Keil Toolset Workshop

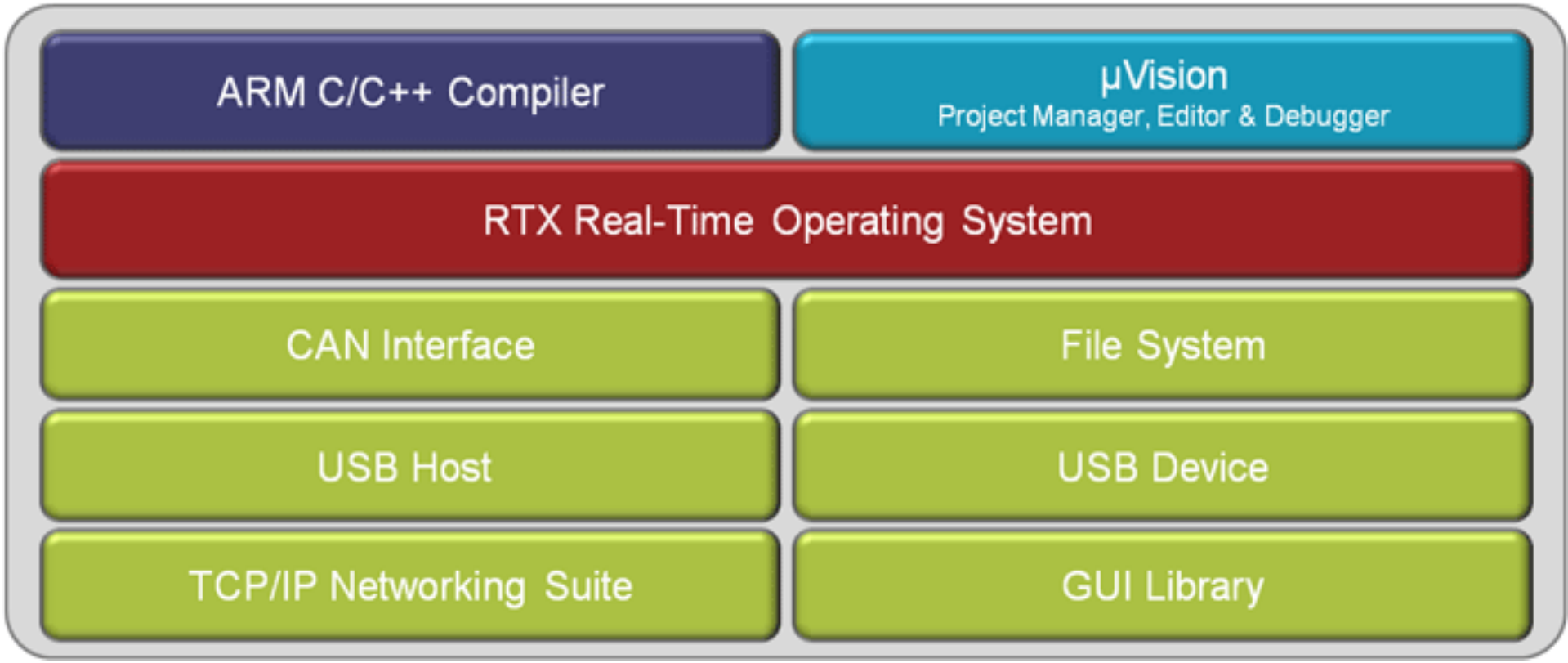
Getting a leg up on ARM development

Kevin Bernhardt
Product Specialist

Today's Agenda

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

Keil: Tools for ARM-Powered Devices



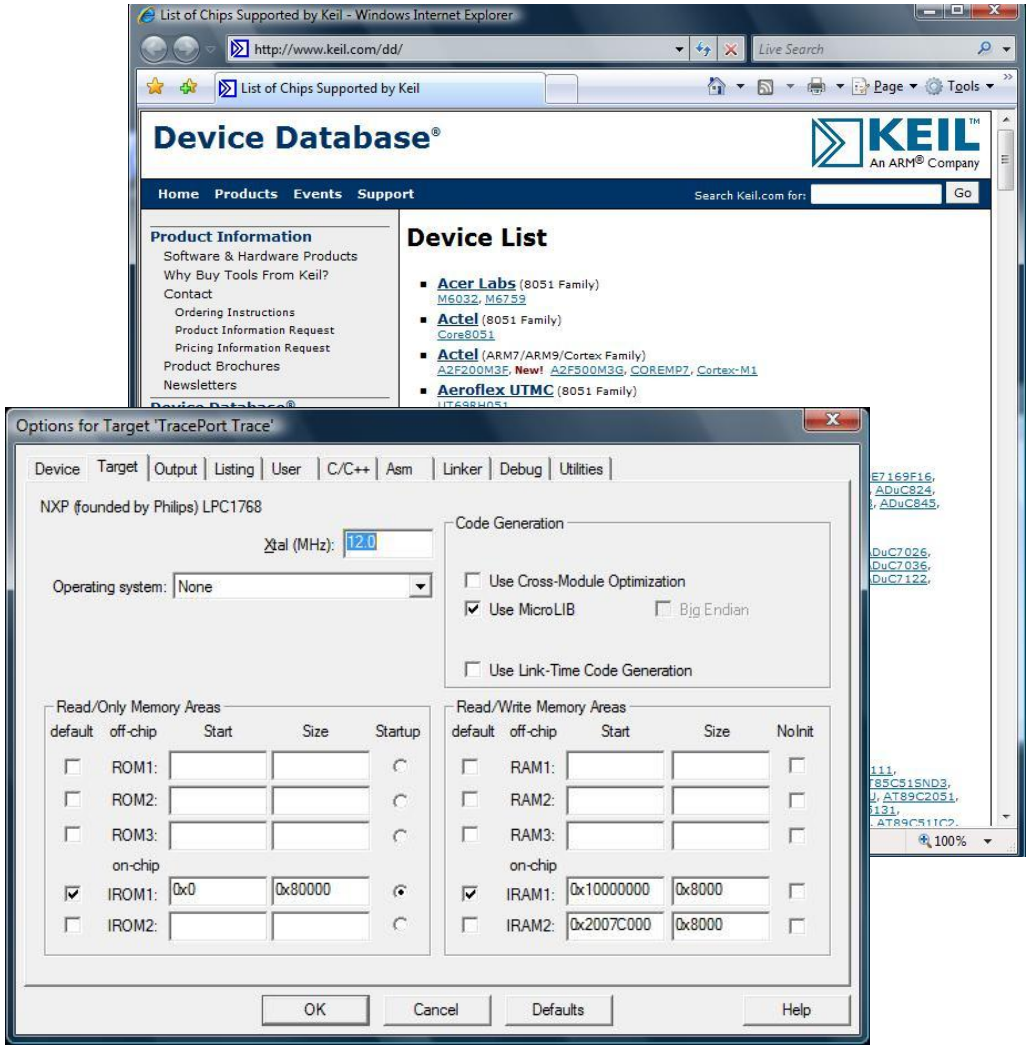
Four Steps

Move forward with the easy-to-use tools

Step 1: Create a Program

Select the device and specify the target hardware

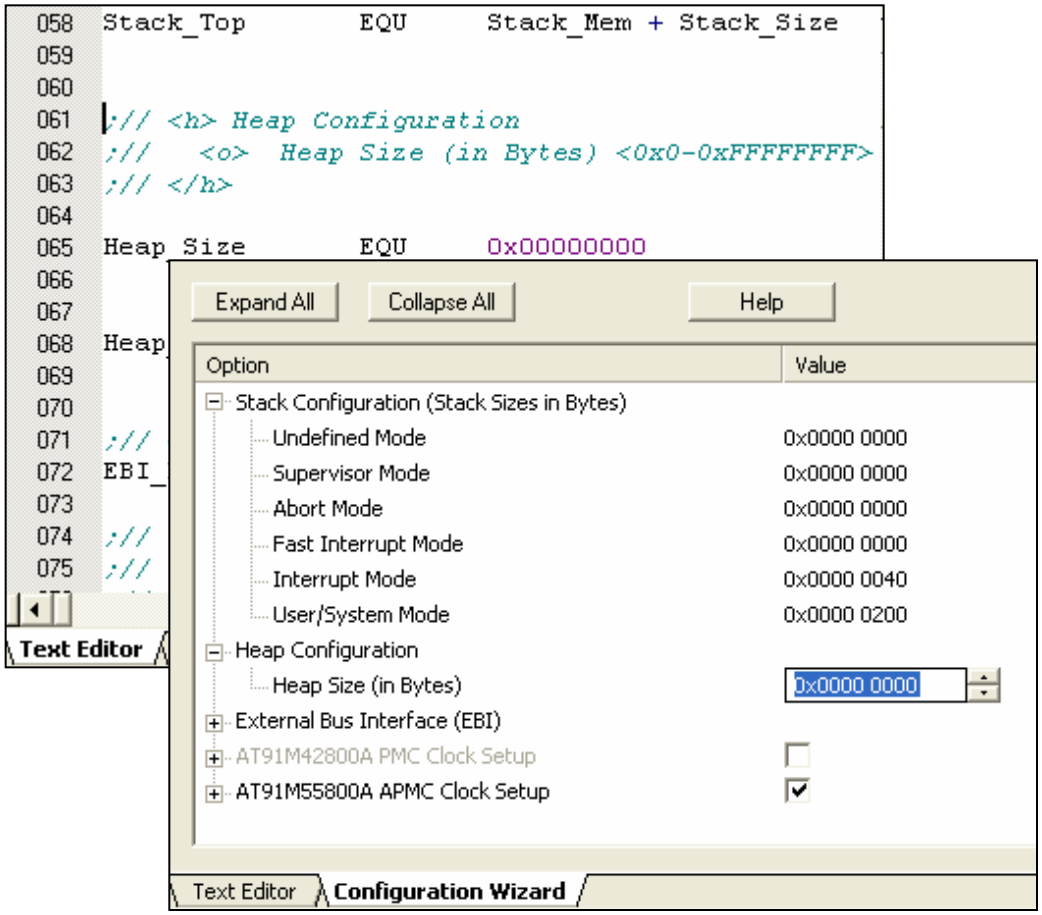
- Find the perfect part with the website tools
- Jump-start your project with online device documentation



