

# 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  $\mu$ Vision Environment
- Architecture overview
- Peripherals
- Interrupts and register banks



# Keil: Tools for ARM-Powered Devices

ARM C/C++ Compiler

µVision  
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

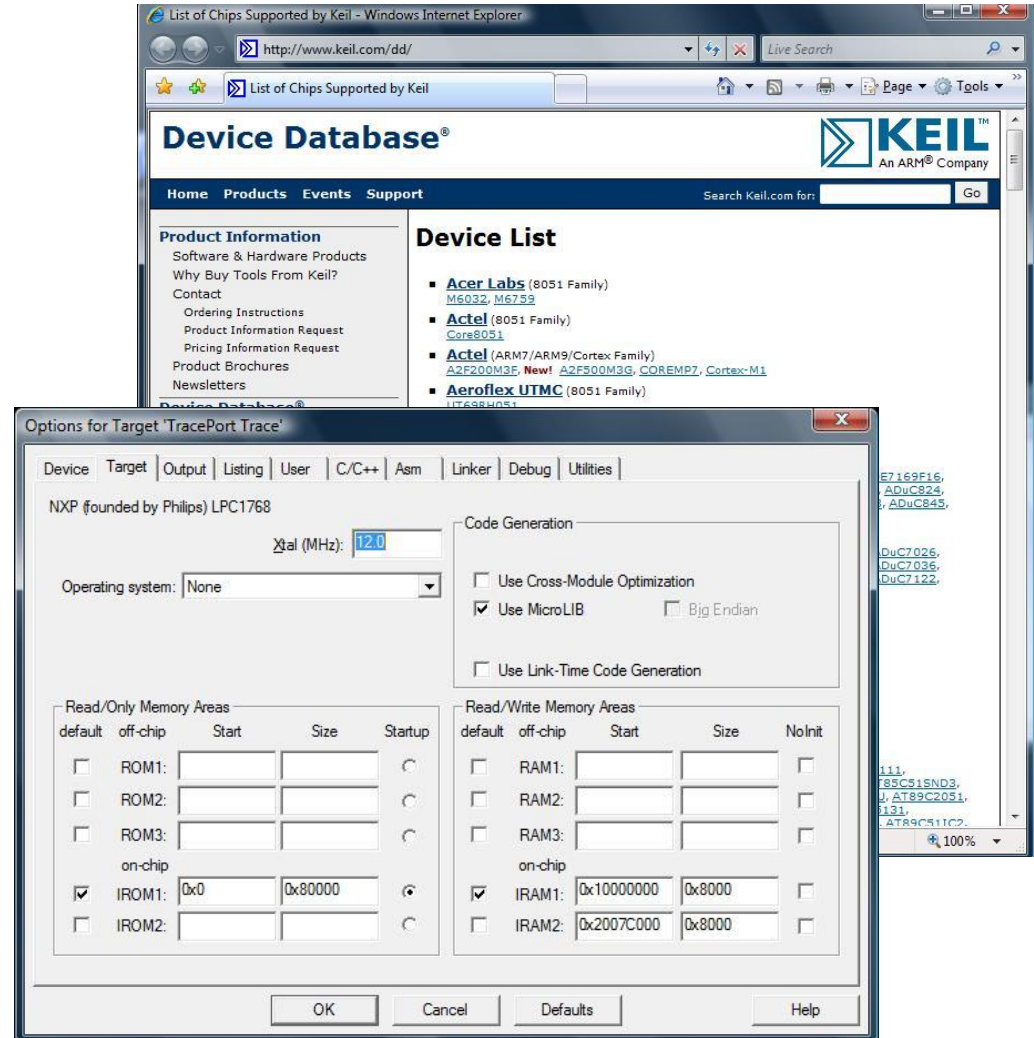
Move forward with the easy-to-use tools



# Step 1: Create a Program

Select the device and specify the target hardware

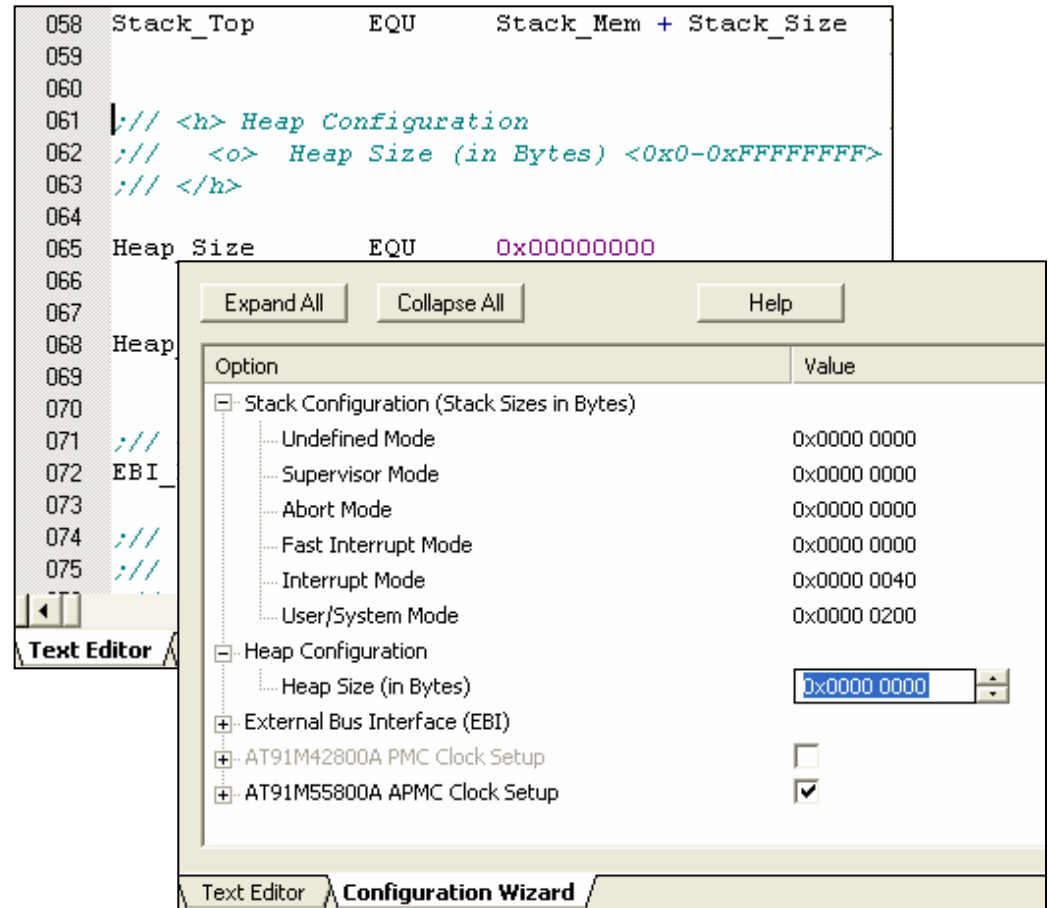
- Find the perfect part with the website tools
- Jump-start your project with online device documentation



# Step 2: Program Setup

Configure the device and create application code

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



## Analyze the program with $\mu$ Vision device simulation

37: int getchar (void) {

38:

260.700 μs	0x01000F06	4770	BX	LR
	0x01000F08	FFFC0014	DD	0xFFFC0014
	0x01000F0C	FFFC001C	DD	0xFFFC001C
	0x01000F10	E59FC000	LDR	R12,[PC]
	0x01000F14	E12FFF1C	BX	R12
	0x01000F18	01000F1D	DD	0x01000F1D

39: while (! (USD\_CSR & USD\_RXRDY)).

