shows values at a memory location

Debug Tools

Disassembly window

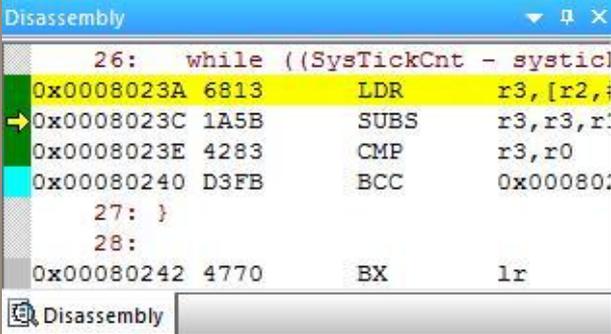
- Shows the target program as mixed source and assembly code

Watch window

- Multiple ways to add variables
 - Use F2 key
 - Hover over variable and right click
 - Select Add to Watch, type watch point command

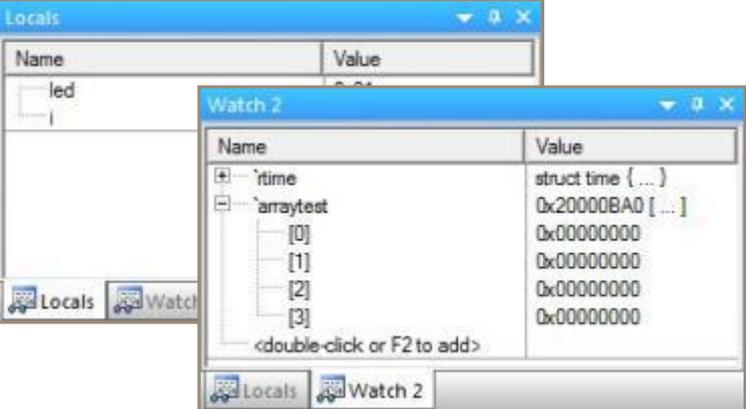
Memory window

- Displays the contents of memory from entered start address
- Updated periodically



Disassembly window showing source code and assembly instructions:

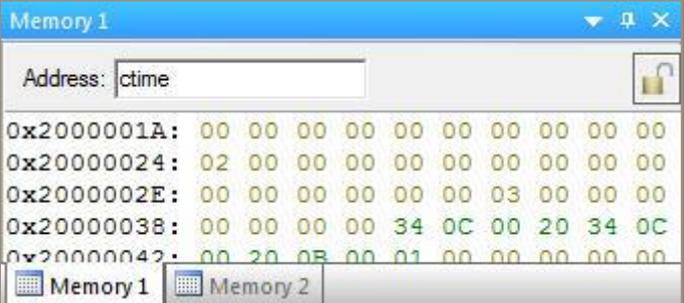
```
26: while ((SysTickCnt - systicl
0x0008023A 6813 LDR r3, [r2, #
0x0008023C 1A5B SUBS r3, r3, r
0x0008023E 4283 CMP r3, r0
0x00080240 D3FB BCC 0x00080
27: }
28:
0x00080242 4770 BX lr
```



Locals window showing variable 'led' and Watch 2 window showing variables 'time' and 'arraytest'.

Name	Value
led	0x00000000

Name	Value
time	struct time { ... }
arraytest	0x20000BA0 [...]
[0]	0x00000000
[1]	0x00000000
[2]	0x00000000
[3]	0x00000000



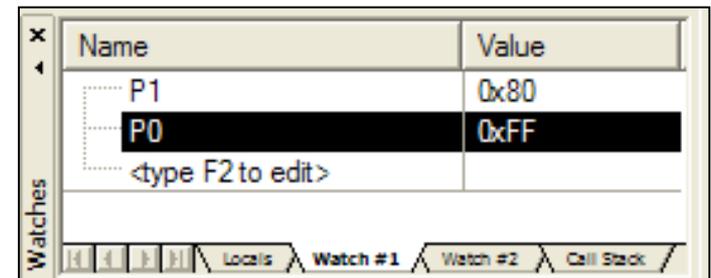
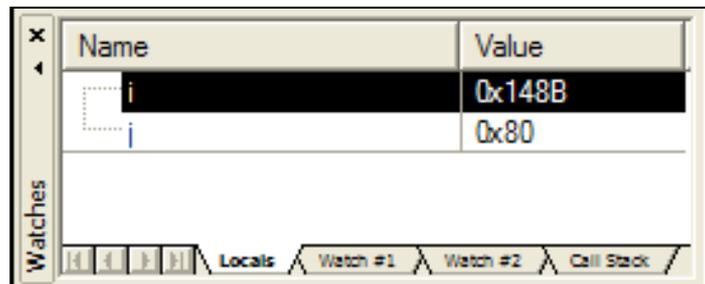
Memory 1 window showing memory contents:

Address	0x2000001A	0x20000024	0x2000002E	0x20000038	0x20000042
0x2000001A:	00 00 00 00 00 00 00 00 00 00	02 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 03 00 00	00 00 00 00 34 0C 00 20 34 0C	00 20 0B 00 01 00 00 00 00 00

Watch Window

Enter watch points in one of three ways:

- Click in the watch window at 'enter here' and hit F2 to enter or edit a variable name.
- In the edit window hover a variable and right click the mouse and select Add to Watch
- In the Command Window type the watch point command



Memory Window

- ▶ (Right Click) to select the output format
 - The default is no output format selected which displays HEX characters
 - Right click on an address and select the Modify command at the bottom of the pop up menu list

Memory Window

- ▶ Displays the contents of different memory areas
 - A memory area is defined in one of four display windows
 - Up to four memory areas can be displayed at one time
- ▶ **Starting Address** of the memory to display is entered in the **Address** entry box
 - EX: C:0x0080 will display a block of Code memory starting at 0x0080
- ▶ View → 'Periodic Window Update' to update target variables while target program is running

Serial Window



View - Serial Window -

- View simulated data sent from chip
- Type to transmit simulated data to chip



Right click for a serial window local menu

- Switch between ASCII /Hex Mode
- Clear the Serial Window



µVision Analysis Windows on the Toolbar



► Under the **View** > **'Analysis Windows'** menu

-  **Code Coverage:** shows the instructions in the program that have executed
-  **Logic Analyzer:** graphically displays signals and program variables as they change over time
-  **Performance analyzer:** records and displays execution times for functions and program blocks

µVision Code Coverage

CODE ✓ Execution Statistics

- Always active for complete project
- Instruction Status:
 - Not executed (grey)
 - Fully executed (green)
 - Skipped (orange)
 - Always taken (cyan)
- Multi-Session Coverage with Save / Restore

```
274      break; /* start m
275
276      case 'Q': /* Quit Co
277          printf ("\nQuit Measurement Recording\n");
278          startflag = 0;
279          break;
280
281      case 'C': /* Clear C
282          printf ("\nClear Measurement Records\n");
283          clear_records ();
284          break;
285
286      default:
287          printf (E
288          printf (m
289      break;
290  )
```

```
378:          SUB      R0, R0, #SVC_Stack_Size
379:
380: // Enter User Mode and set its Stack Pointer
380: 0x000000D0 E2400020 SUB      R0,R0,#SVC_Stack_Size(0x
381:          MSR      CPSR_c, #Mode_USR
381: 0x000000D4 E321F010 MSR      CPSR_c,#Mode_USR(0x0000
382:          MOV      SP, R0
383:
384: // Enter the C code
384: 0x000000D8 E1A0D000 MOV      R13,R0
385:          LDR      R0,=?C?INIT
385: 0x000000DC E59F0020 LDR      R0,[PC,#0x0020]
386:          TST      R0,#1 ; Bit-0 set:
386: 0x000000E0 E3100001 TST      R0,#PLL_SETUP(0x00000000
387:          LDREQ    LR,=exit?A ; ARM Mode
387: 0x000000E4 059FE01C LDREQ    R14,[PC,#0x001C]
388:          LDRNE   LR,=exit?T ; Thumb Mode
388: 0x000000E8 159FE01C LDRNE   R14,[PC,#0x001C]
389:          BX      R0
390:          ENDP
```