Step 2: Program Setup

Configure the device and create application code

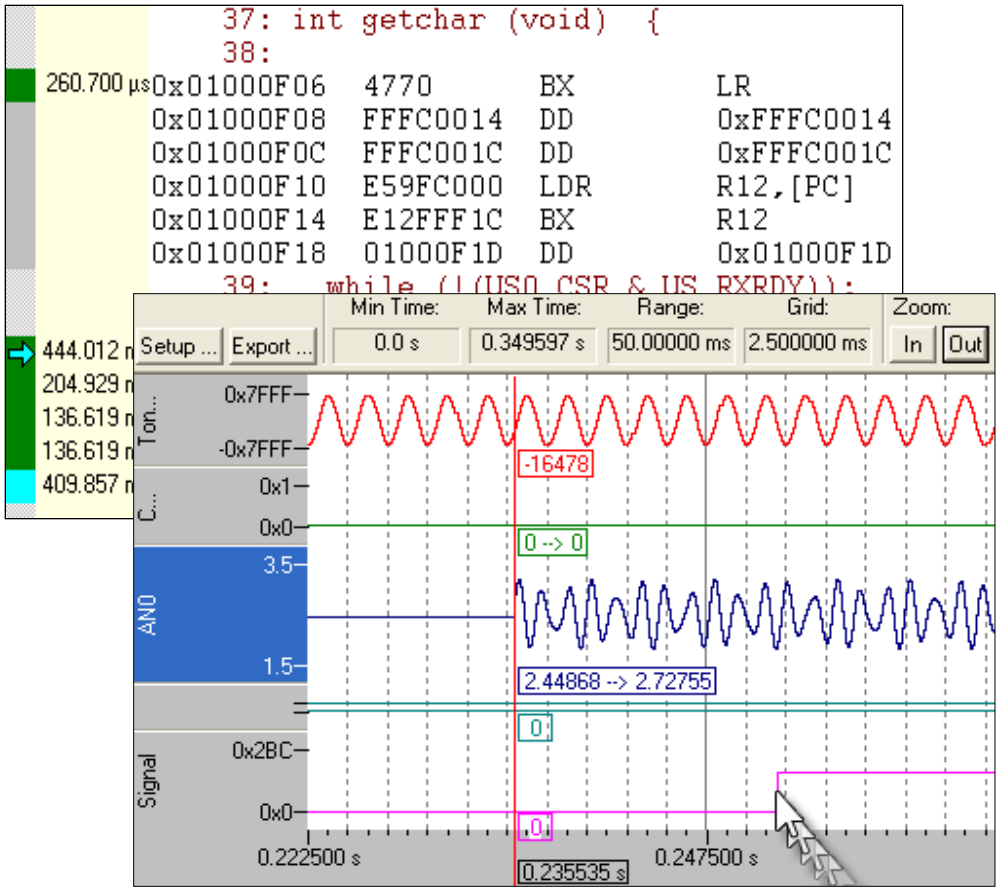
- Use Keil's pre-made startup files and chip specific examples



Step 3: Program Testing

Analyze the program with µVision device simulation

- Take advantage of the analysis tools in the simulator



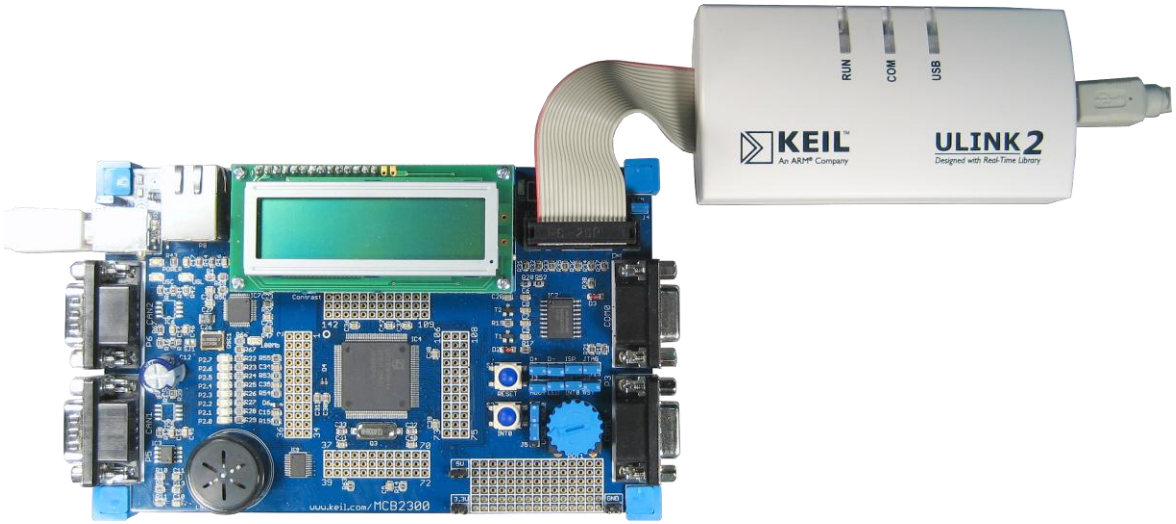
Step 4: Target Debugging

Flash download and final testing in target hardware

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

Target Debug

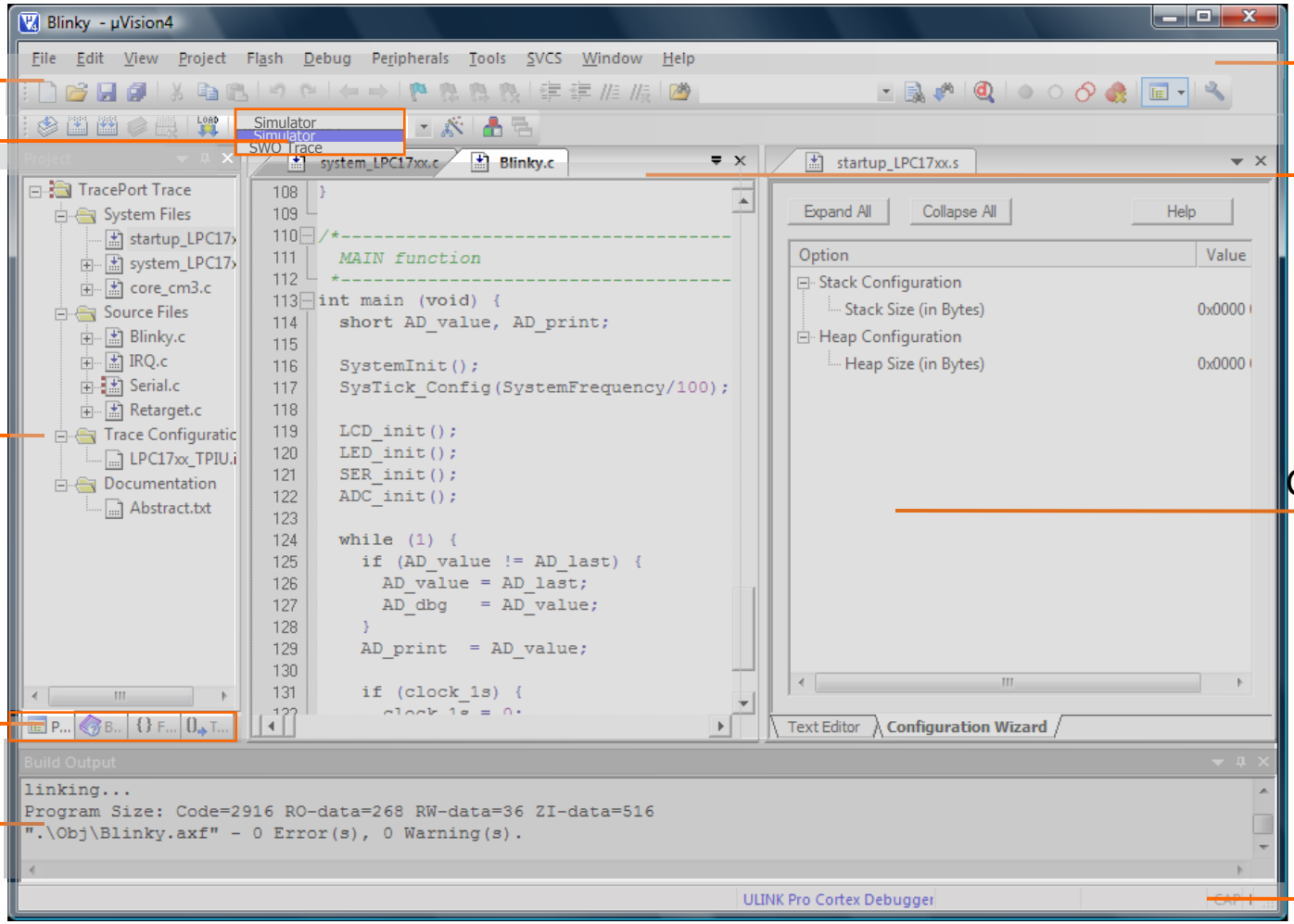
ULINK2 USB-JTAG Adapter
with Real-Time Agent