444.012 n

204.929 n

136.619 n

136.619 n

409.857 n

Min Time: 0.0 s Max Time: 0.349597 s Range: 50.00000 ms Grid: 2.500000 ms Zoom: In Out

Ton... 0x7FFF -0x7FFF -16478

C... 0x1 0x0 0 -> 0

AN0 3.5 1.5 2.44868 -> 2.72755

Signal 0x2BC 0x0 0 0

0.222500 s 0.235535 s 0.247500 s

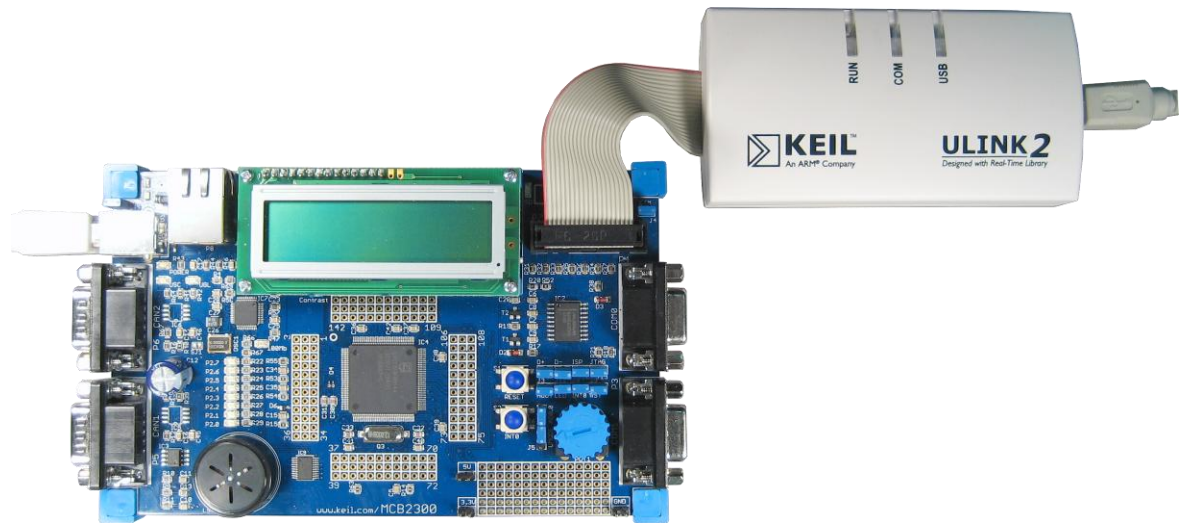
# Step 4: Target Debugging

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



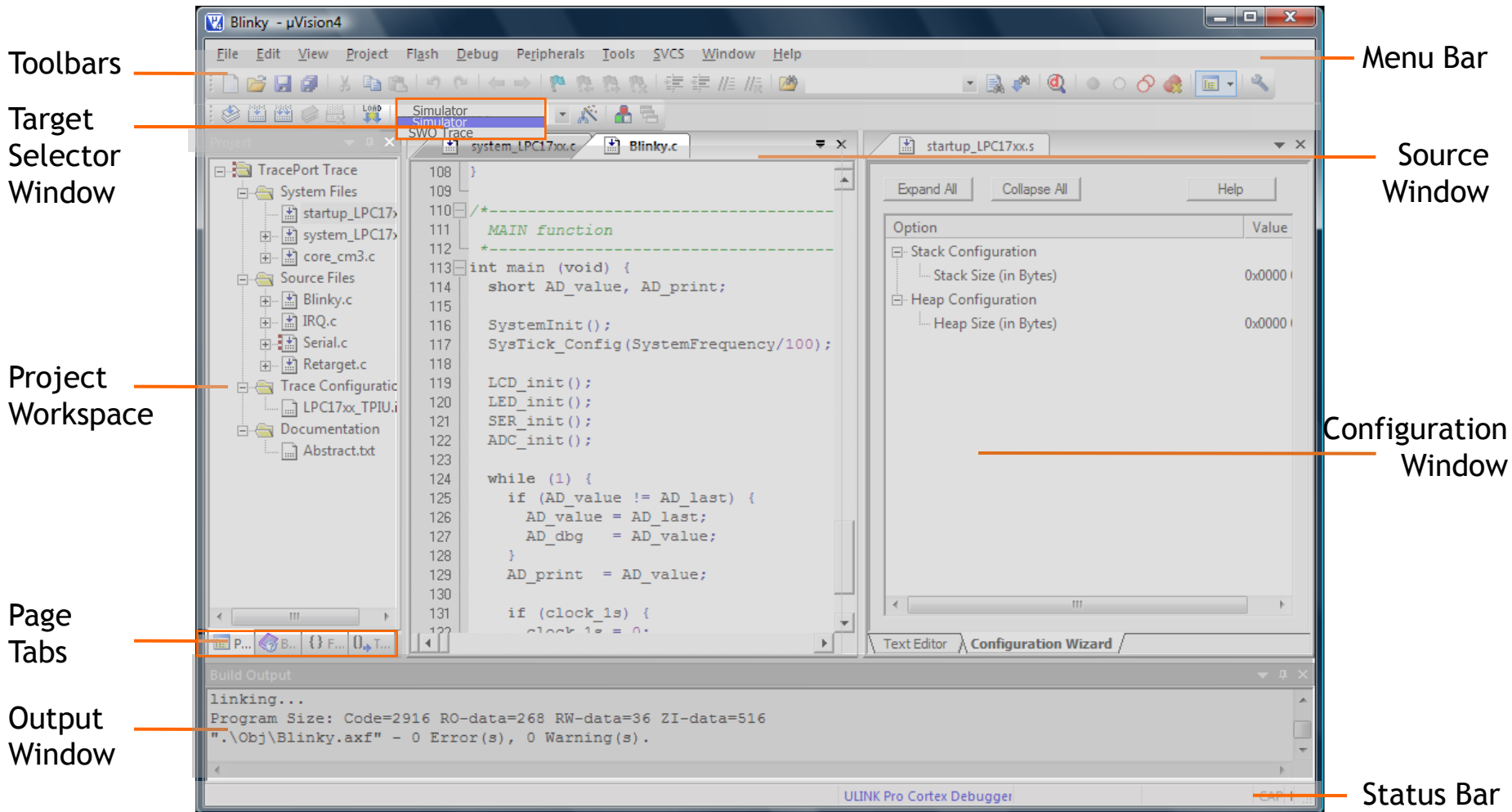


# $\mu$ Vision4 Editor

The Keil Integrated Development Environment



# µVision - Build and Edit Mode

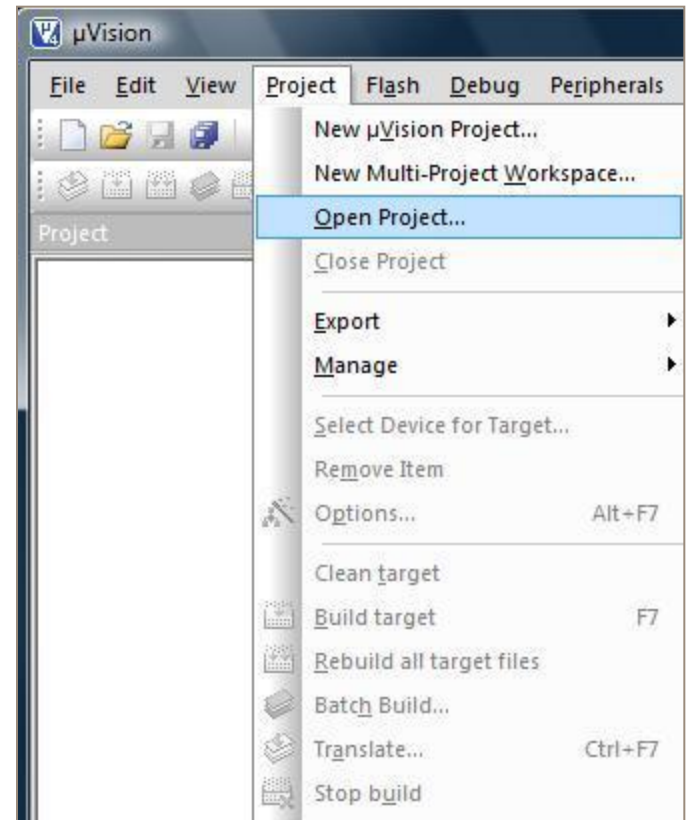


# Example 1 - Open and Build a Project

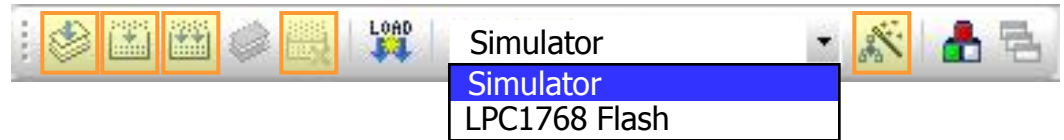
➤ Go to Project → 'Open Project' and Open:  
`..\ARM\Examples\Blinky.Uv2`

## ➤ 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 RO-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  $\mu$ 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 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 on the right, showing control and calibration settings. The 'Watch' and 'Memory' windows at the bottom show the current state of the system. The status bar at the very bottom indicates the simulation is running.