Modules/Functions	Execution percentage
Blinky	
SysTick_Handler	100% of 5 instructions
Delay	85% of 7 instructions, 1 condjump(s)
main	81% of 54 instructions, 3 condjump(s)
system_SAM3U	
SystemInit	62% of 108 instructions, 6 condjump(s)

µVision Logic Analyzer

Timing Analysis

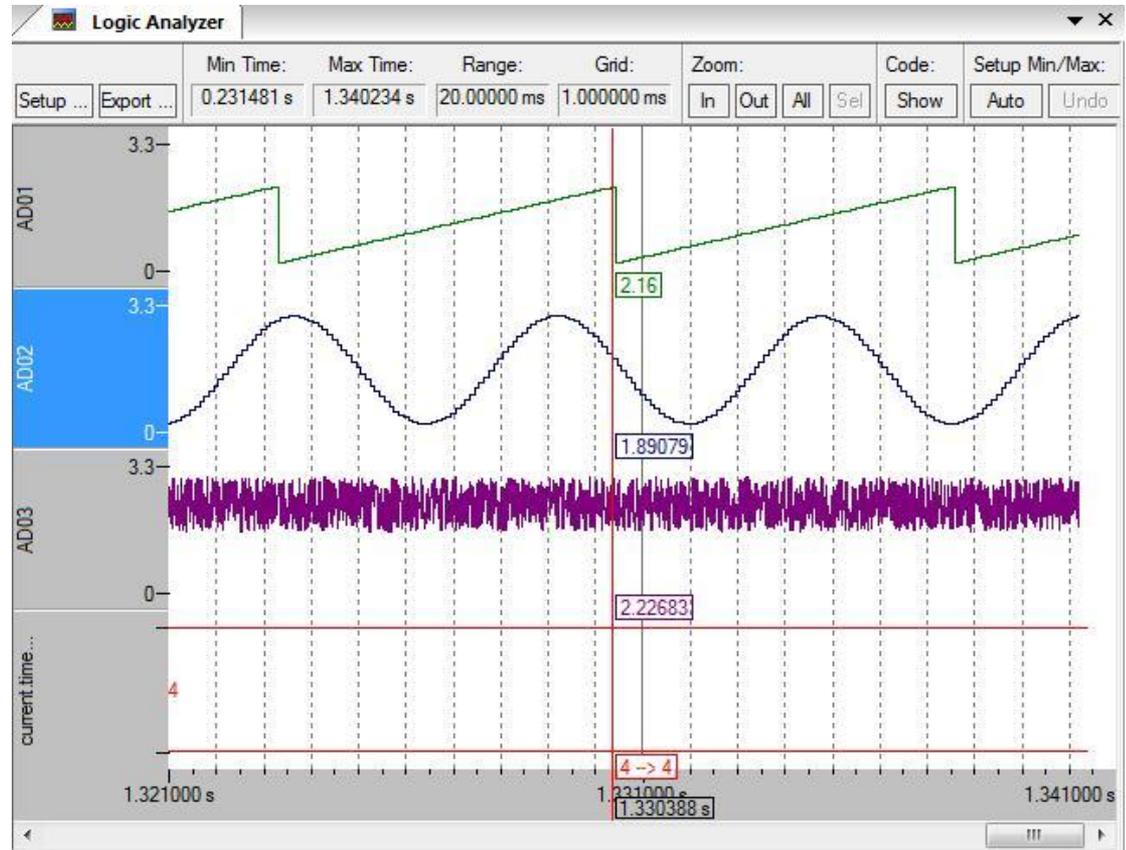
- Analog and Digital I/O Pins and Signals
- Internal Variables

Exact Timing

- Using Cursor Line
- Tool-Tip Delta Information

Display Style

- Analog
- Bit
- State Change



Logic Analyzer Setup

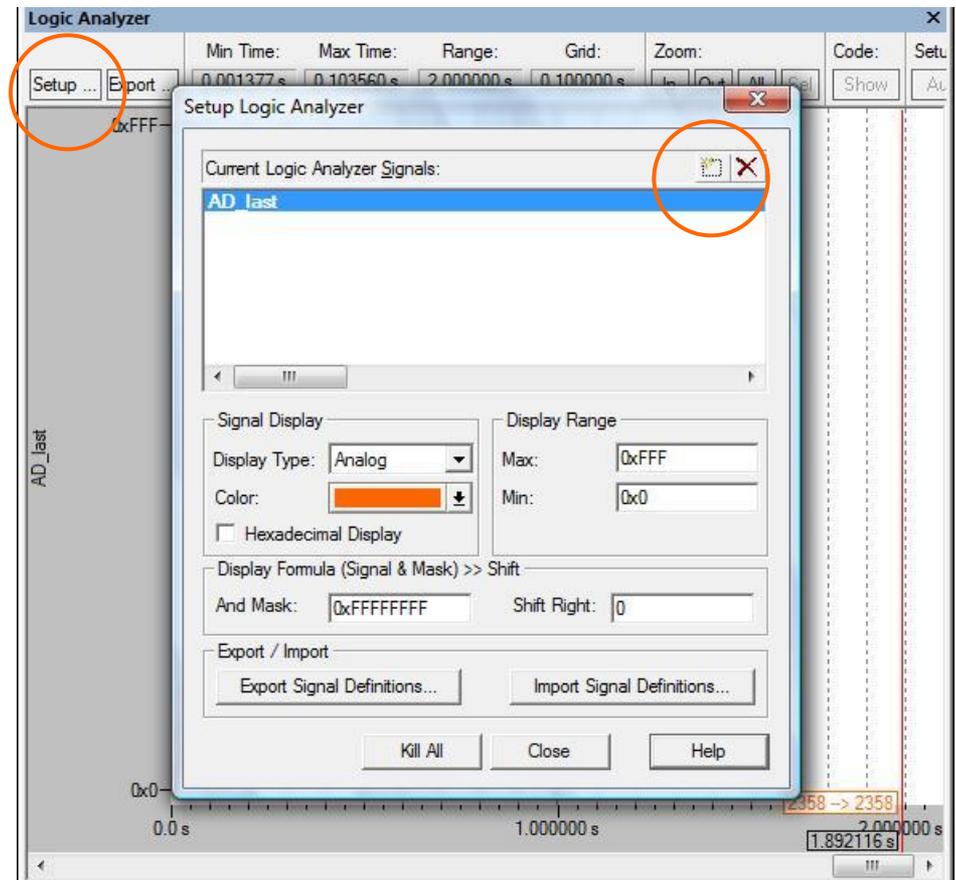


Setup is simple

- Click the 'Setup' Button
- Click the 'New' button
- Type a global name
- Modify the parameters
- Press 'Close'



...Or drag a variable from the Symbol window into the Logic Analyzer



Logic Analyzer

- Presently the LA is set up to monitor the pseudo-analog input to AD0 generated by the Signal function we just reviewed.
- The LA may or may not be displaying some of the step functions.
- Run the program for several seconds and then stop the program.
- With a left mouse click, select one of the signals and select ZOOM in for a good display. Note the Zoom centers the cursor and zooms around the cursor.