µVision4 Editor

The Keil Integrated Development Environment

µVision - Build and Edit Mode



Toolbars

Target Selector Window

Project Workspace

Page Tabs

Output Window

Menu Bar

Source Window

Configuration Window

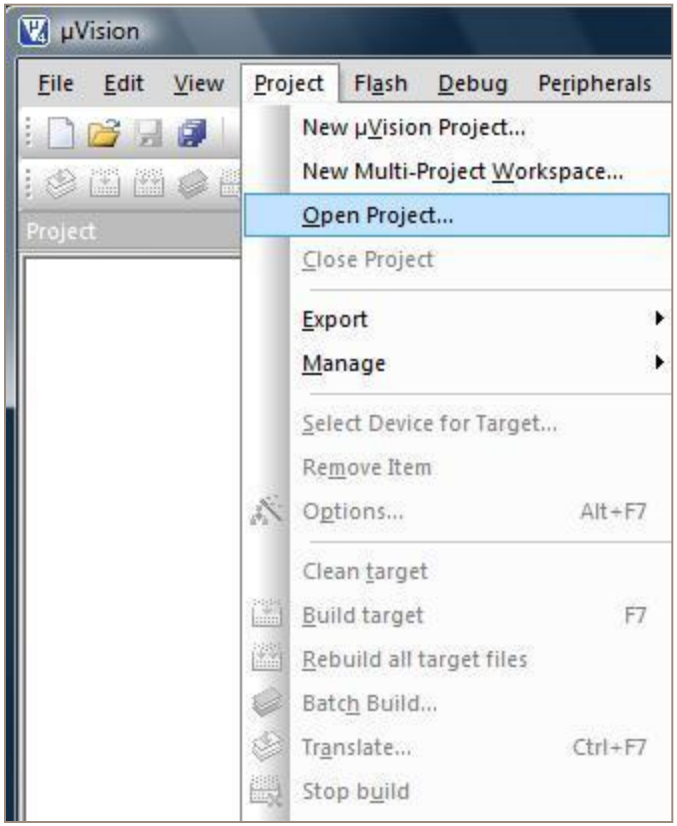
Status Bar

Example 1 - Open and Build a Project

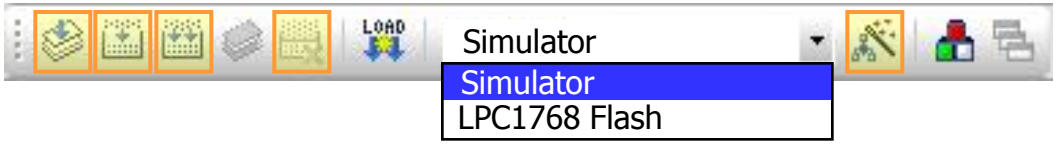
- Go to Project → 'Open Project' and Open:

  ..\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



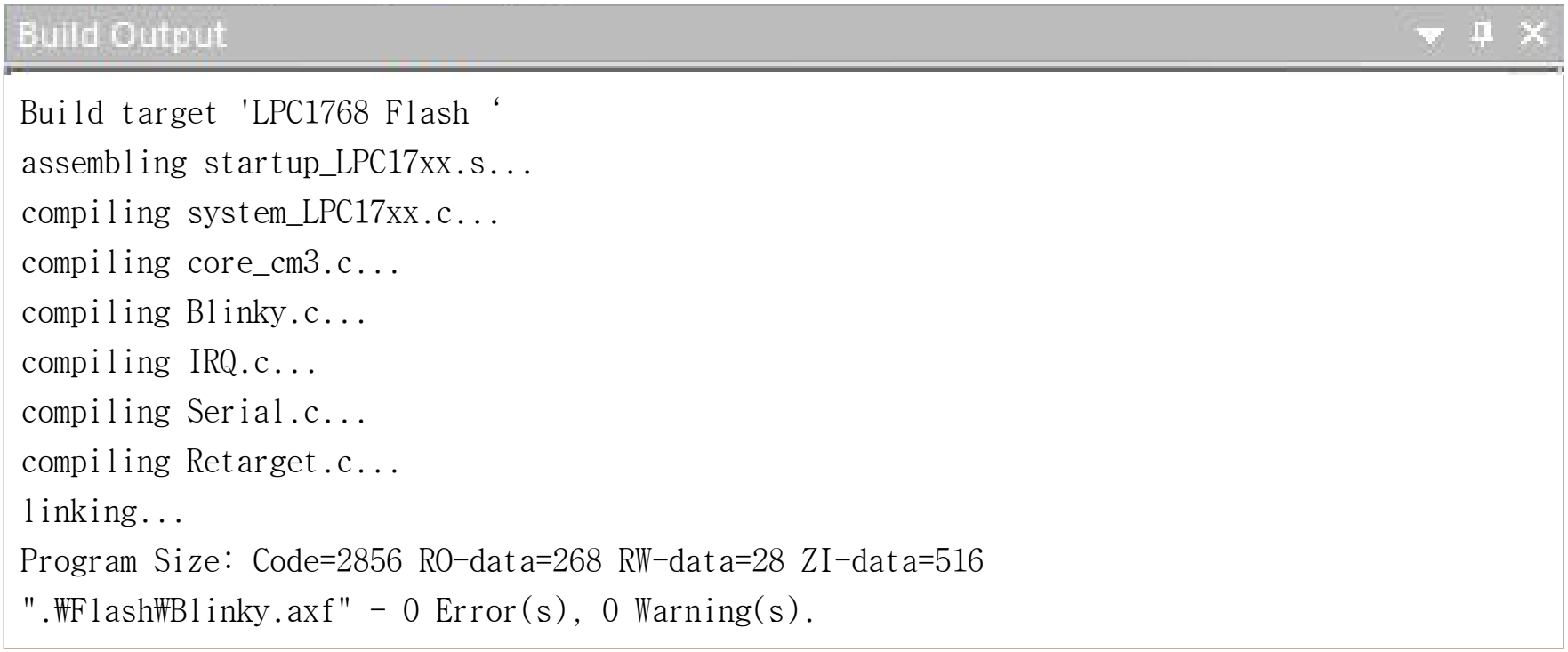
µVision Build Toolbar



- 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

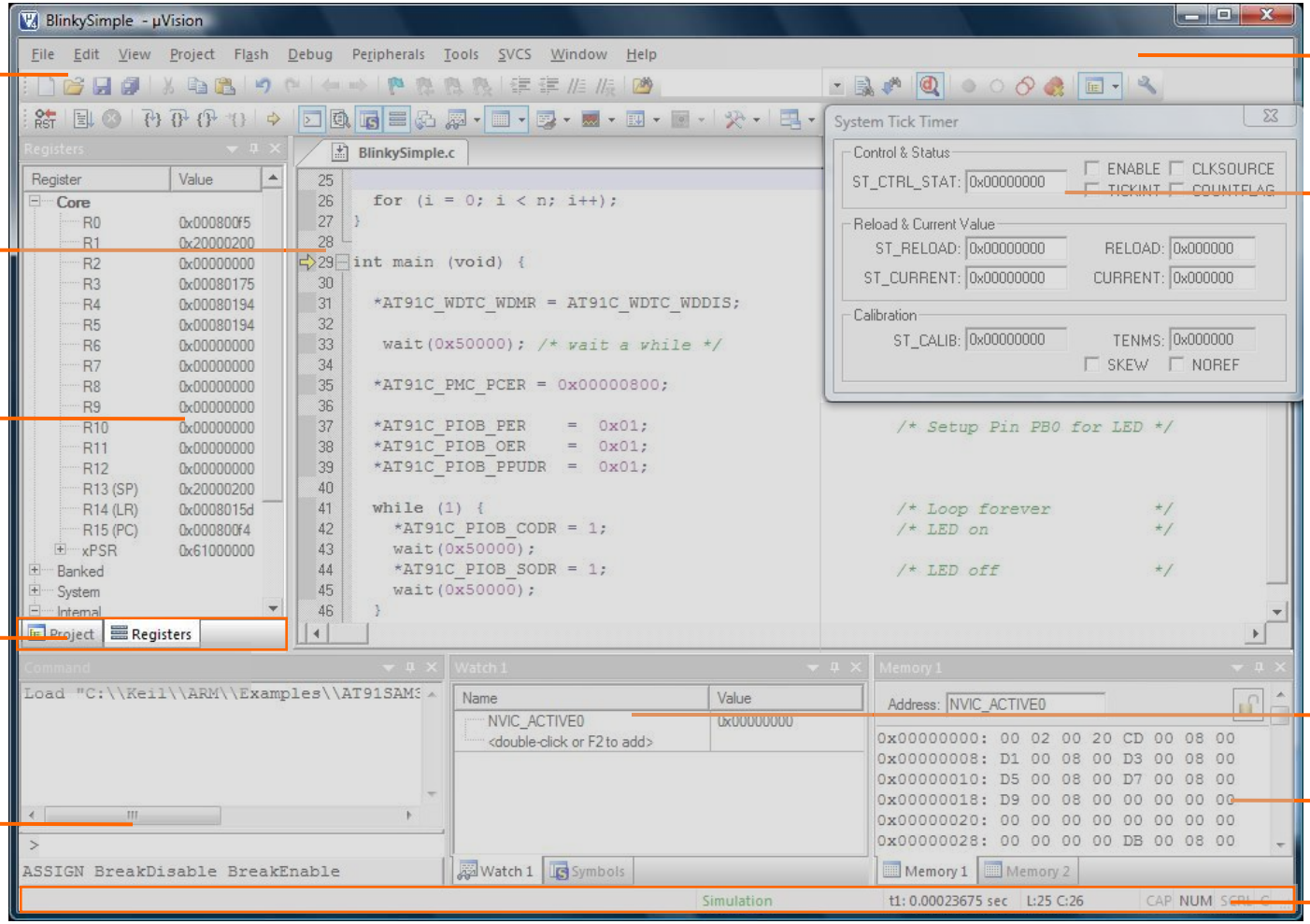


Switch to Debug Mode

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

- Switch µVision into Debug mode
  - Go to **Debug — Start / Stop Debug Session**
  - The **Debug** Button will be selected in debug mode