Labels pointing to specific UI elements:

- Toolbars
- Menu Bar
- Source Window
- Peripheral Dialog
- Project Workspace
- Page Tabs
- Watch Window
- Memory Window
- Output Window
- Status Bar

# 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:

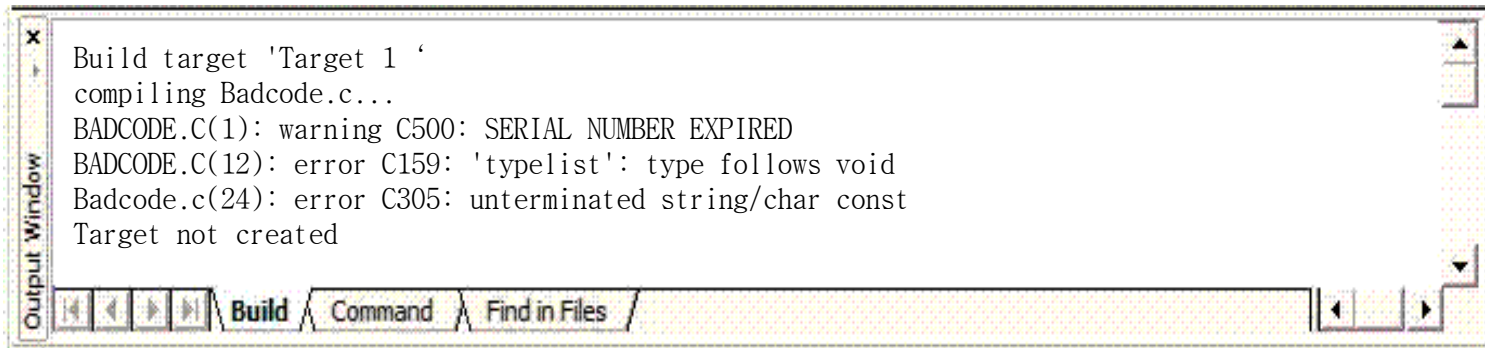
- Color syntax
- Error messages
- Hot keys

```
/*-----  
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;  
  
    for (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



```
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
```



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 errorrrs. (Spelling Errors)
- 🏗 Rebuild the target after each error fix to update the build tab

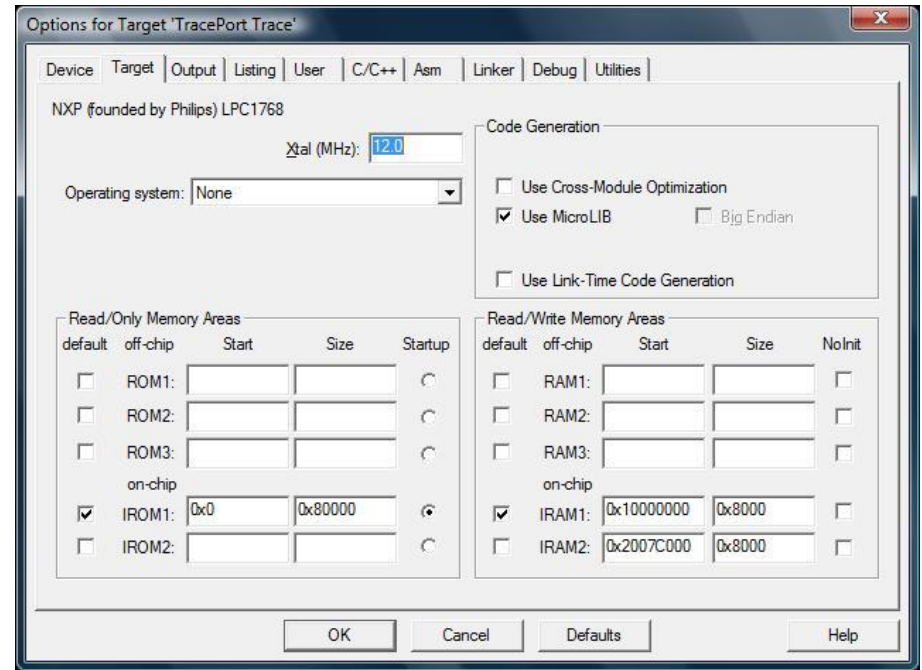


# Memory - Target Tab for ARM



## 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

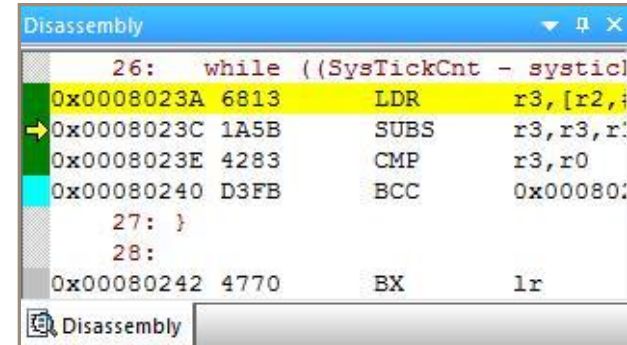
- Shows the target program as mixed source and assembly code

## Watch window

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

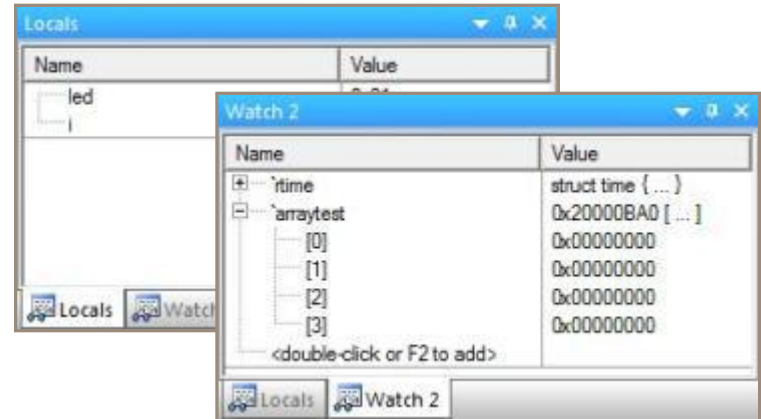
## Memory window

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

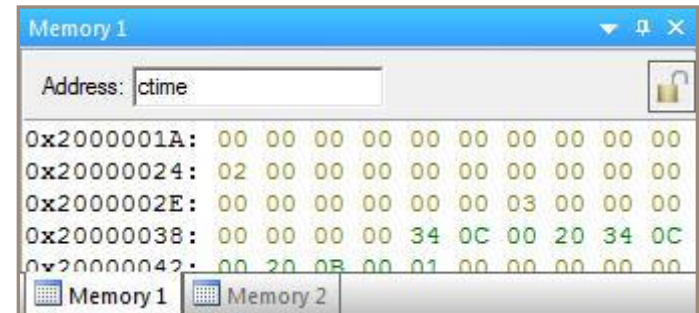


Disassembly window showing a while loop. The code is displayed in a table with columns for address, disassembly, and comment. The current instruction is highlighted in yellow.

Address	Disassembly	Comment
26:	while ((SysTickCnt - systicl	
0x0008023A 6813	LDR	r3, [r2, i
0x0008023C 1A5B	SUBS	r3, r3, r
0x0008023E 4283	CMP	r3, r0
0x00080240 D3FB	BCC	0x000802
27: }		
28:		
0x00080242 4770	BX	lr



Locals window showing a variable named 'led' with a value of 0. The Watch window shows a variable named 'arraytest' with a value of 0x20000BA0. The Watch window also shows a variable named 'ctime' with a value of struct time {...}.



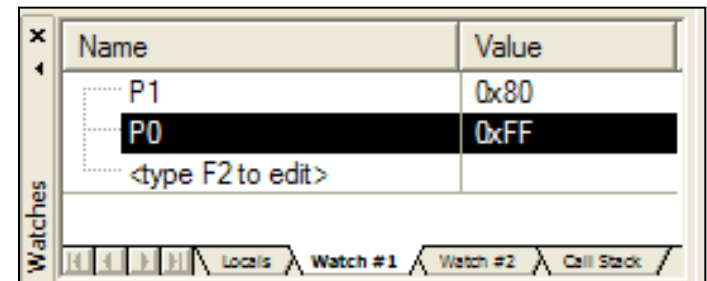
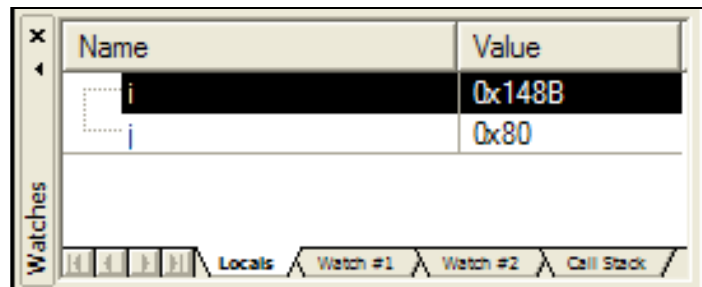
Memory window showing memory contents. The address is set to ctime. The memory is displayed in a table with columns for address and data.

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



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

