



Keil Elektronik GmbH  
(Germany)



Keil Software, Inc.  
(USA)



Distributor Network

## Hans Schneeberger

### Embedded Development Tools For ...

- Consumer Electronics
- Industrial Control
- Networking and Office Automation
- Automotive and Space Exploration

### Chip Support For ...

- 8051 and Extended 8051/251 Microcontrollers
- ARM based Microcontrollers
- 166 and ST10 Microcontrollers

### Representatives In ...

- 40 Countries World-Wide
- Munich Germany
- Dallas, TX USA



# Keil Development Tools for ARM



## Third Party Utilities

**Software Version Control System**

**PC-LINT**  
intensive syntax check  
with MISRA support

**I-Logix Rhapsody UML 2.0**

**GUI Library**  
with Display Simulation

**CAN Connector**  
for Device Simulation

**MATLAB/Simulink**  
Simulation Interface

## Integrated Development Environment

**µVision3 Editor**  
Project Management & Make Utility

**C Compiler**

**Macro Assembler**

**Libraries**

**Linker / Locator**

**µVision3 Debugger**

**CPU & Peripheral Simulation**

**Keil ULINK JTAG Adapter**

## Advanced RTX Real-Time Operating System

**Real-Time Kernel**

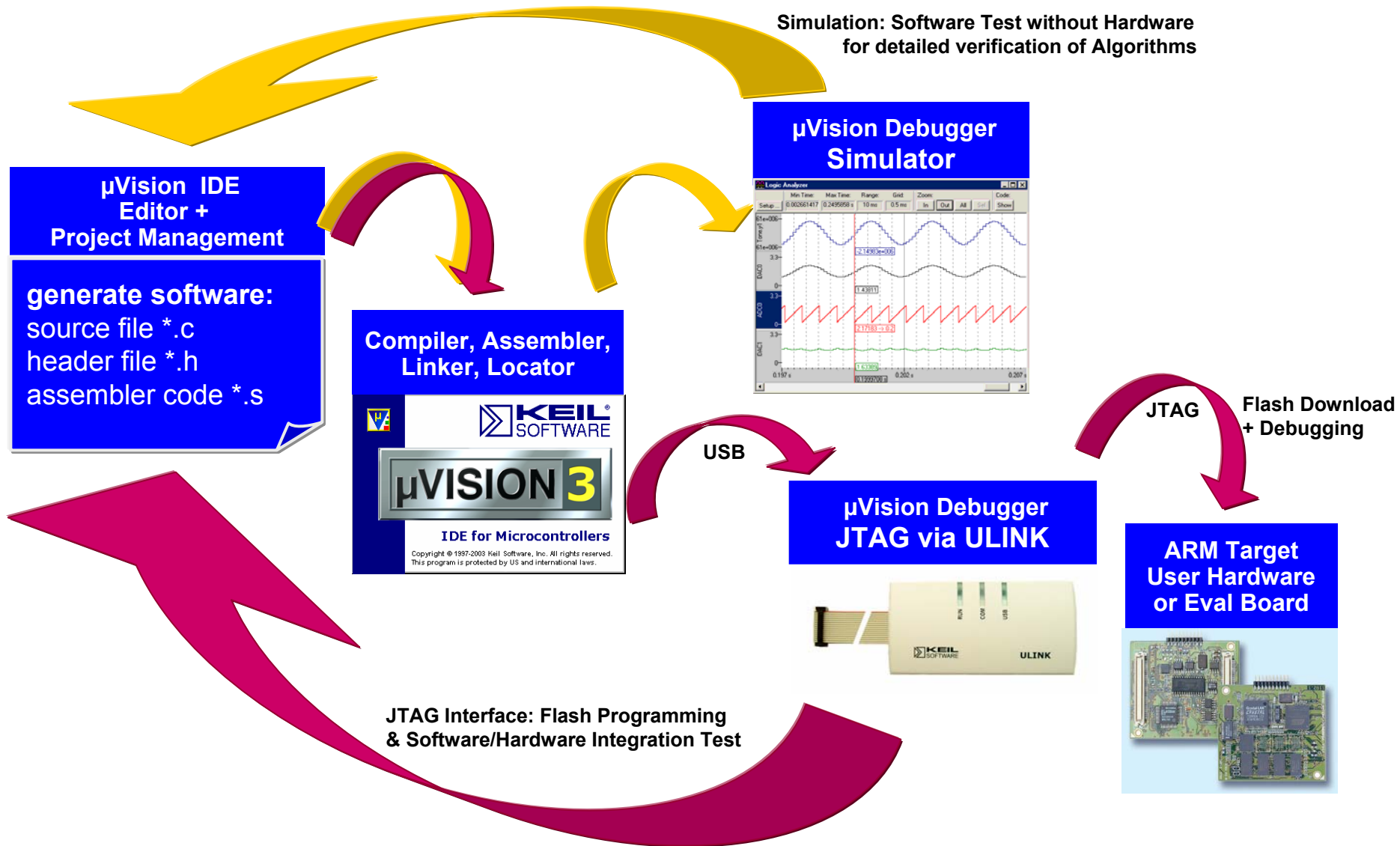
**TCP/IP Network Suite**  
TCP, UDP, PPP, and SLIP Interface  
HTTP Server with CGI Scripting  
Telnet Server, TFTP Server  
SMTP Client, DNS Resolver