µVision Debug Mode



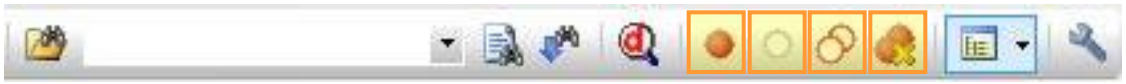
µVision Debug Toolbar

- 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

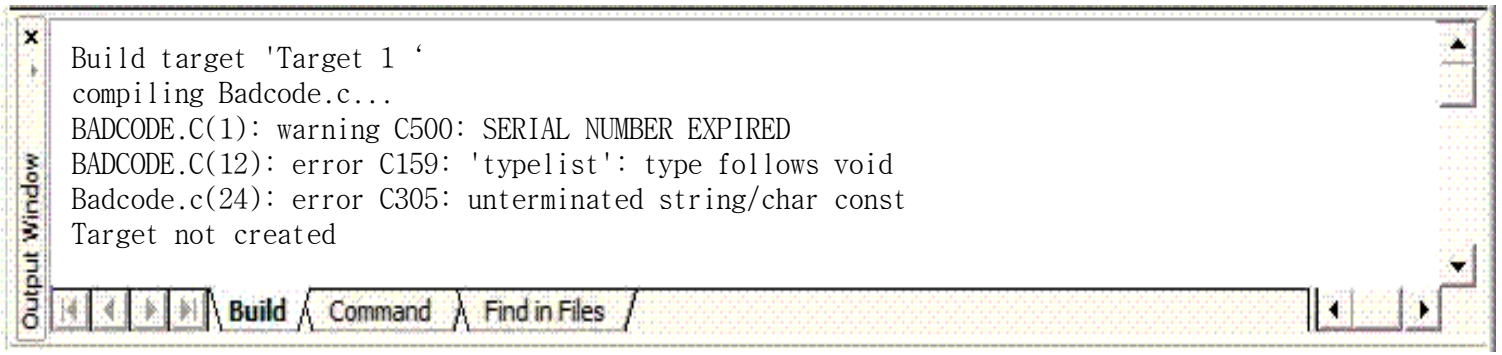
- Example Objectives:
  - Build the project
  - Use the editor, compiler to correct the errors

- 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 %uWn", i);