Logic Analyzer

- Move the cursor to the middle of one of the rising Horizontal steps.
- Press the right keyboard arrow key one time. The cursor will lock to a rising edge of the signal.
- The run time to the cursor position and the previous to next value will be presented in the lower position of the LA.
- Position the mouse pointer at the next signal rising edge. Note the display.
- You will see the old value, new value and delta values of the two pointers.

µVision Execution Profiling

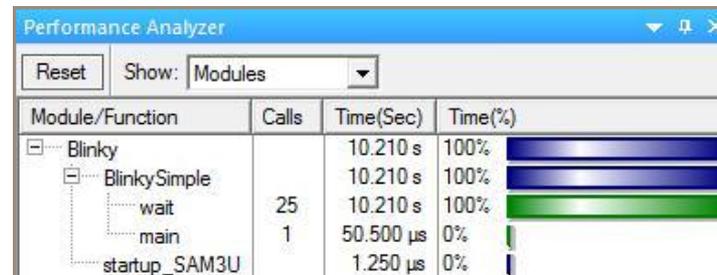
➤ Detailed Timing Statistics

➤ Shows:

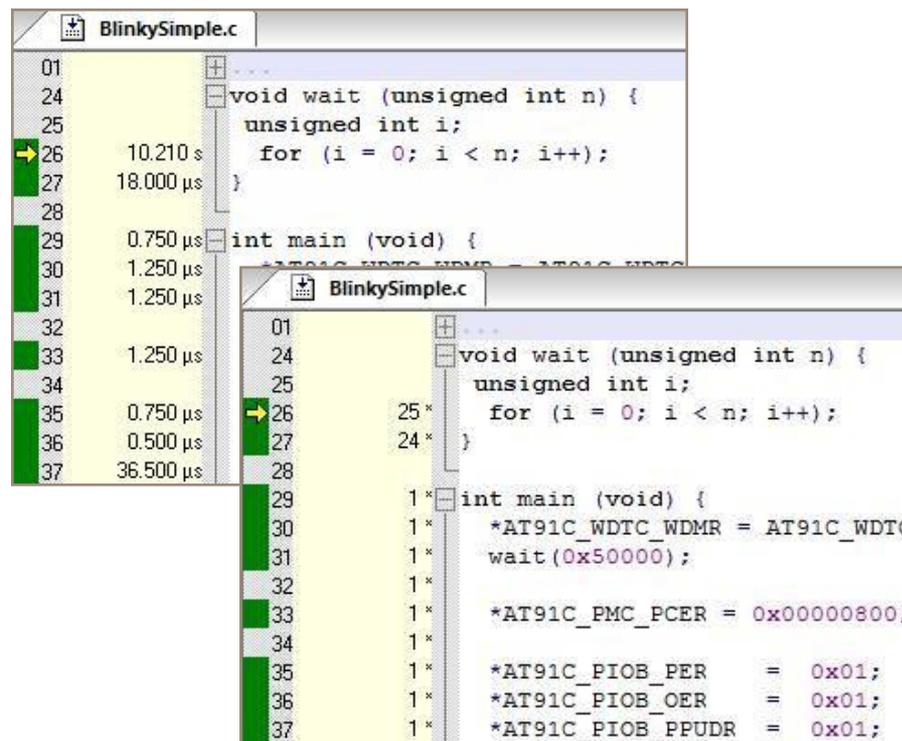
- Execution Time
- Number of Executions

➤ With Flexible Views:

- Source Windows
- Disassembler Window
- Complete Project Overview



Module/Function	Calls	Time(Sec)	Time(%)
Blinky		10.210 s	100%
BlinkySimple		10.210 s	100%
wait	25	10.210 s	100%
main	1	50.500 µs	0%
startup_SAM3U		1.250 µs	0%



```
01 ...
24 void wait (unsigned int n) {
25   unsigned int i;
26   for (i = 0; i < n; i++);
27 }
28
29 int main (void) {
30   *AT91C_WDTC_WDMR = AT91C_WDTC
31   wait(0x50000);
32
33   *AT91C_PMC_PCER = 0x00000800;
34
35   *AT91C_PIOB_PER   = 0x01;
36   *AT91C_PIOB_OER   = 0x01;
37   *AT91C_PIOB_PPUDR = 0x01;
```

µVision Debugger

- ▶ Keil Software's source level debugger and simulator.
 - Supports MCU peripherals via hardware driver DLL's.
 - Features single instruction step, single function step, breakpoint, trace and watch points.
 - Displays registers, code memory, data memory and stack space.
 - Simulates timers, interrupts, I/O ports and other on chip peripherals.

Options for Target - Debug Tab

- **Initialization File** Process the specified file as command input when starting a debug session
- **Breakpoints** Restore breakpoint settings from the previous debug session
- **Toolbox** Restore toolbox buttons from the previous debug session
- **Watchpoints & PA** Restore Watchpoints and Performance Analyzer settings from the previous debug session
- **Memory Display** Restore the memory display settings from the previous debug session

Options for Target - Debug Tab

- ▶ **CPU DLL and Driver DLL Parameter** Configures the internal μ Vision debug DLLs.
 - The settings are taken from the device database
 - Do not attempt to modify the DLL parameters
 - We will show how to build a new Device Data Base in a later session

Special Files

- ▶ Advanced configuration for your Project

StartUp Code - Purpose

➤ The Startup file:

- Clears the data memory
- Initializes hardware and stack pointers
- Sets up Memory

➤ Some devices require a CPU initialization code that needs to match the configuration of your hardware design

StartUp Code

- Several variants are furnished by device type
 - STARTUP.S - ARM startup code
- Starting a new project will usually initiate a dialog asking if you would like a copy placed in your project
- Always start with a fresh startup file placed in your project file

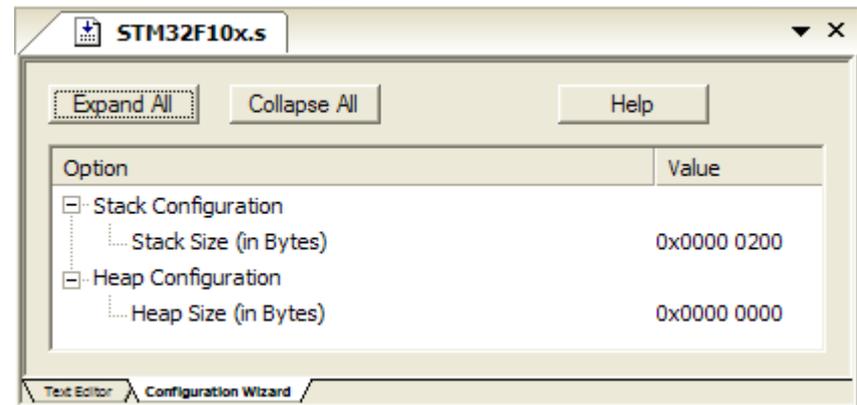
Configuration Wizard

- ▶ Works in Comments of code
- ▶ Create simple to use windows to quickly set up code, such as startup code
- ▶ See:

www.keil.com/support/docs/2735.htm

```
;/*****  
;/* STM32F10x.s: Startup file for STM32F10x device */  
;/*****  
;/* << Use Configuration Wizard in Context Menu >> */  
;/*****  
;/* This file is part of the uVision dev tools.  */  
;/* (c) 2005-2007 Keil Software All rights reserved*/  
  
;// <h> Stack Configuration  
;//  <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>  
;// </h>
```