```
378:          SUB      R0, R0, #SVC_Stack_Size
379:
380: // Enter User Mode and set its Stack Pointer
381: 0x000000D0 E2400020 SUB      R0,R0,#SVC_Stack_Size(0x
382: 0x000000D4 E321F010 MSR      CPSR_c, #Mode_USR
383: 382:          MOV      SP, R0
384: // Enter the C code
385: 0x000000D8 E1A0D000 MOV      R13,R0
386: 385:          LDR      R0,=PC?INIT
387: 0x000000DC E59F0020 LDR      R0,[PC,#0x0020]
388: 386:          TST      R0,#1 ; Bit-0 set:
389: 0x000000E0 E3100001 TST      R0,#PLL_SETUP(0x00000000
390: 387:          LDREQ    LR,=exit?A ; ARM Mode
391: 0x000000E4 059FE01C LDREQ    R14,[PC,#0x001C]
392: 388:          LDRNE    LR,=exit?T ; Thumb Mode
393: 0x000000E8 159FE01C LDRNE    R14,[PC,#0x001C]
394: 389:          BX      R0
395: 390:          ENDP
```

Code Coverage	
Update	Reset
Module: <All Modules>	
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)
CODE	
Code Coverage	

# µVision Logic Analyzer



## Timing Analysis

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



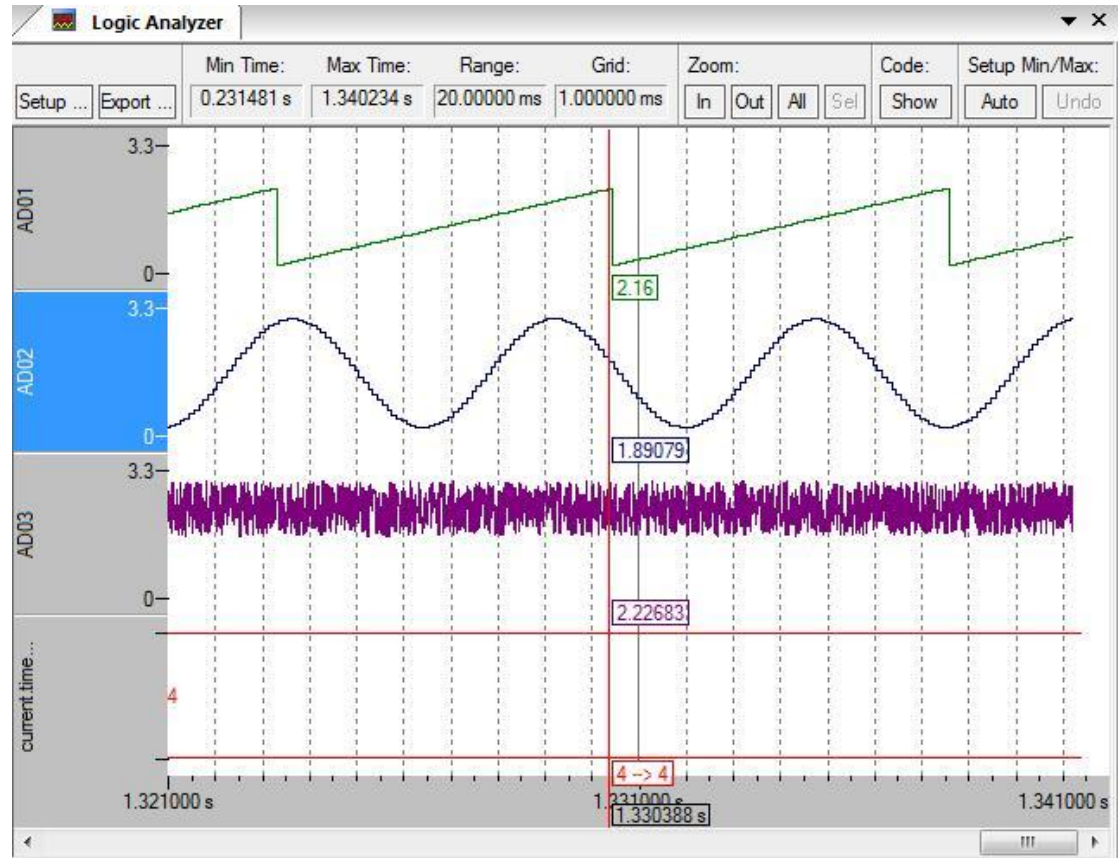
## Exact Timing

- Using Cursor Line
- Tool-Tip Delta Information



## Display Style

- Analog
- Bit
- State Change



# Logic Analyzer Setup

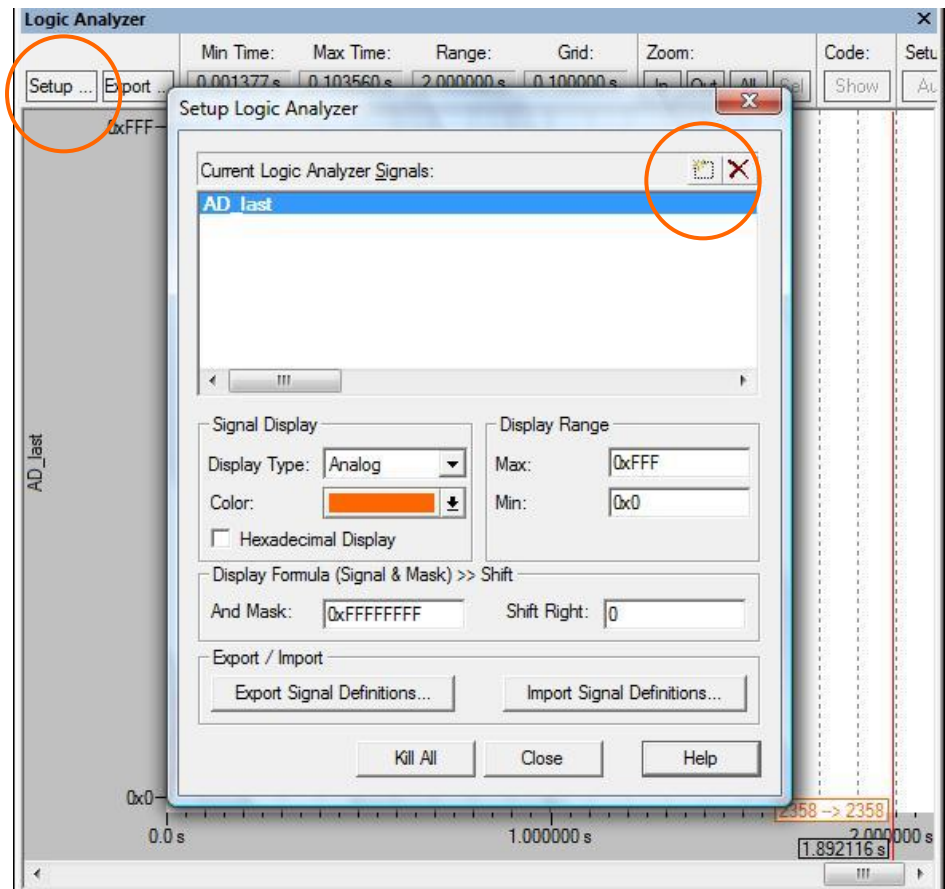


Setup is simple

- 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

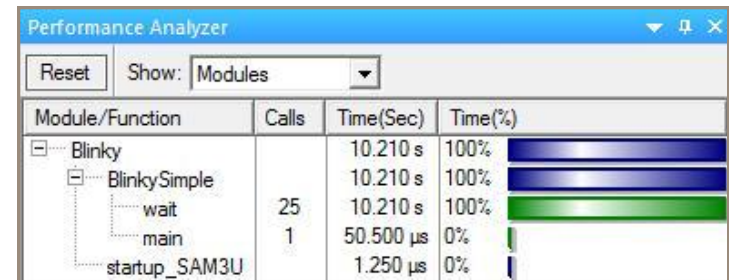
## ➤ Detailed Timing Statistics

## ➤ Shows:

- Execution Time
- Number of Executions

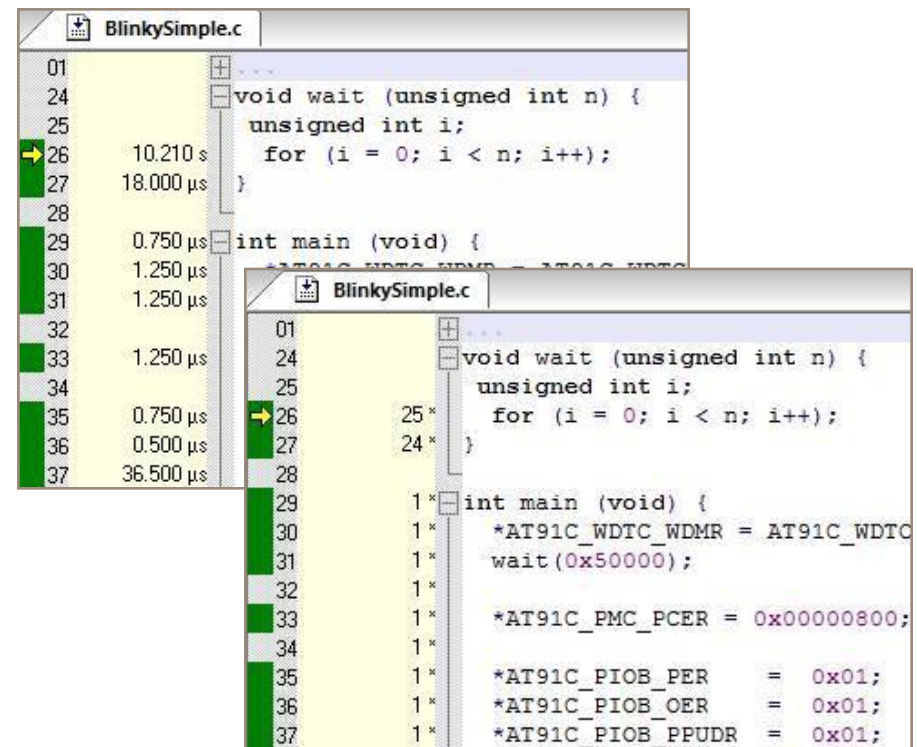
## ➤ With Flexible Views:

- Source Windows
- Disassembler Window
- Complete Project Overview



Performance Analyzer window showing execution statistics for the Blinky module. The 'Show:' dropdown is set to 'Modules'. The table lists the following data:

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%



Two windows showing the source code and disassembly for BlinkySimple.c. The top window is the Source window, and the bottom window is the Disassembler window.

**Source Window (BlinkySimple.c):**