  fellow += i;
  printf ("Fellow = %ldWn, fellow);
  printf ("End of loopWn")
  }
}
```

Finding Compiler Errors

- Build the program, view the output window



- 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'

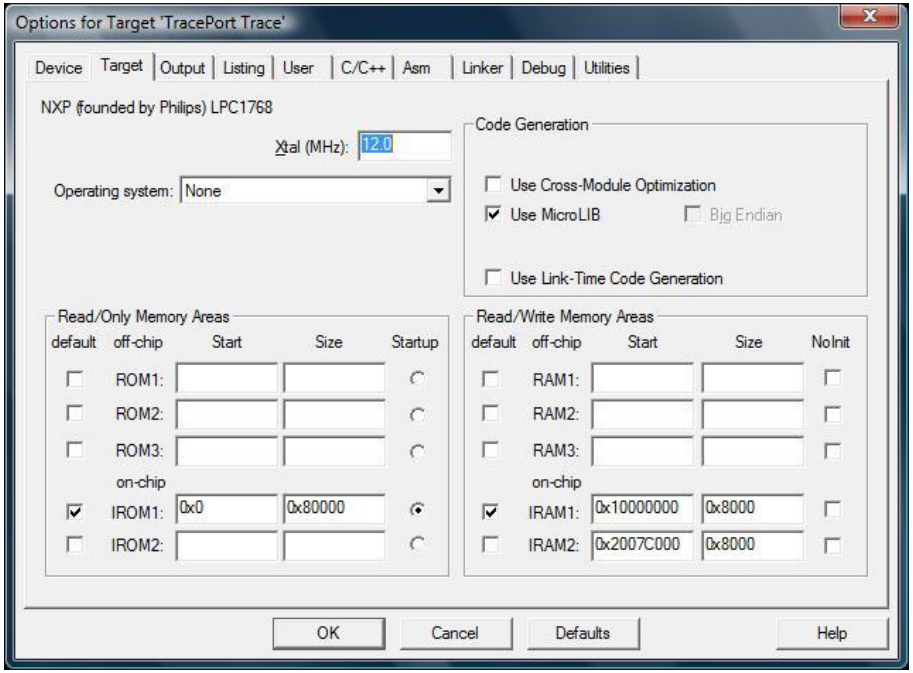
µVision - Finish Fixing the Errors

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

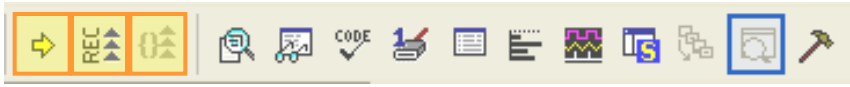
Memory – Target Tab for ARM

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

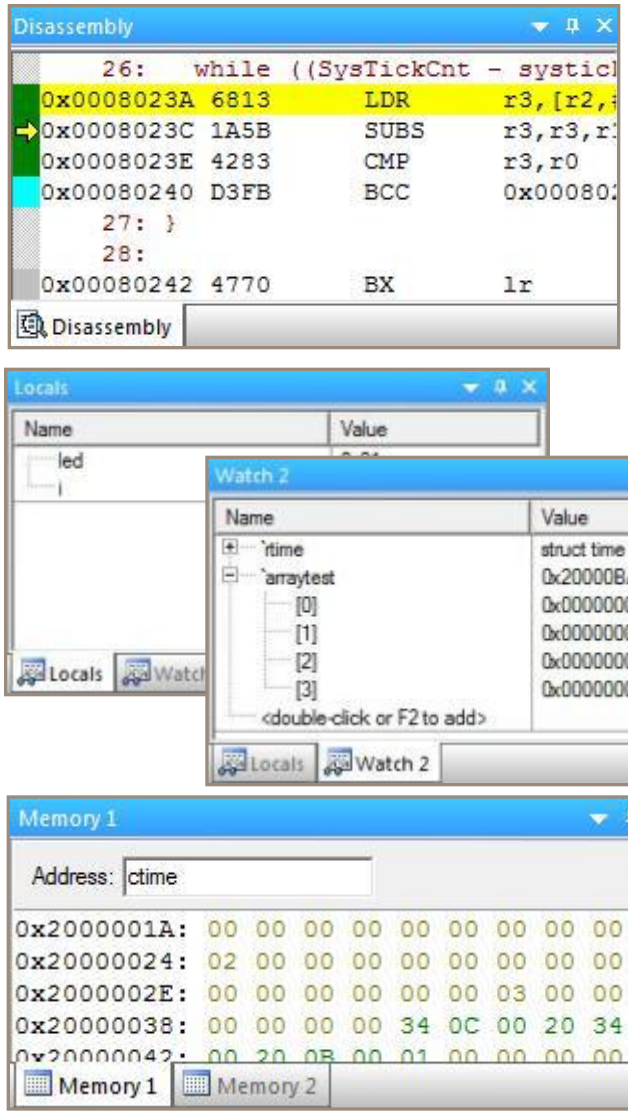
Get to Know Your Code

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

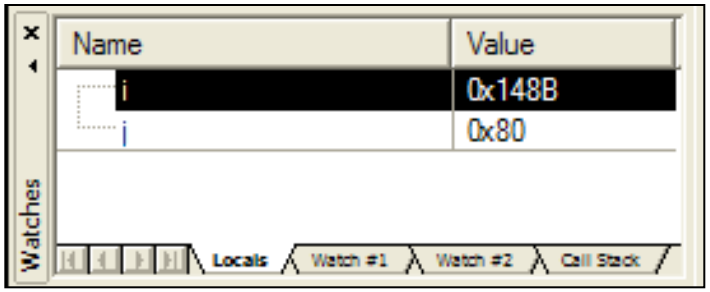
Debug Tools

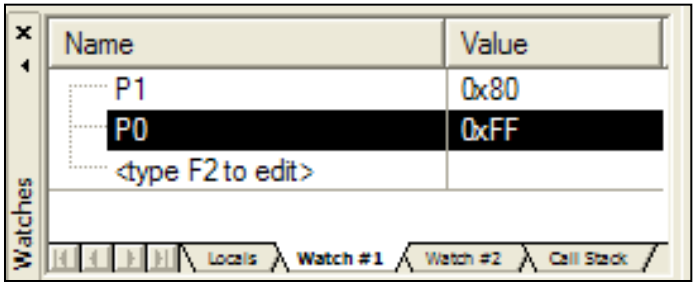
- Disassembly window
  - Shows the target program as mixed source and assembly code
- Watch window
  - Multiple ways to add variables
    - Use F2 key
    - Hover over variable and right click
    - Select Add to Watch, type watch point command
- Memory window
  - Displays the contents of memory from entered start address
  - Updated periodically



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

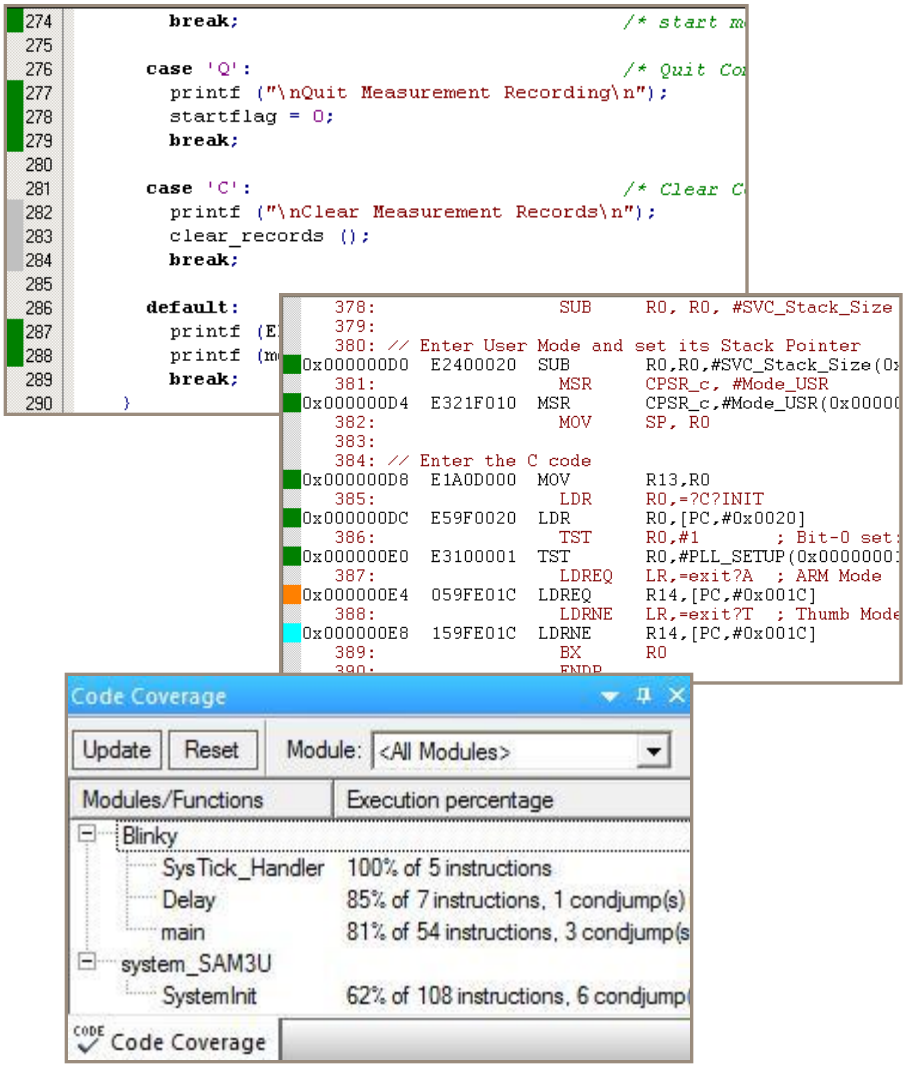
- View - Serial Window –
  - View simulated data sent from chip
  - Type to transmit simulated data to chip
- Right click for a serial window local menu
  - Switch between ASCI /Hex Mode
  - Clear the Serial Window

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

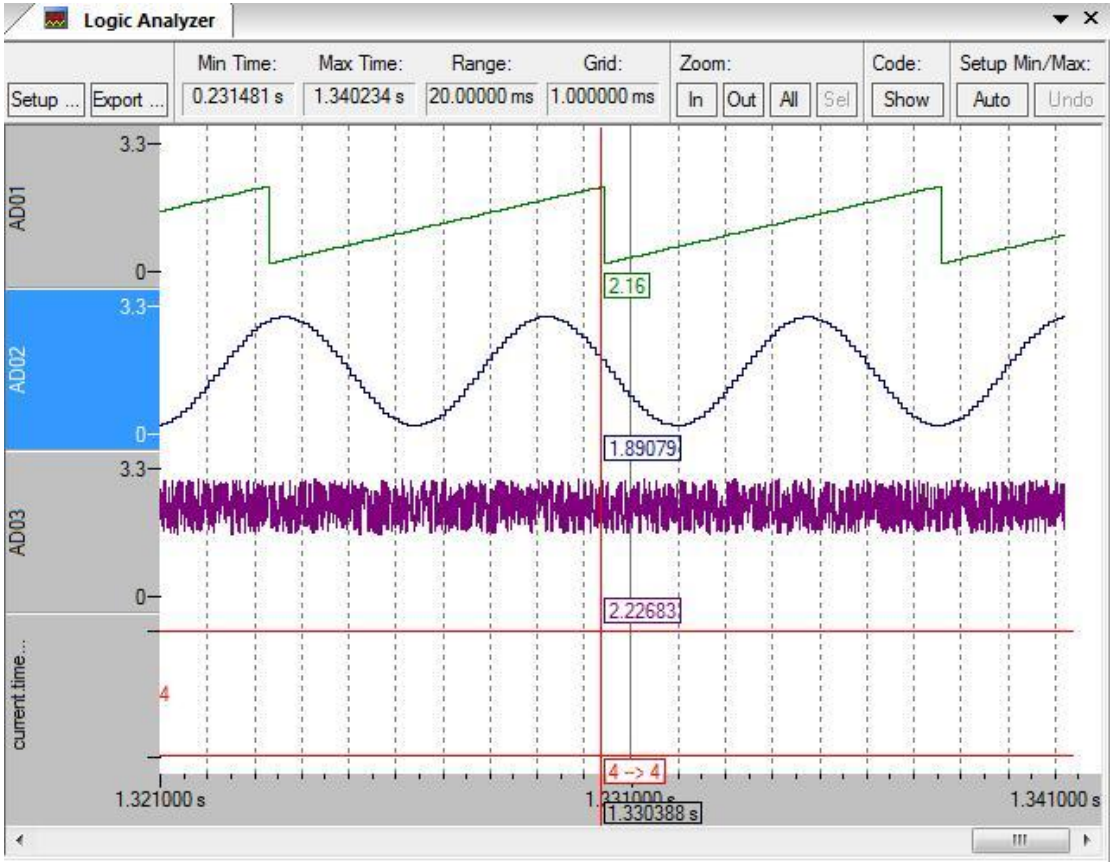
µVision Code Coverage

- 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



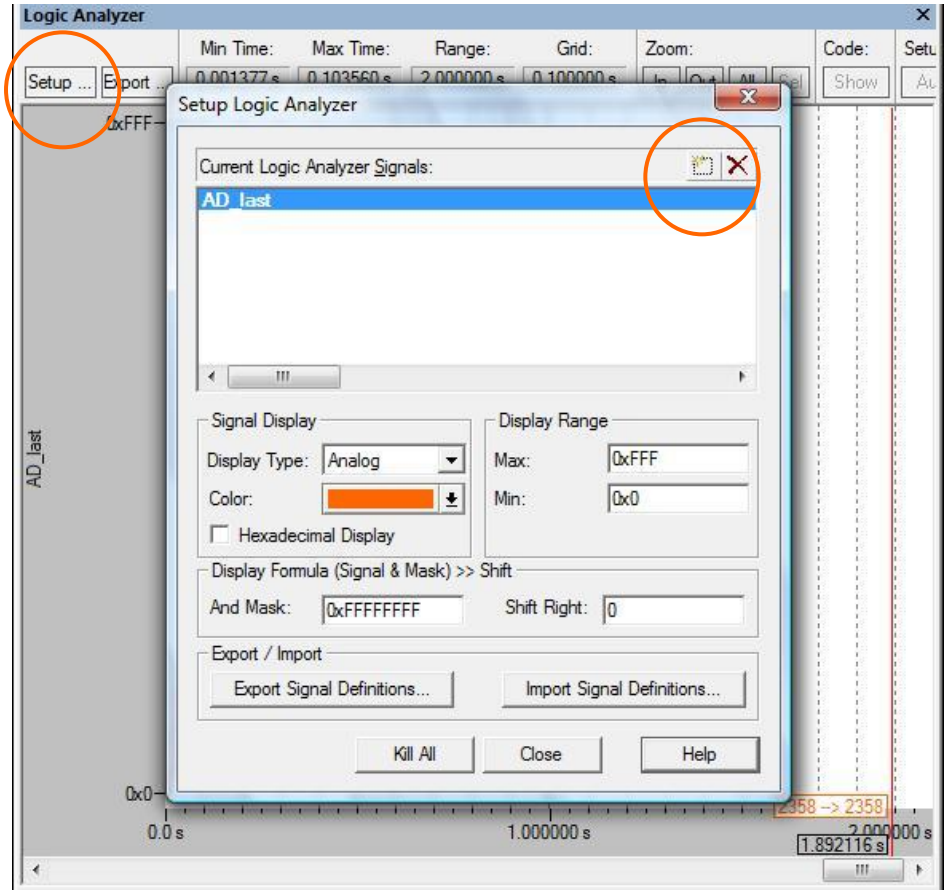
µVision Logic Analyzer

- 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

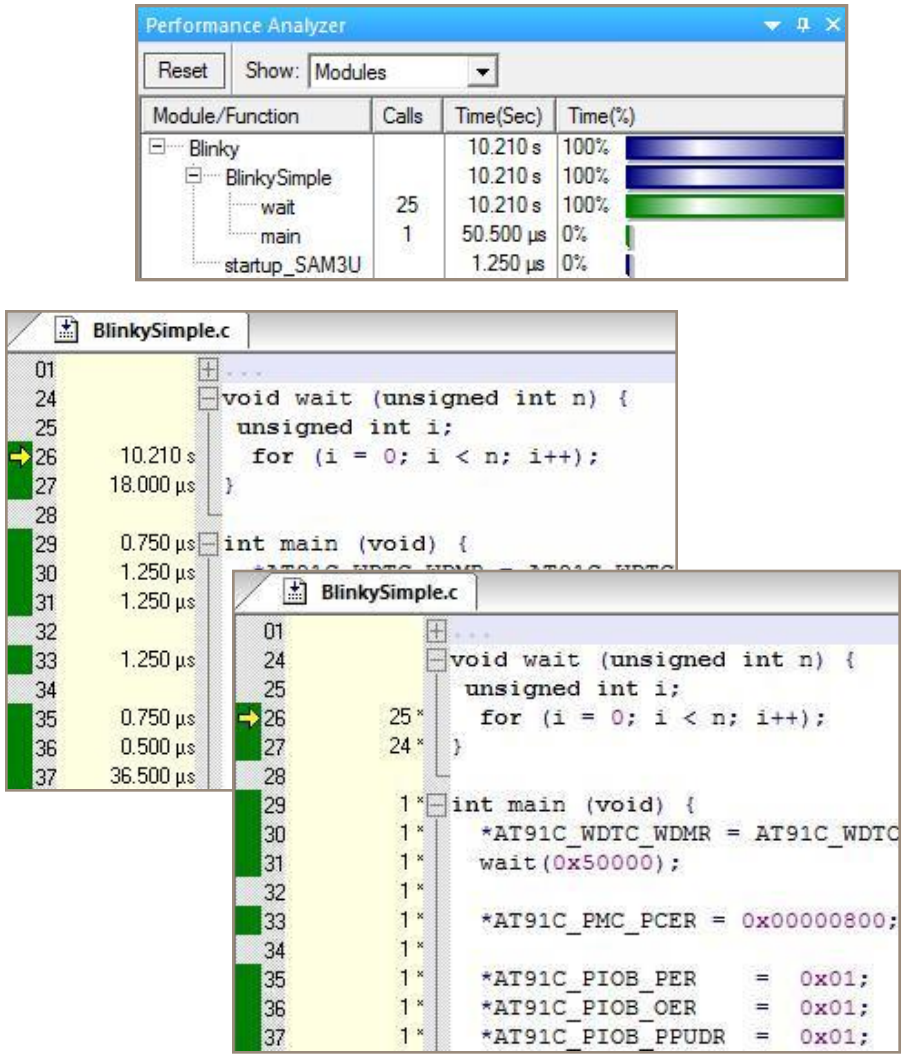
- Presently the LA is set up to monitor the pseudo-analog input to AD0 generated by the Signal function we just reviewed.
- The LA may or may not be displaying some of the step functions.
- Run the program for several seconds and then stop the program.
- With a left mouse click, select one of the signals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



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 µVision 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

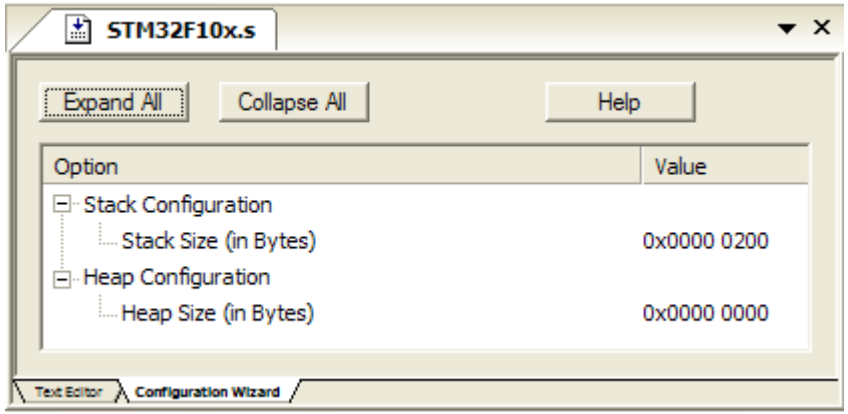
- Several variants are furnished by device type
  - STARTUP.S - ARM startup code
- Starting a new project will usually initiate a dialog asking if you would like a copy placed in your project
- Always start with a fresh startup file placed in your project file

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
```