Stack_Size EQU 0x00000200

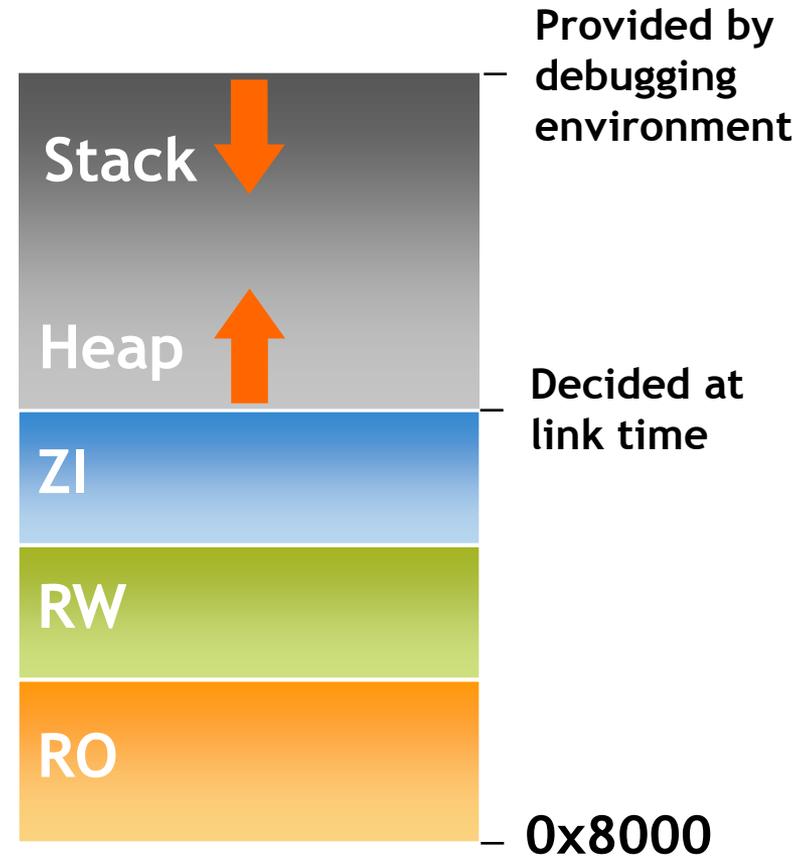


Scatter Loading File

- Absolute Control over Object Placement

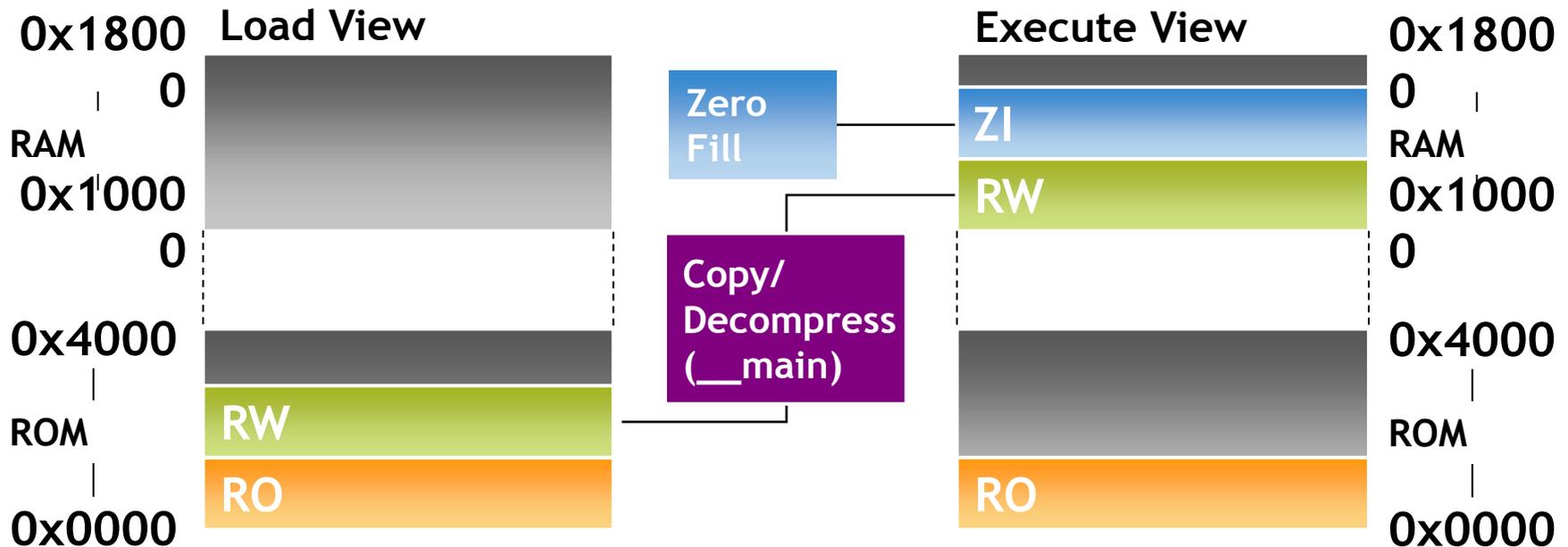
Default Memory Map

- Code is linked to load and execute at 0x8000 by default
- The heap is placed directly above the data region
- The stack base location is read from the debugging environment by C library startup code



Scatterloading - Simple Example

- RO code and data stays in ROM
- C library initialization code (in `__main`) will :
 - Copy/decompress RW data from ROM to RAM
 - Initialize the ZI data in RAM to zero



Keil Scatter file

➤ Used to absolutely place files

- Go to Project → Options for Target → Linker → Uncheck 'Use Memory Layout from Target Dialog'
- Below is the default Scatter file generated by the linker:

```
LR_IROM1 0x00000000 0x00008000 { ; load region size_region
ER_IROM1 0x00000000 0x00008000 { ; load address = execution address
.o (RESET, +First)
*(InRoot$$Sections)
.ANY (+RO)
}
RW_IRAM1 0x40000000 0x00002000 { ; RW data
.ANY (+RW +ZI)
}
}
```

Placing Memory Mapped Registers

- Define registers in a source file, for example timer_reg.c:

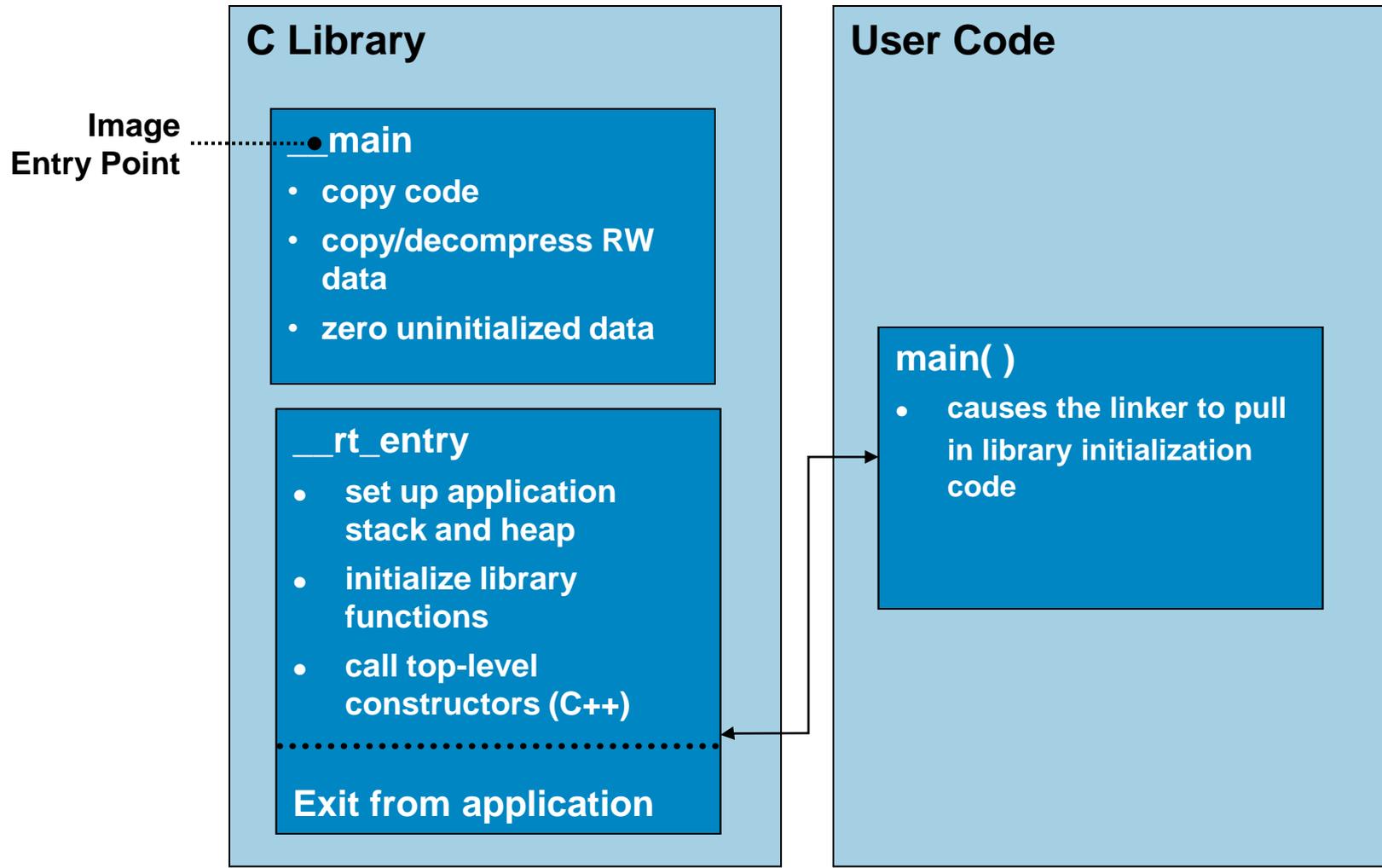
```
__attribute__((zero_init)) struct {  
    volatile unsigned reg1;      /* timer control */  
    volatile unsigned reg2;      /* timer value  */  
} timer_reg;
```

- Add an execution region to the scatter file to place registers at the required address:

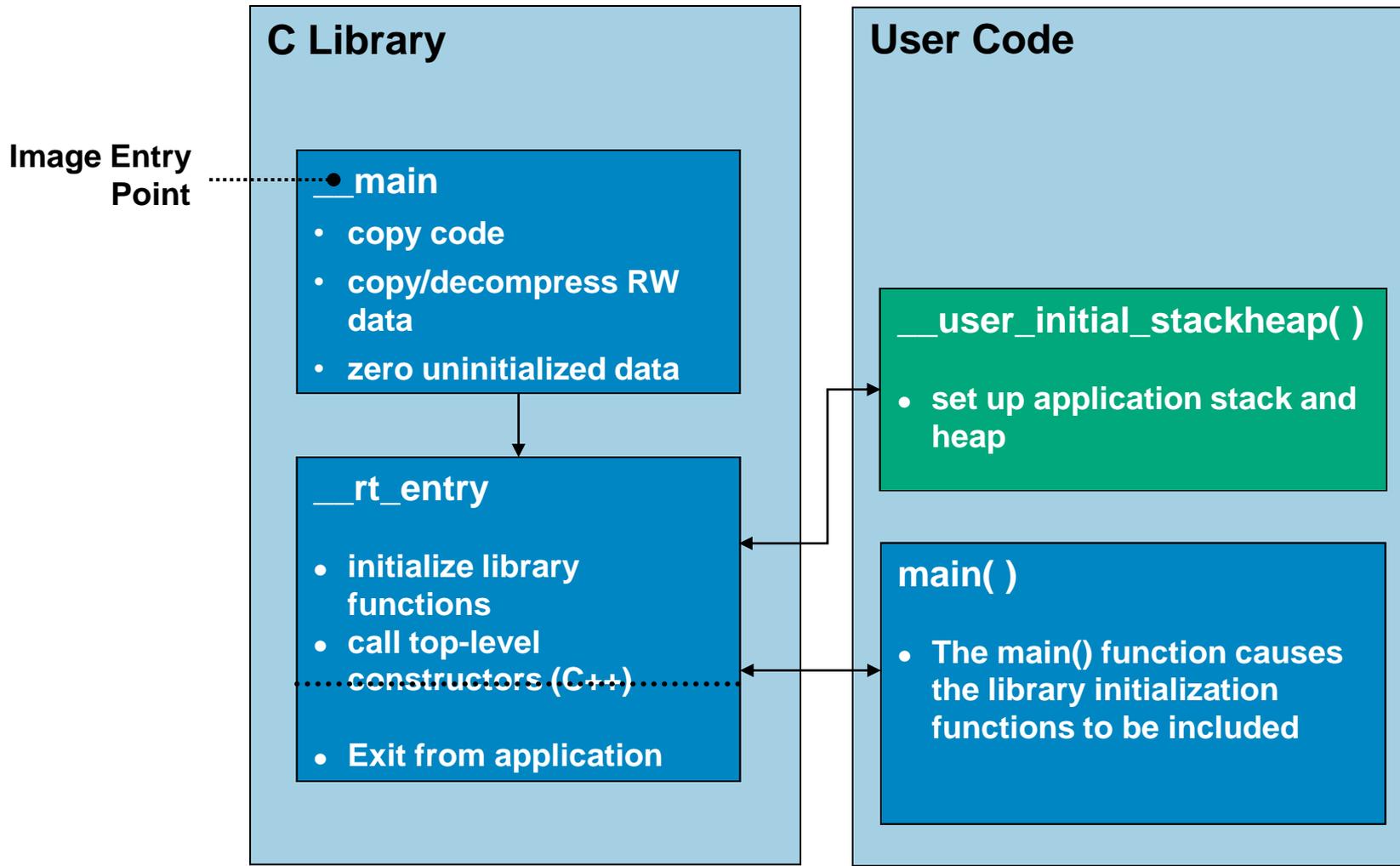
```
LOAD_FLASH 0x24000000 0x04000000 {  
    TIMER 0x40000000 UNINIT      {  
    timer_reg.o (+ZI)           {  
    }  
}
```

What does `__main` do?