```
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_WDMR;  
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;  
38 }
```

**Disassembler Window (BlinkySimple.c):**

```
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_WDMR;  
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;  
38 }
```

# 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  $\mu$ 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

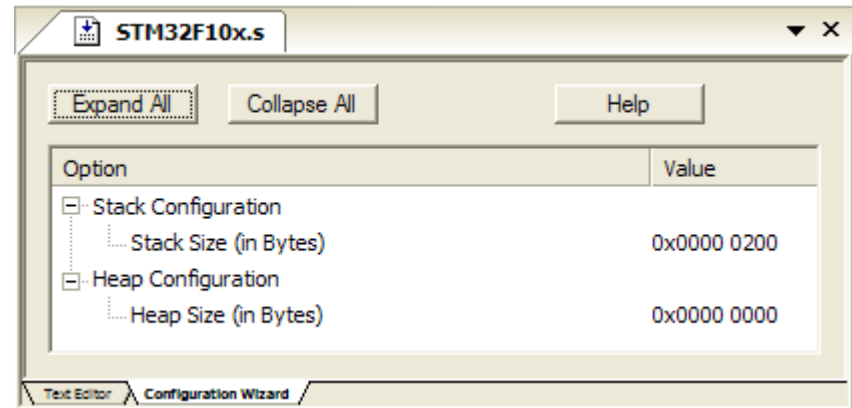
# Configuration Wizard

- ▶ 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](http://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





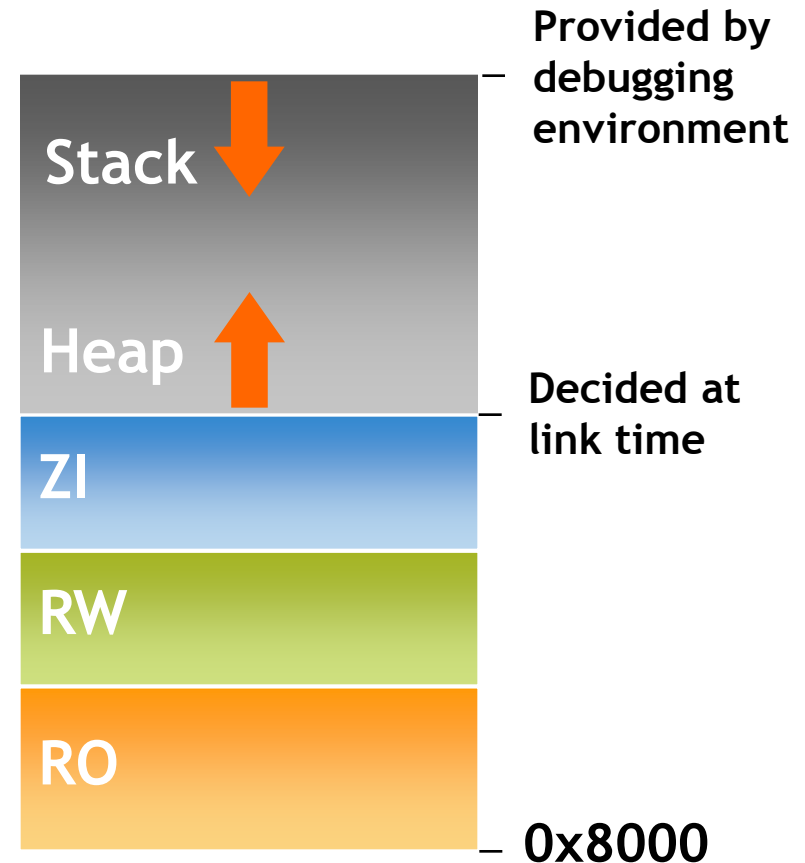
# Scatter Loading File

---

- Absolute Control over Object Placement

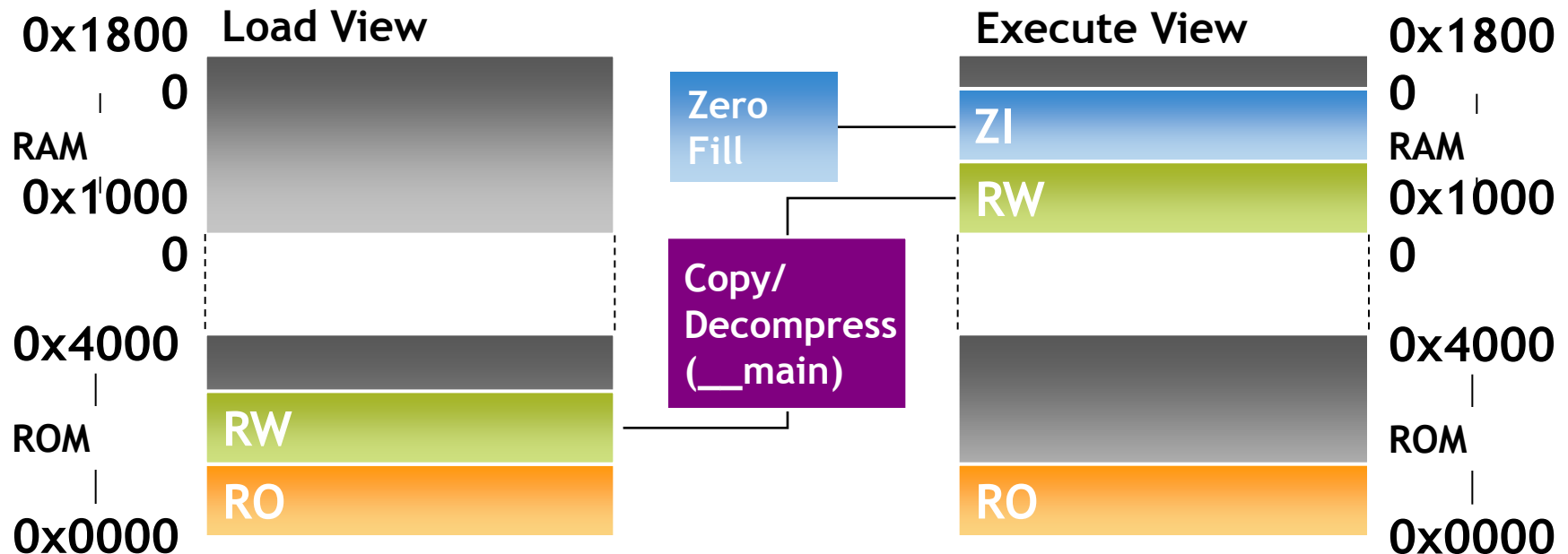
# Default Memory Map

- 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;
```