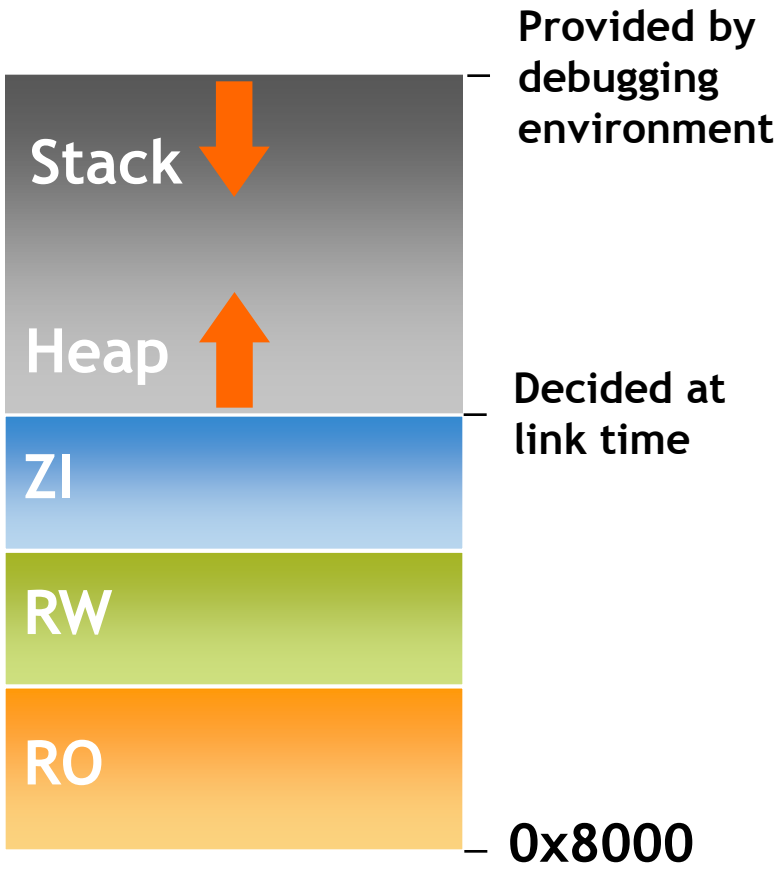


Scatter Loading File

- Absolute Control over Object Placement

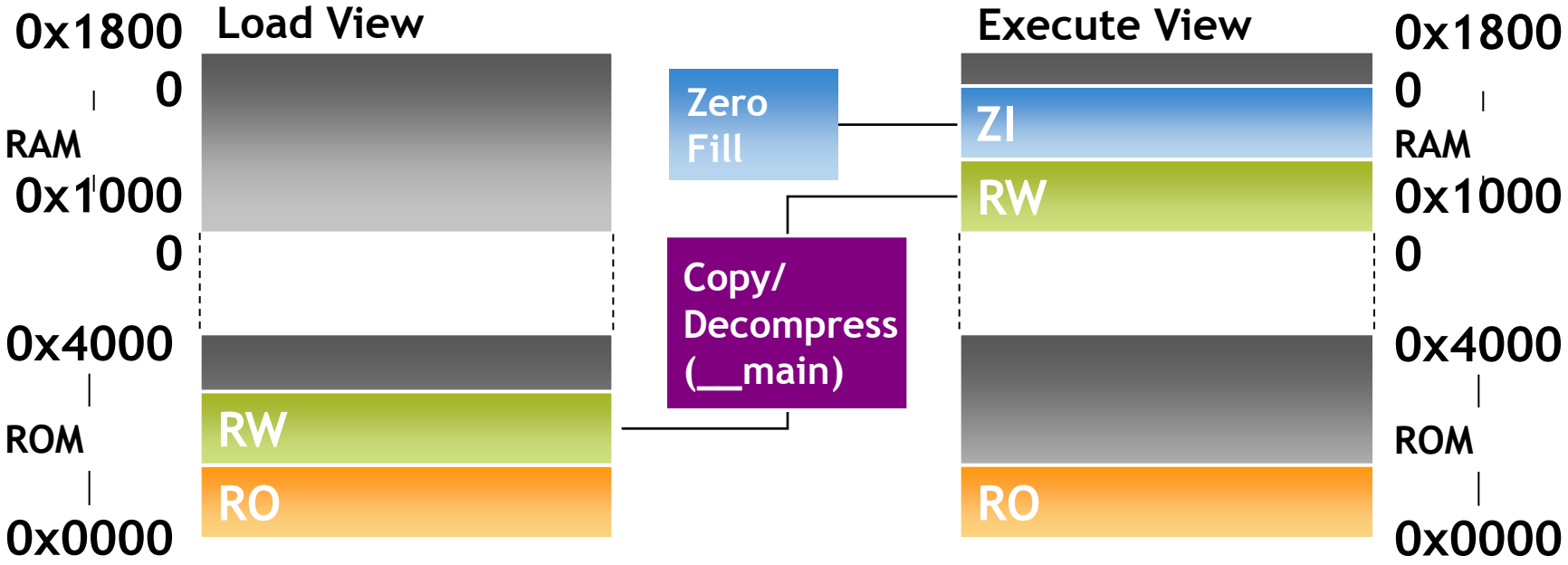
Default Memory Map

- 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:

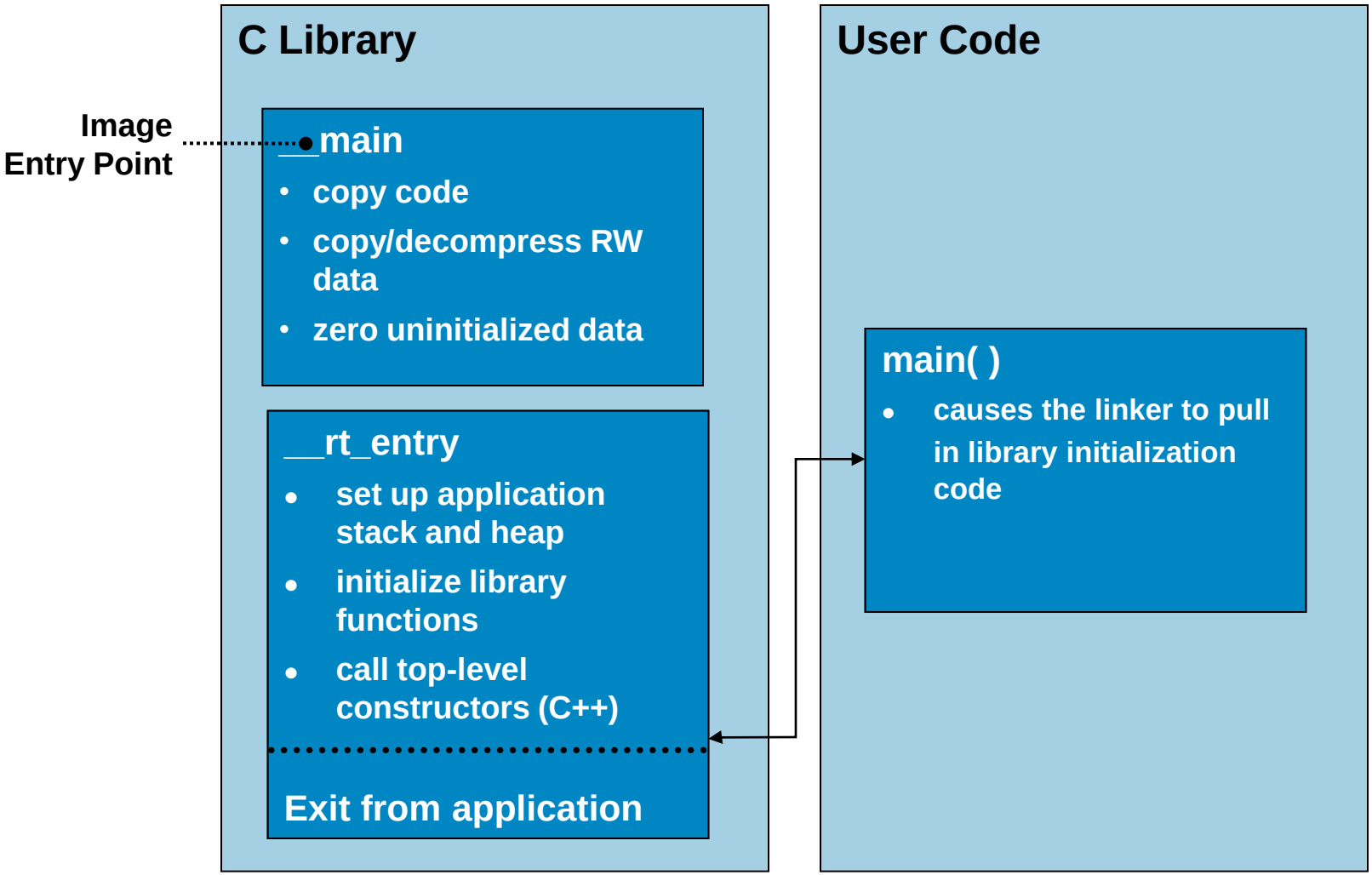
```
__attribute__ ((zero_init)) struct      {
     volatile unsigned reg1;            /* timer control */
     volatile unsigned reg2;            /* timer value   */
} timer_reg;
```