Application Startup



Stack and Heap Initialization



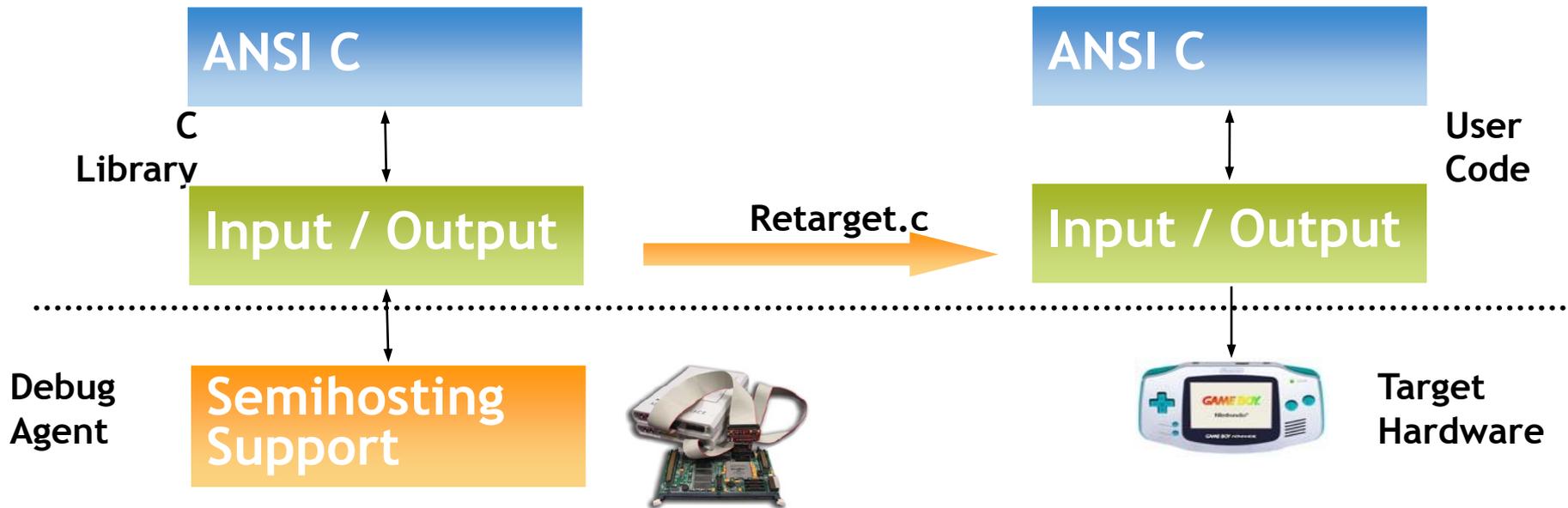
Retarget.c File

➤ Details behind what this file does

Retargeting the C Library

RealView is an out of the box debugger

- Debug features must be disabled before shipping
- For example: `printf()` should go to LCD screen, not debugger console



Keil Retarget.c file

- Required for all ARM projects
- Copy the default retarget.c file from:
..\ARM\Startup\Retarget.c

```
#pragma
import(__use_no_semihosting_swi)

extern int sendchar (int ch);

struct __FILE { int handle; };
FILE __stdout;

int fputc (int ch, FILE *f)      {
    return (sendchar(ch));
}

int ferror (FILE *f)            {
    return EOF;
}

void __ttywrch (int ch)
{
    sendchar (ch);
}

void _sys_exit (int return_code) {
    label: goto label;
}
```

SWI Tables / SVC Tables

Execution Mode Considerations

- It is important to consider which mode your main application will run in
 - User Mode is an unprivileged mode - protects your system

- System initialization can only be executed in privileged modes
 - Need to carry out privileged operations, e.g. enable interrupts

- If you want to run your application in a privileged mode, simply exit your reset handler in System Mode
 - Provides a stack not used by exception modes (User Mode Stack Pointer)

- If you want to run your application in user mode, you will need to change to user mode in `SubMain()`
 - However, `__user_initial_stackheap()` must have access to your application mode registers
 - Solution is to exit reset handler in system mode, so that
 - All C lib initialization code has access to user registers, but can still perform privileged operations

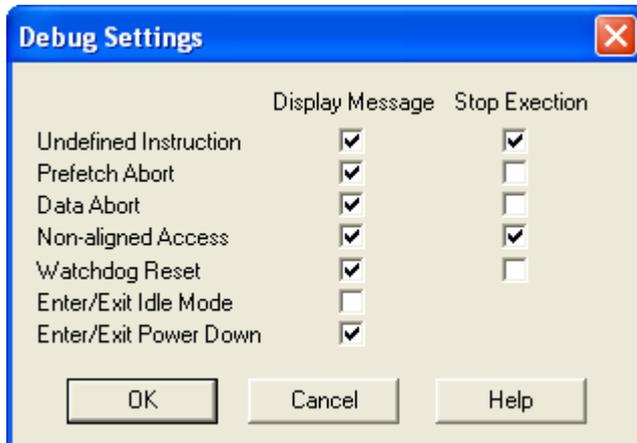
Map File

Unused Section Elimination/Entry Points

- By default, the linker will remove from the final image any code sections that are never executed, or data that is never referred to
 - To see if any sections have been removed, link with ‘--info unused’
 - If a section is not marked as +FIRST or +LAST, ‘--keep’ can be used to prevent required sections being removed
- Suggested link line for ROMmable images is:
`armlink obj1.o obj2.o --scatter scatter.scat
--info unused --entry startup -o prog.axf`

Detect Illegal Memory Accesses

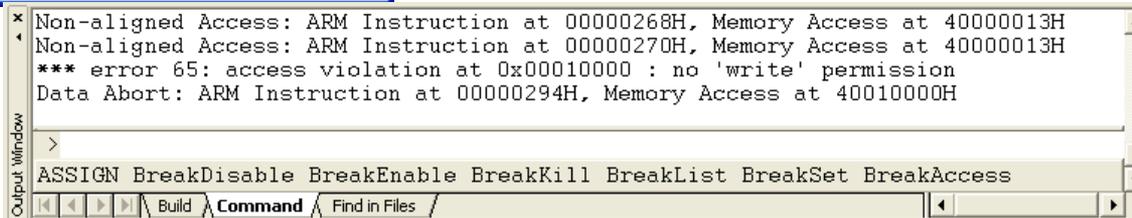
- Requirement: Detect illegal accesses that fail in Hardware
- MAP command allows to define access rights: read, write, execute
 - On illegal memory access you may display messages or stop simulation



```
// Un-aligned memory access create wrong values in HW
// but are hard to detect during JTAG Debugging
typedef struct {
    COLOR          Color;
    unsigned short Level;
    Buffer[10];
} LEVEL;

struct {
    INPUT_EVENT  Event;
    unsigned char
} Input;

unsigned short SetStruct (void) {
    LEVEL *pLevel;
    pLevel = (LEVEL *)&Input.Buffer[0]; // set pointer
    pLevel->Level = 0x1234;             // UN-ALIGN
    ACCESS!
```