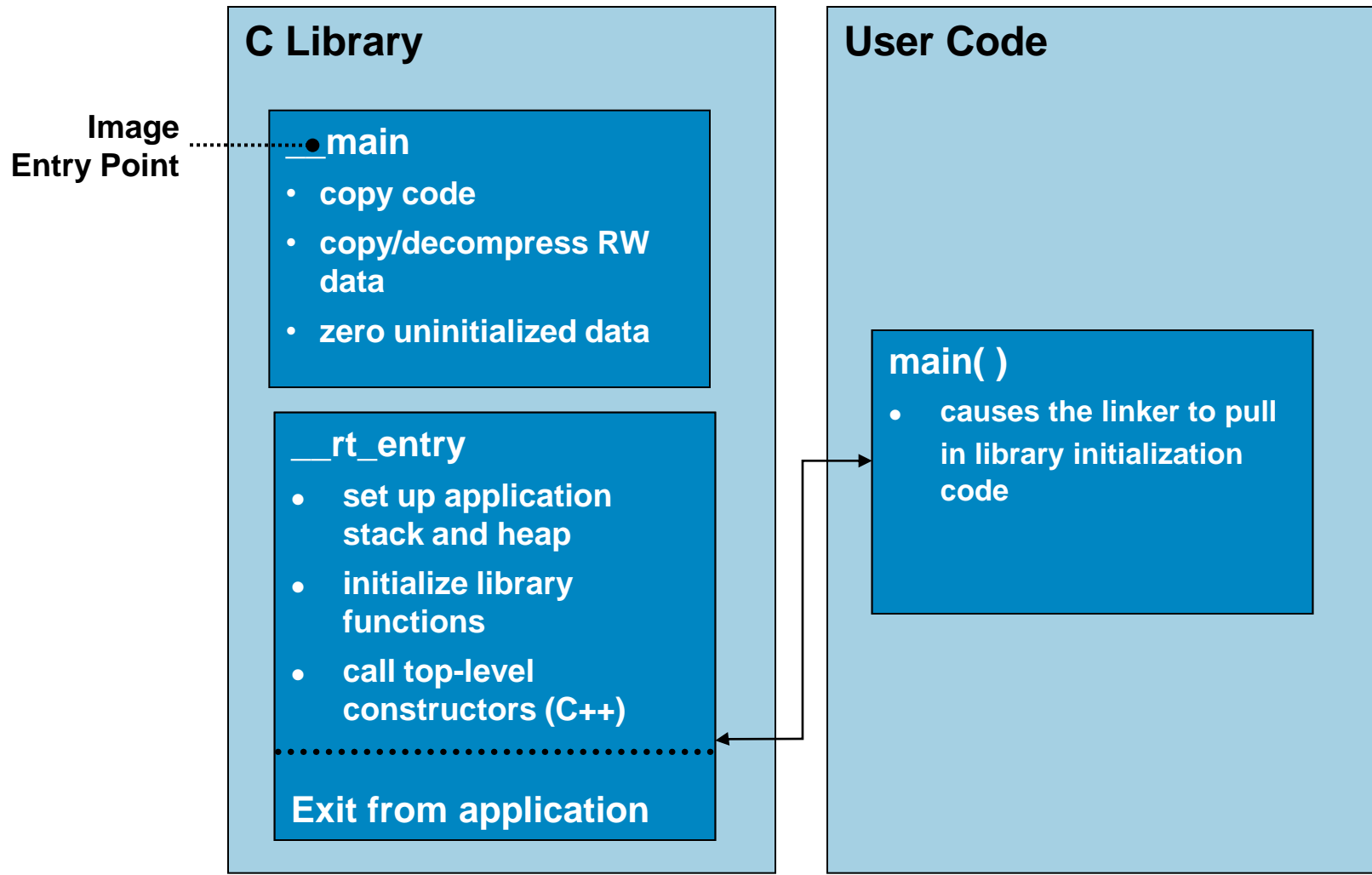
- Add an execution region to the scatter file to place registers at the required address:

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

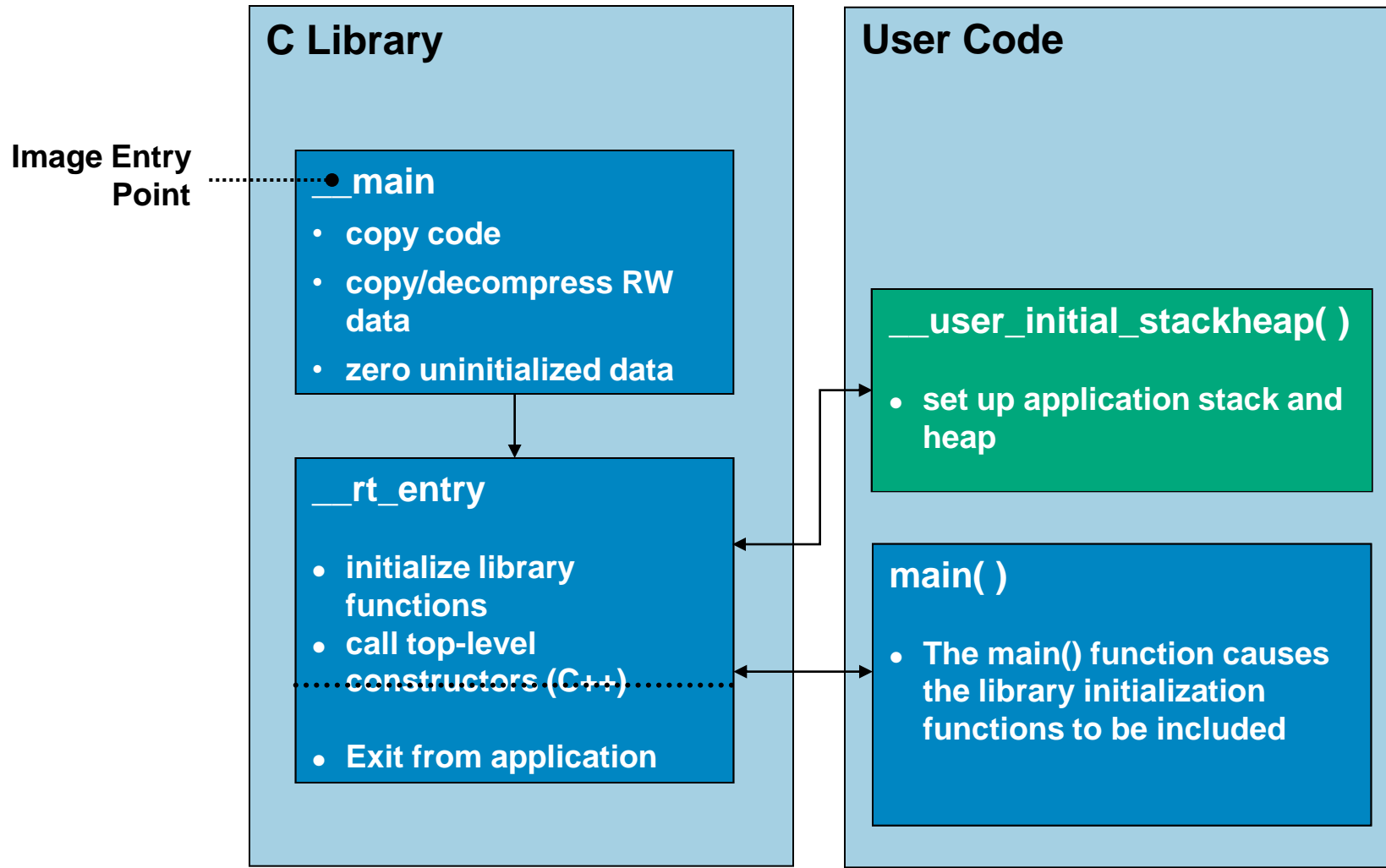
# What does `__main` do?

---

# Application Startup



# Stack and Heap Initialization





# Retarget.c File

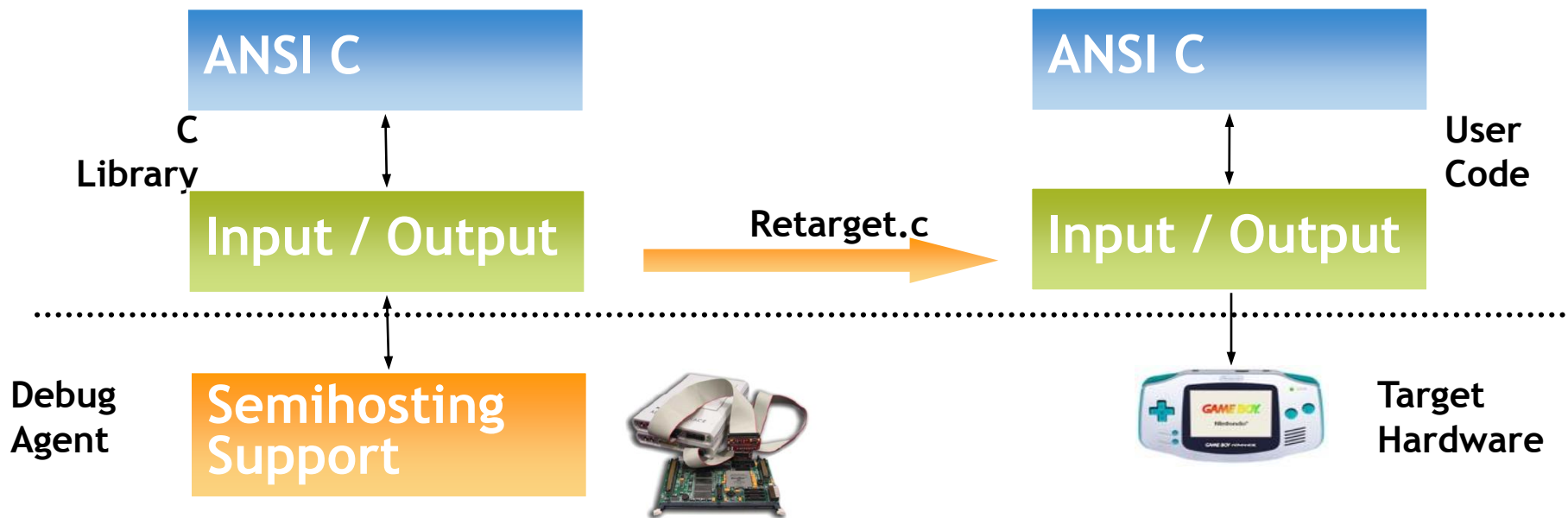
---

▢ 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 `$Sub$$main( )`
  - 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

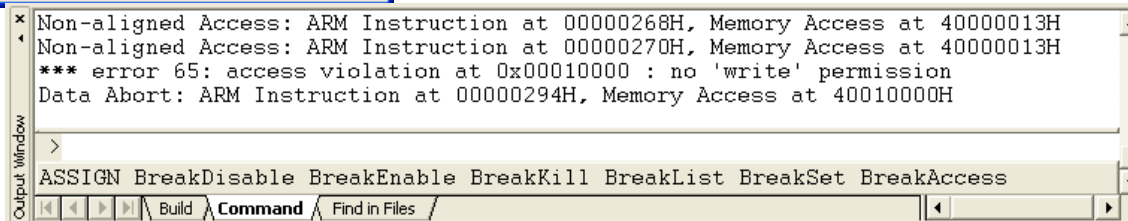
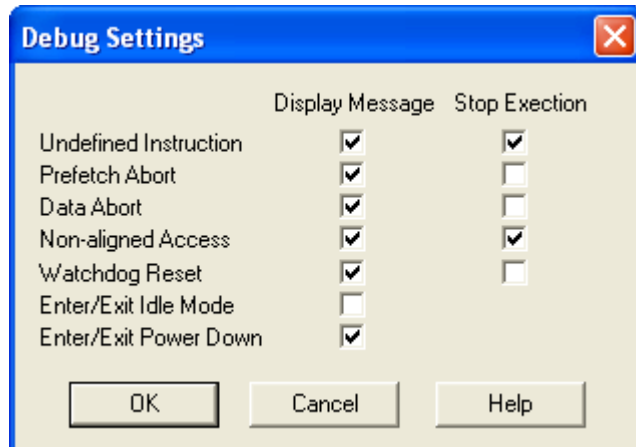
---