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

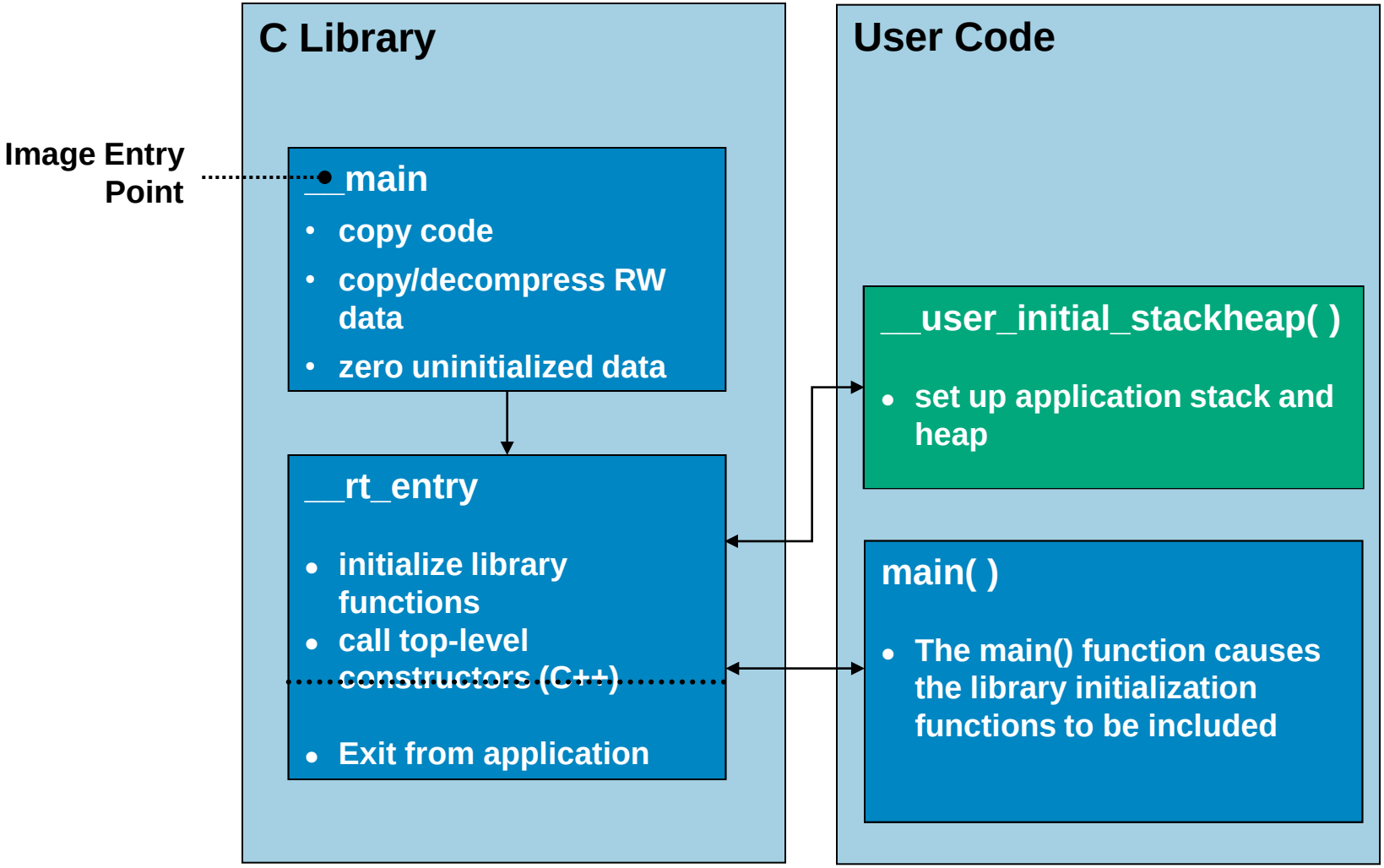
```
LOAD_FLASH 0x24000000 0x04000000        {
    TIMER 0x40000000 UNINIT             {
    timer_reg.o (+ZI)
    }
}
```

What does __main do?

Application Startup



Stack and Heap Initialization

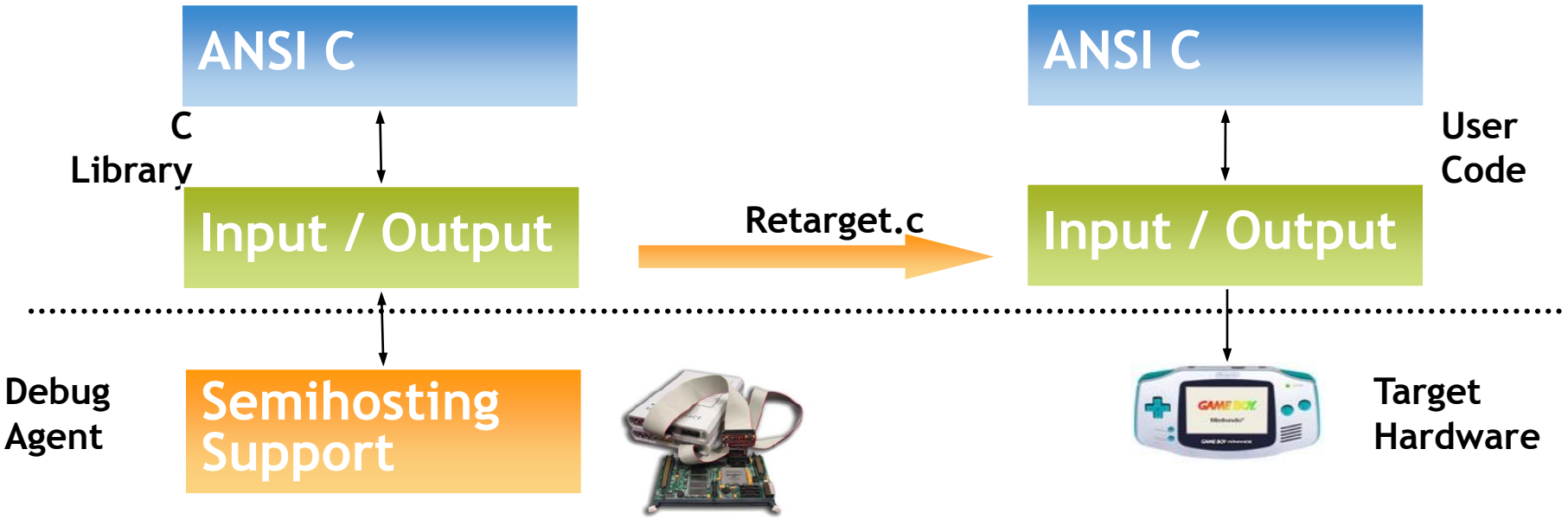


Retarget.c File

- 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 $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

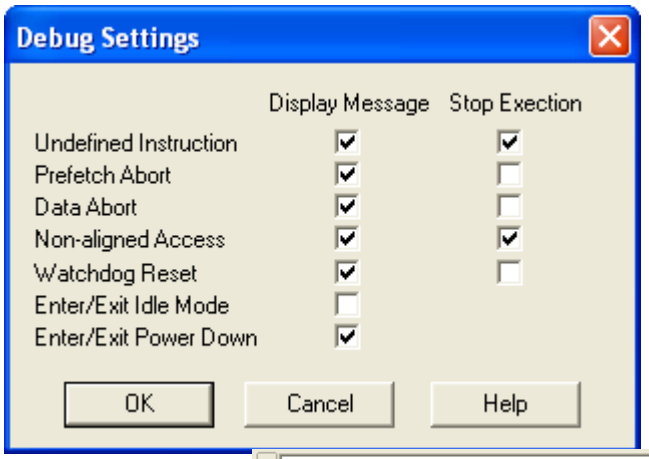
Unused Section Elimination/Entry Points

- 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:

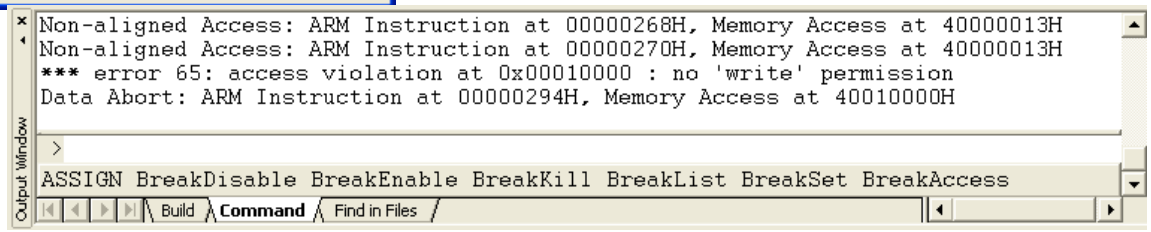
```
armlink obj1.o obj2.o --scatter scatter.scat
    --info unused --entry startup -o prog.axf
```