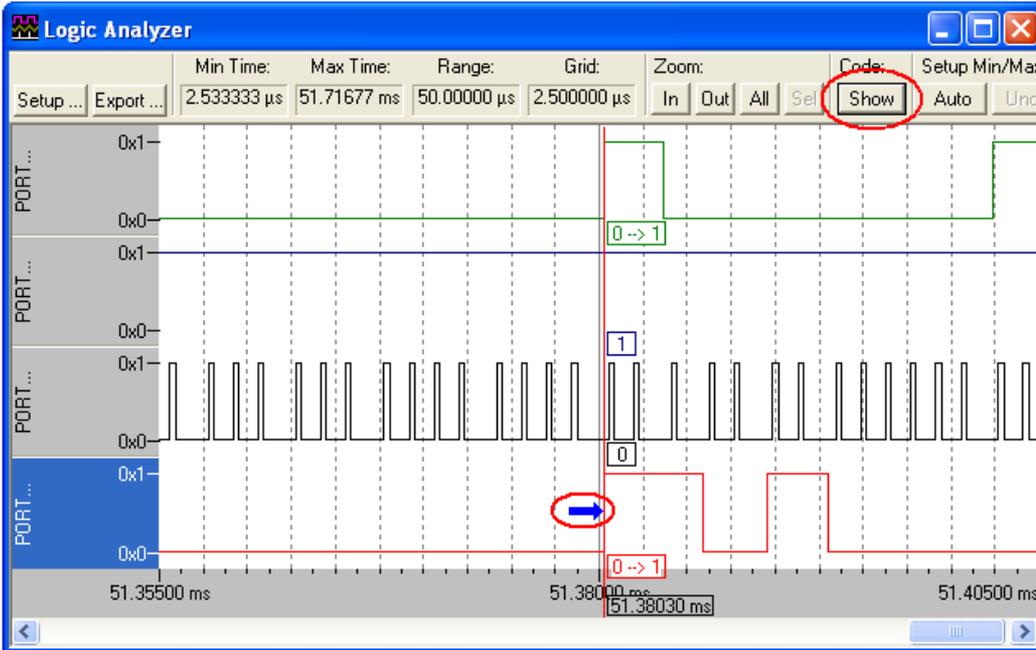


<http://www.keil.com/download/docs/323.asp>

Use Case: Detect I/O Glitches

▶ Requirement: Analyze timing and glitches of JTAG I/O Pins

- Using the Logic Analyzer the JTAG signal pin timing is verified.
- Glitches of I/O pins can be analyzed.
- Synchronisation with the Source Code simplifies corrections.



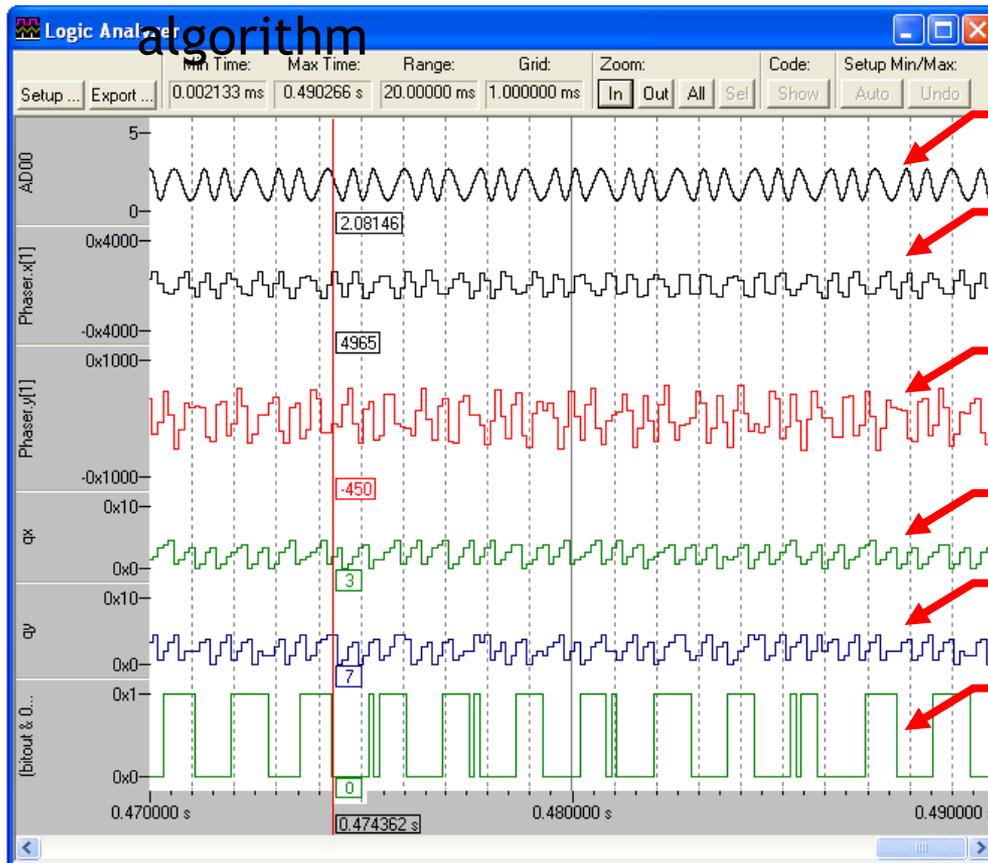
```
F:\VRKUV3\ULINK2\FIRMWARE2\JTAG_CMD.c
281 /*
282  * Set JTAG I/O lines: TRST, TMS, TDI,
283  */
284 static void alt_o (U32 v) {
285     U32 i;
286     U32 x;
287
288     x = PIN_TRST | PIN_TMS | PIN_TDI;
289     x &= ~v;
290     if (x) FIOCLR = x;
291     if (v) FIOSET = v;
292     for (i = 0; i < NoWS; i++);
293     FIOSET = PIN_TCLK;
294     for (i = 0; i < NoWS; i++);
295     FIOCLR = PIN_TCLK;
296 }
297
298
299 /*
300  * Reset TAP with TRST = 0 and go to Ru.
301  */
302 static void InitJTAG (void) {
```

<http://www.keil.com/download/docs/322.asp>

Use Case: Modem Receiver for CLID

➤ Requirement: Replace hardware with software algorithms

- CPU time is critical since CPU is needed also for other functions
- Check internal variables over time during the development of algorithm