# 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.sc  
--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!
```

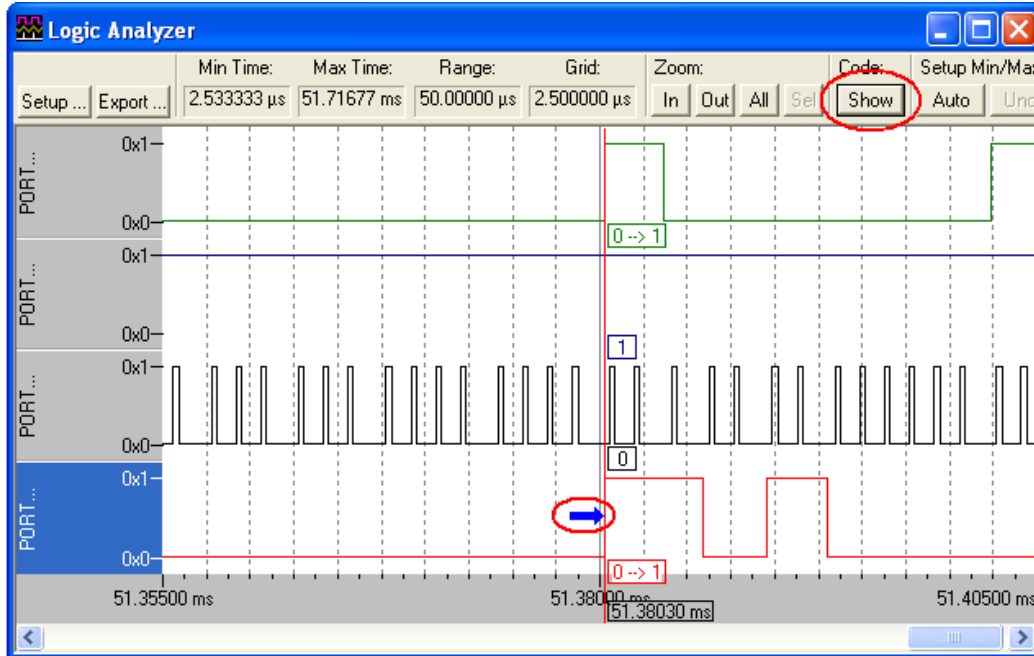
<http://www.keil.com/download/docs/323.asp>



# 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.



The source code window shows the file 'F:\VRKUV3\ULINK2\FIRMWARE2\JTAG\_CMD.c'. The code includes comments and function definitions for JTAG initialization. A red circle highlights line 291, which is part of a loop that configures JTAG pins. The code is as follows:

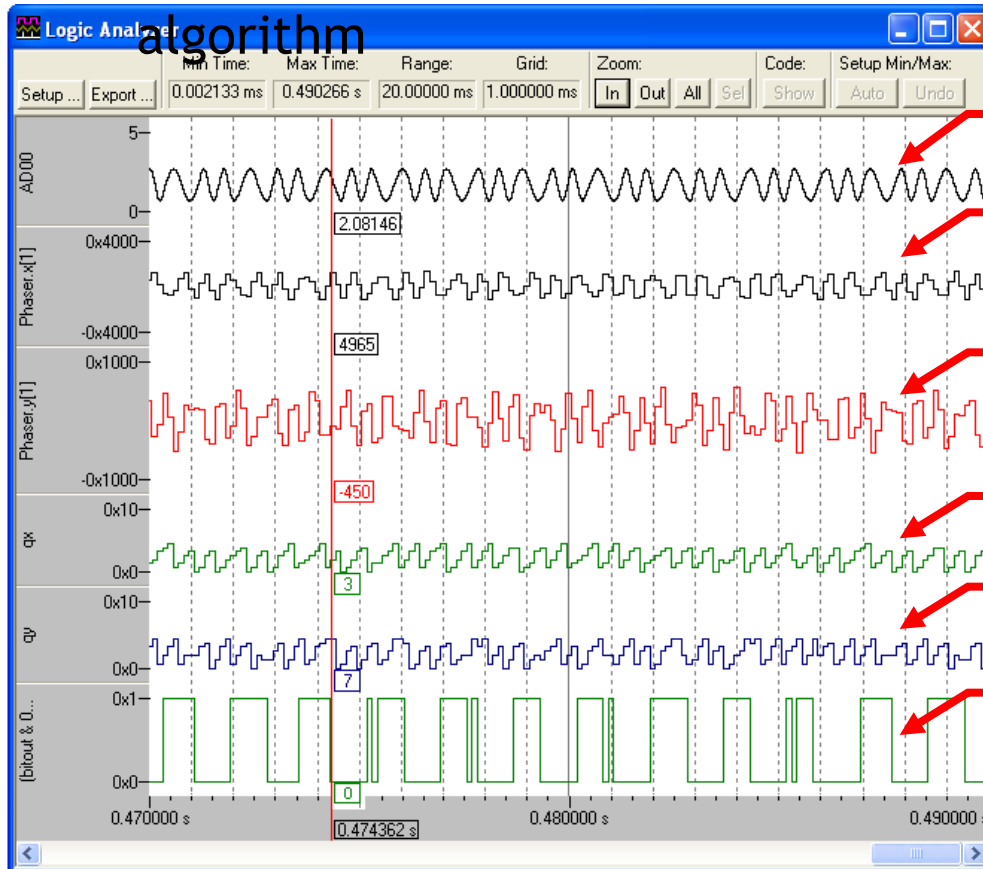
```
281 /*  
282  * Set JTAG I/O lines: TRST, TMS, TDI,  
283  */  
284 static void alt_o (U32 v) {  
285     U32 i;  
286     U32 x;  
287  
288     x = PIN_TRST | PIN_TMS | PIN_TDI;  
289     x ^= ~v;  
290     if (x) FIOCLR = x;  
291     if (v) FIOSET = v;  
292     for (i = 0; i < NoWS; i++);  
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



Analog modem signal on CPU input

Digital samples after filtering

De-phase algorithm performs phase shifting

PLL quadrant of de-phaser input

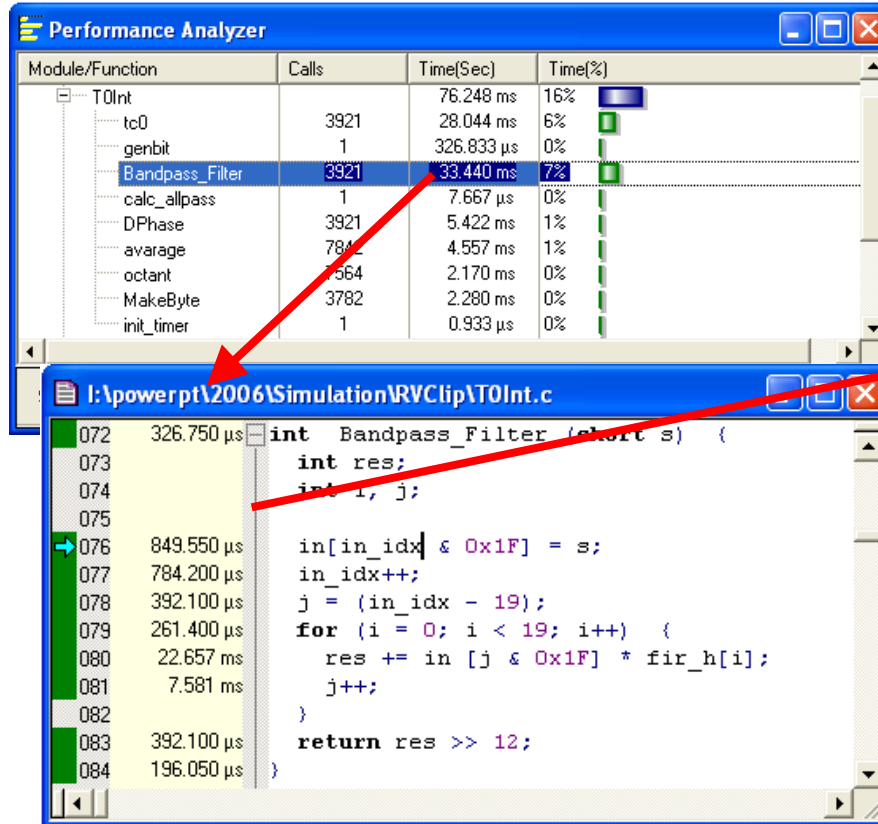
PLL quadrant of de-phaser output

Extracted Bit before final filtering

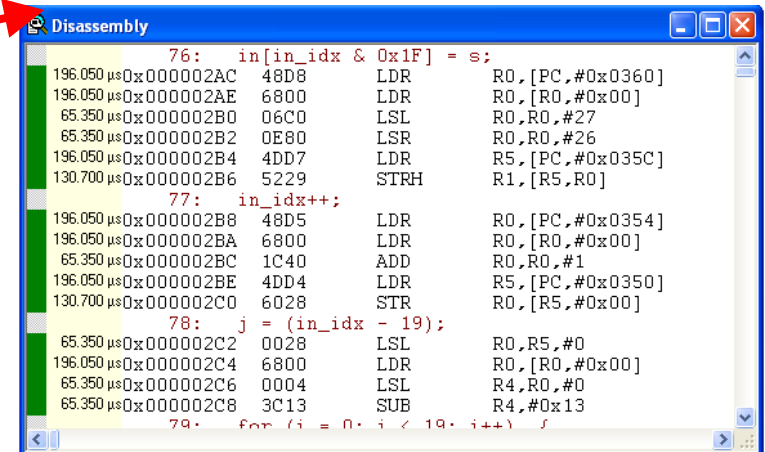
# Use Case: Modem Receiver for CLID

## ► 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

## Event Counters

- Real-time values of event counters

## Exception Trace

- Statistical information about program exceptions