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



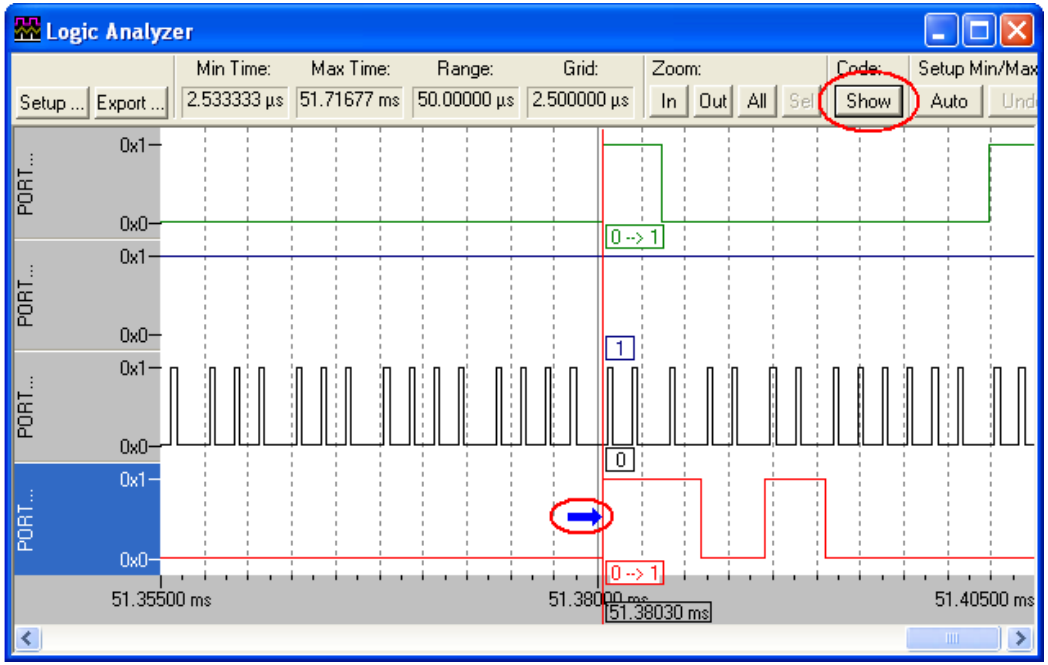

```
// Un-aligned memory access create wrong values in HW
// but are hard to detect during JTAG Debugging
typedef struct  {                  struct  {
 COLOR           Color;              INPUT_EVENT   Event;
 unsigned short  Level;              unsigned char
     Buffer[10];
} LEVEL;               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

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

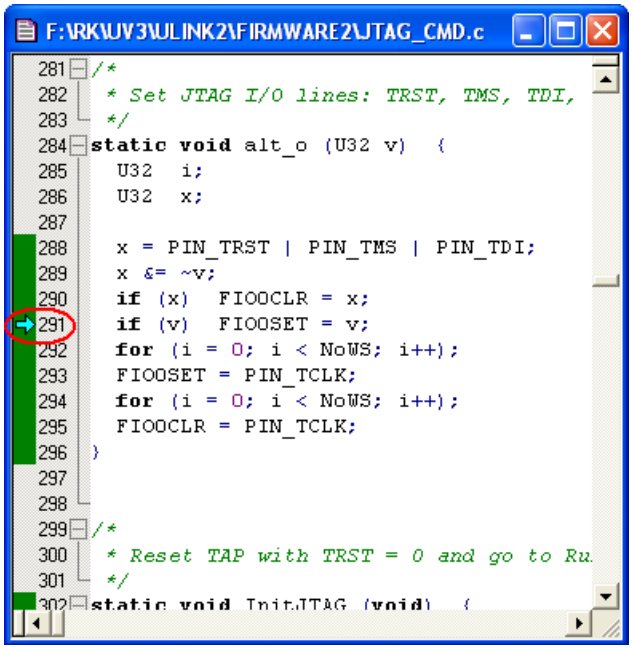


http://www.keil.com/download/docs/323.asp

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.

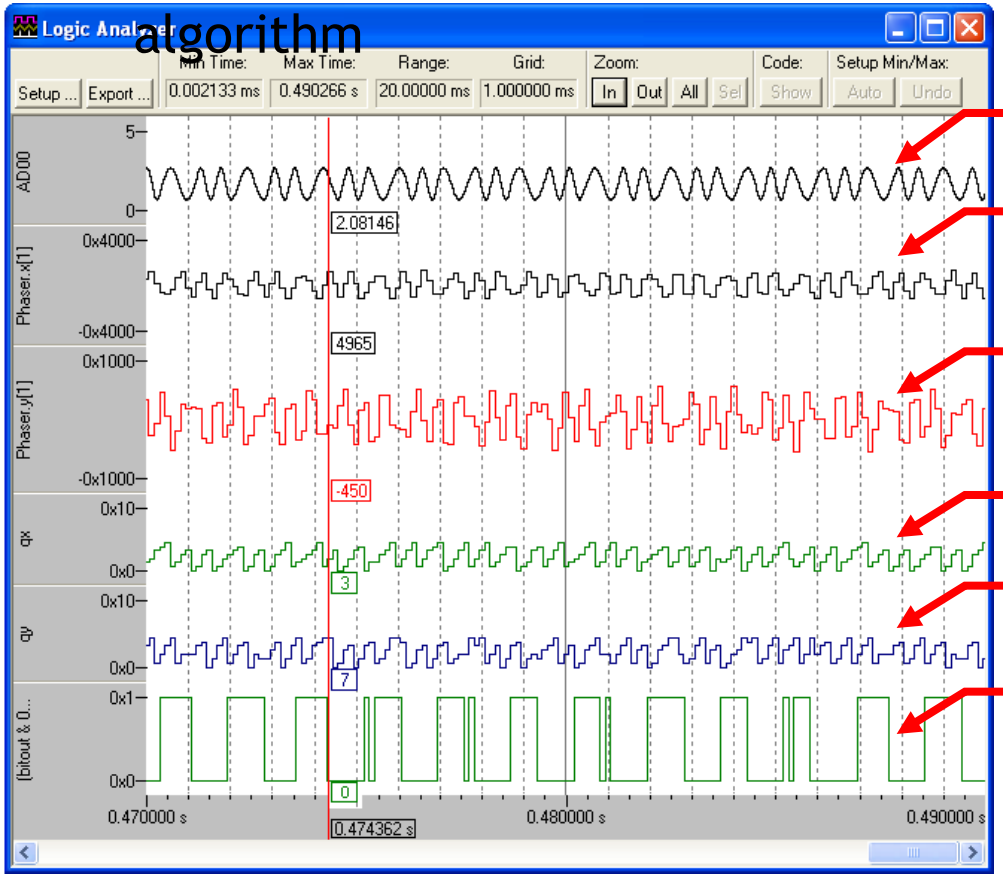




http://www.keil.com/download/docs/322.asp

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

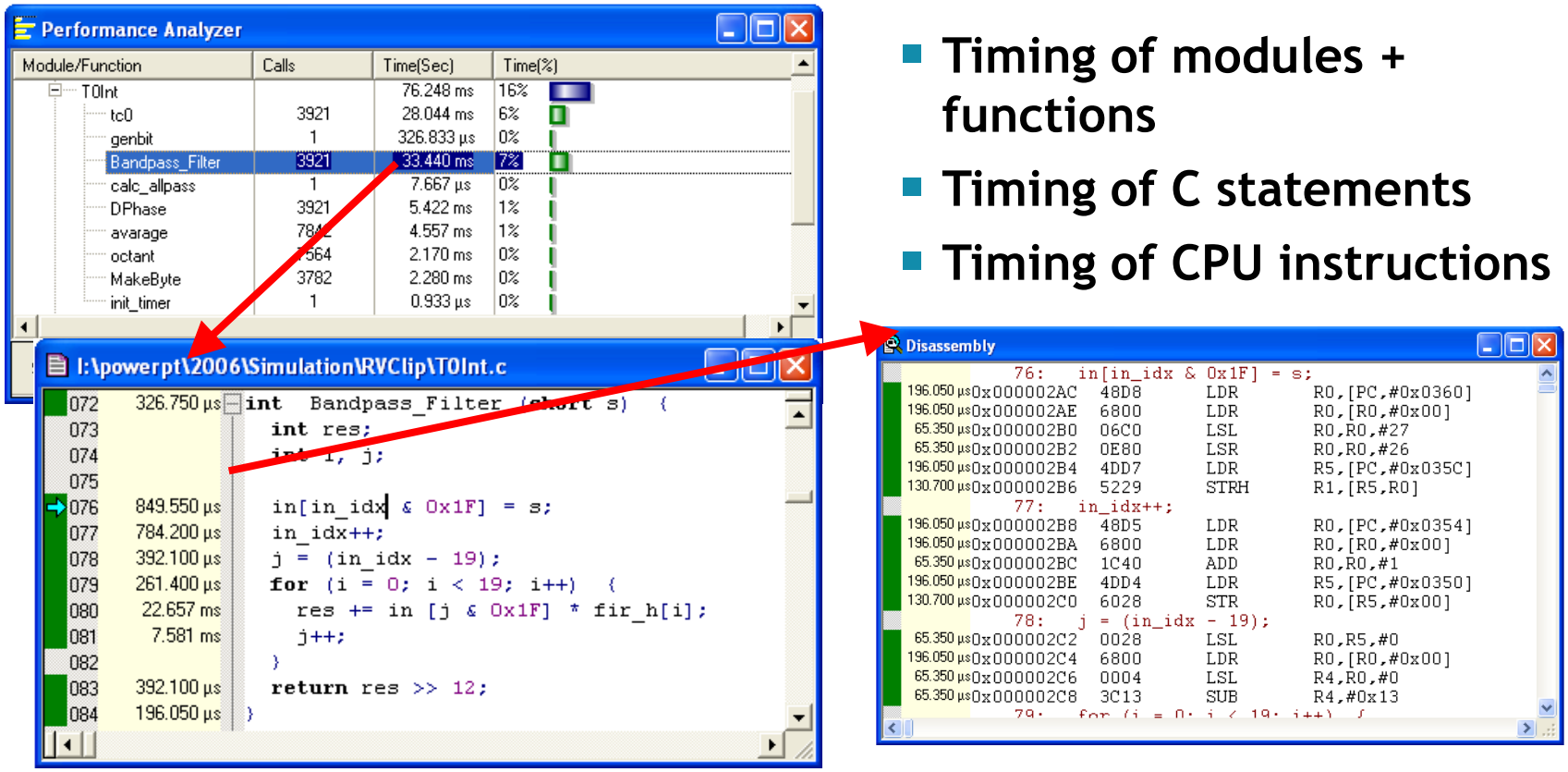
PLL quadrant of de-phaser input

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

Event Counters

- Real-time values of event counters

Exception Trace

- Statistical information about program exceptions