Analog modem signal on CPU input

Digital samples after filtering

De-phase algorithm performs phase shifting

PLL quadrant of de-phaser input

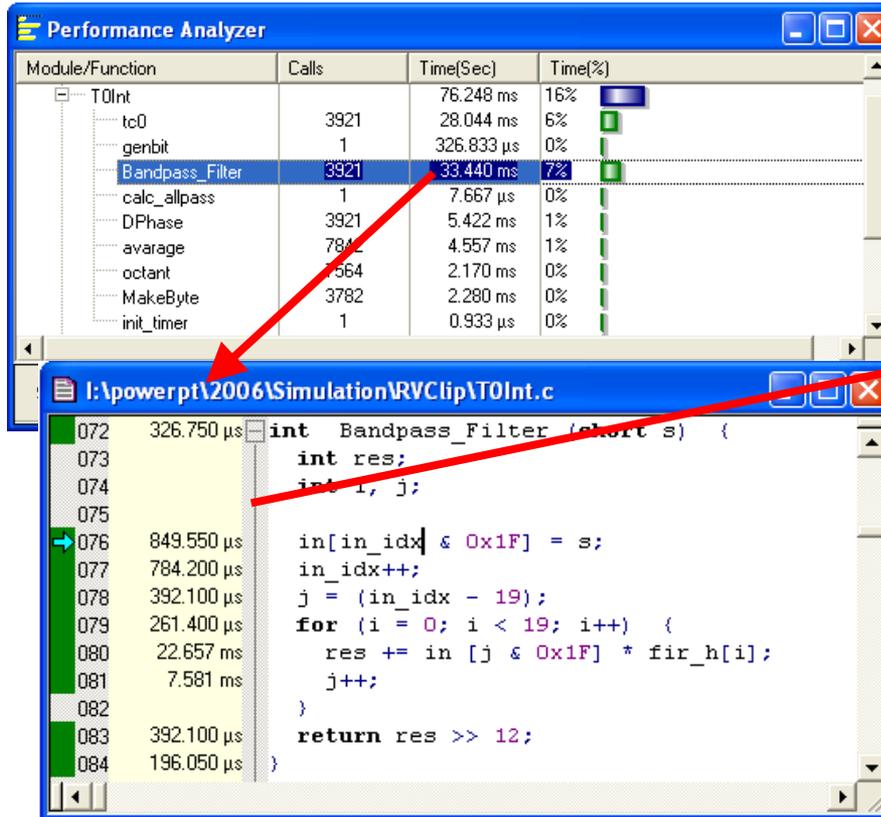
PLL quadrant of de-phaser output

Extracted Bit before final filtering

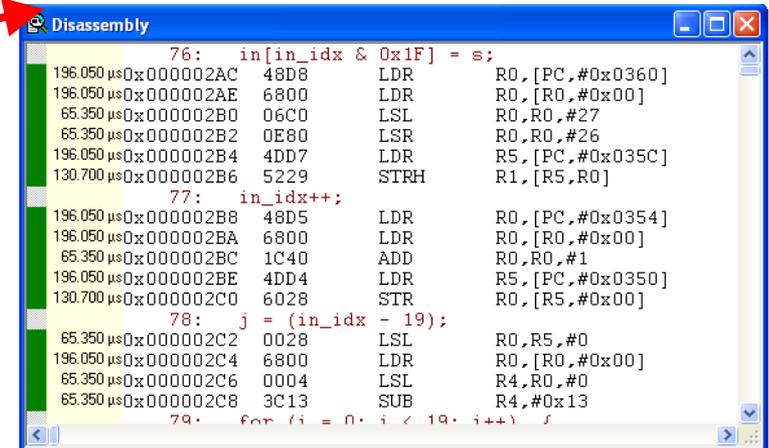
Use Case: Modem Receiver for CLID

▶ Analysis of required CPU time of the final algorithm

- Open the Performance Analyzer that records execution timing



- Timing of modules + functions
- Timing of C statements
- Timing of CPU instructions

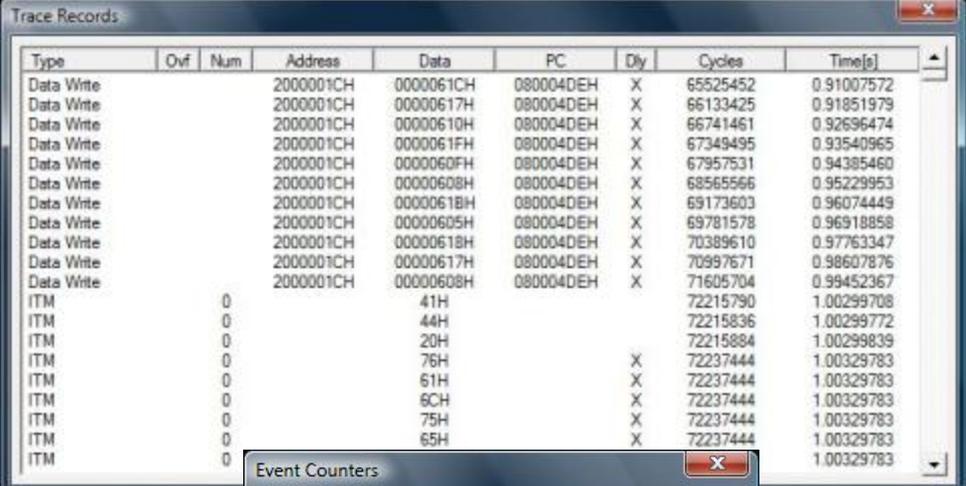


<http://www.keil.com/download/docs/326.asp>

SWV Trace Windows

Trace Records

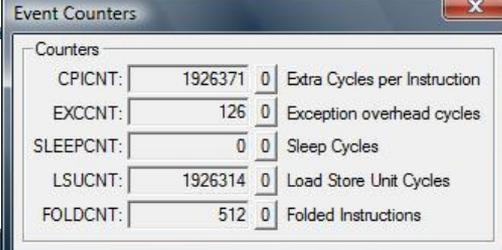
- Time stamps, PC sample, read/write accesses
- Updated while target system is running



Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Write			2000001CH	0000061CH	080004DEH	X	65525452	0.91007572
Data Write			2000001CH	00000617H	080004DEH	X	66133425	0.91851979
Data Write			2000001CH	00000610H	080004DEH	X	66741461	0.92696474
Data Write			2000001CH	0000061FH	080004DEH	X	67349495	0.93540965
Data Write			2000001CH	0000060FH	080004DEH	X	67957531	0.94385460
Data Write			2000001CH	00000608H	080004DEH	X	68565566	0.95229953
Data Write			2000001CH	00000618H	080004DEH	X	69173603	0.96074449
Data Write			2000001CH	00000605H	080004DEH	X	69781578	0.96918858
Data Write			2000001CH	00000618H	080004DEH	X	70389610	0.97763347
Data Write			2000001CH	00000617H	080004DEH	X	70997671	0.98607876
Data Write			2000001CH	00000608H	080004DEH	X	71605704	0.99452367
ITM		0		41H			72215790	1.00299708
ITM		0		44H			72215836	1.00299772
ITM		0		20H			72215884	1.00299839
ITM		0		76H		X	72237444	1.00329783
ITM		0		61H		X	72237444	1.00329783
ITM		0		6CH		X	72237444	1.00329783
ITM		0		75H		X	72237444	1.00329783
ITM		0		65H		X	72237444	1.00329783
ITM		0						1.00329783

Event Counters

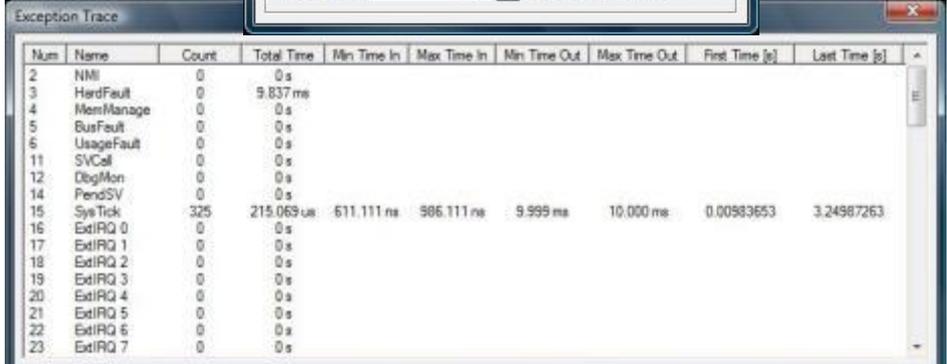
- Real-time values of event counters



Counters	Value	Description
CPICNT:	1926371	Extra Cycles per Instruction
EXCCNT:	126	Exception overhead cycles
SLEEP CNT:	0	Sleep Cycles
LSUCNT:	1926314	Load Store Unit Cycles
FOLD CNT:	512	Folded Instructions

Exception Trace

- Statistical information about program exceptions



Num	Name	Count	Total Time	Min Time In	Max Time In	Min Time Out	Max Time Out	First Time [s]	Last Time [s]
2	NMI	0	0 s						
3	HardFault	0	9.837 ms						
4	MemManage	0	0 s						
5	BusFault	0	0 s						
6	UsageFault	0	0 s						
11	SVCall	0	0 s						
12	DebugMon	0	0 s						
14	PendSV	0	0 s						
15	SysTick	325	215.069 us	611.111 ns	986.111 ns	9.999 ms	10.000 ms	0.00983653	3.24987263
16	ExtIRQ 0	0	0 s						
17	ExtIRQ 1	0	0 s						
18	ExtIRQ 2	0	0 s						
19	ExtIRQ 3	0	0 s						
20	ExtIRQ 4	0	0 s						
21	ExtIRQ 5	0	0 s						
22	ExtIRQ 6	0	0 s						
23	ExtIRQ 7	0	0 s						