**Flash File System**

**USB Device Interface**

**CAN Interface**

# Software Design Flow



# Simulation & JTAG Debugging

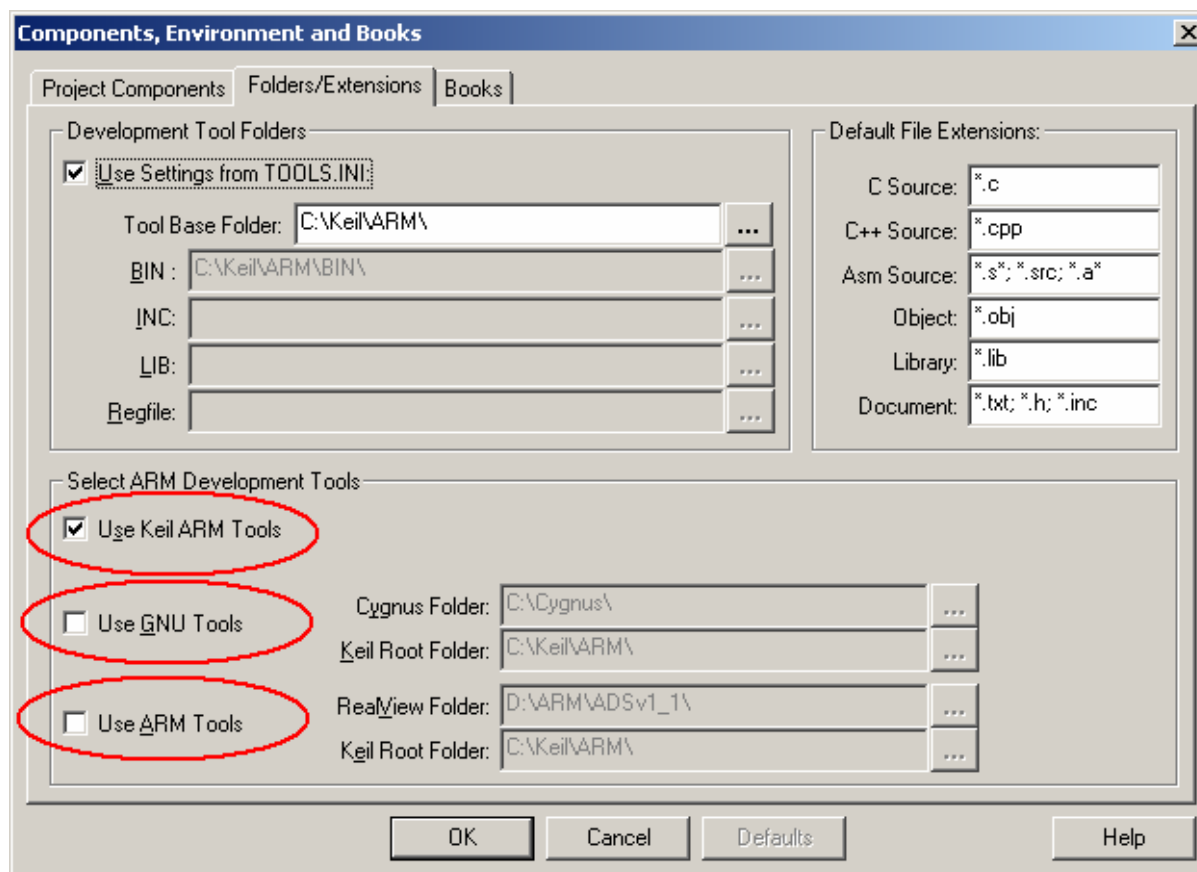


Simulation allows software testing with no hardware. Simulation provides quick test cycles, improves product quality, and enables reproducible regression tests.

| Feature                   | Simulator                                                                                                   | JTAG Debugger                                                                                                           | Explanation                                                                               |
|---------------------------|-------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Breakpoints               | <b>Unlimited Breakpoint capabilities</b> including access, conditional and execution breakpoints            | <b>Embedded ICE restrictions:</b> only two execution/access breakpoints in Flash. Unlimited software breakpoints in RAM | Single-chip design has only Flash ROM for program code                                    |
| Peripherals               | Simulated peripherals <b>fully synchronized</b> with program execution                                      | Peripherals <b>may not stop</b> when a Breakpoint is reached                                                            | Simulation allows detailed analysis of complex algorithms that interface to peripherals   |
| Power down Idle Mode      | Power down and Idle mode <b>fully simulated</b>                                                             | Embedded ICE <b>not available</b> in power down / idle mode                                                             | Testing of situations that stop or even destroy real hardware is possible via simulation. |
| Trace and Timing Analysis | <b>Full timing &amp; execution analysis</b> with Code Coverage, Trace, Timing Profile, Logic Analyzer       | Extended test features <b>via ETM</b> , but impossible via JTAG                                                         | ETM requires additional I/O pins; not available on some devices                           |
| Input Signal Patterns     | Input signal generation <b>with script language</b> ; synchronized with program execution (single stepping) | Input signals <b>provided by hardware</b> and external test equipment                                                   | Simulation for verification of specifications; JTAG debugging for real-world testing      |
| Real-Time Debugging       | Simulation is <b>timing accurate</b> and in correct relation to peripherals                                 | JTAG Debugging runs real-time <b>at full CPU speed</b> .                                                                | Simulation is 4 - 10 times slower than a high-speed ARM device, but timing accurate.      |

# Choice of ARM Compilers

The Keil  $\mu$ Vision Debugger for ARM supports: **GNU ARM toolchain**, **ARM ADS/RealView**, and the **Keil CARM Compiler**.  $\mu$ Vision makes it easy to operate all these tools with integrated Make and Tool Configuration Dialogs.



# μVision Device Simulation



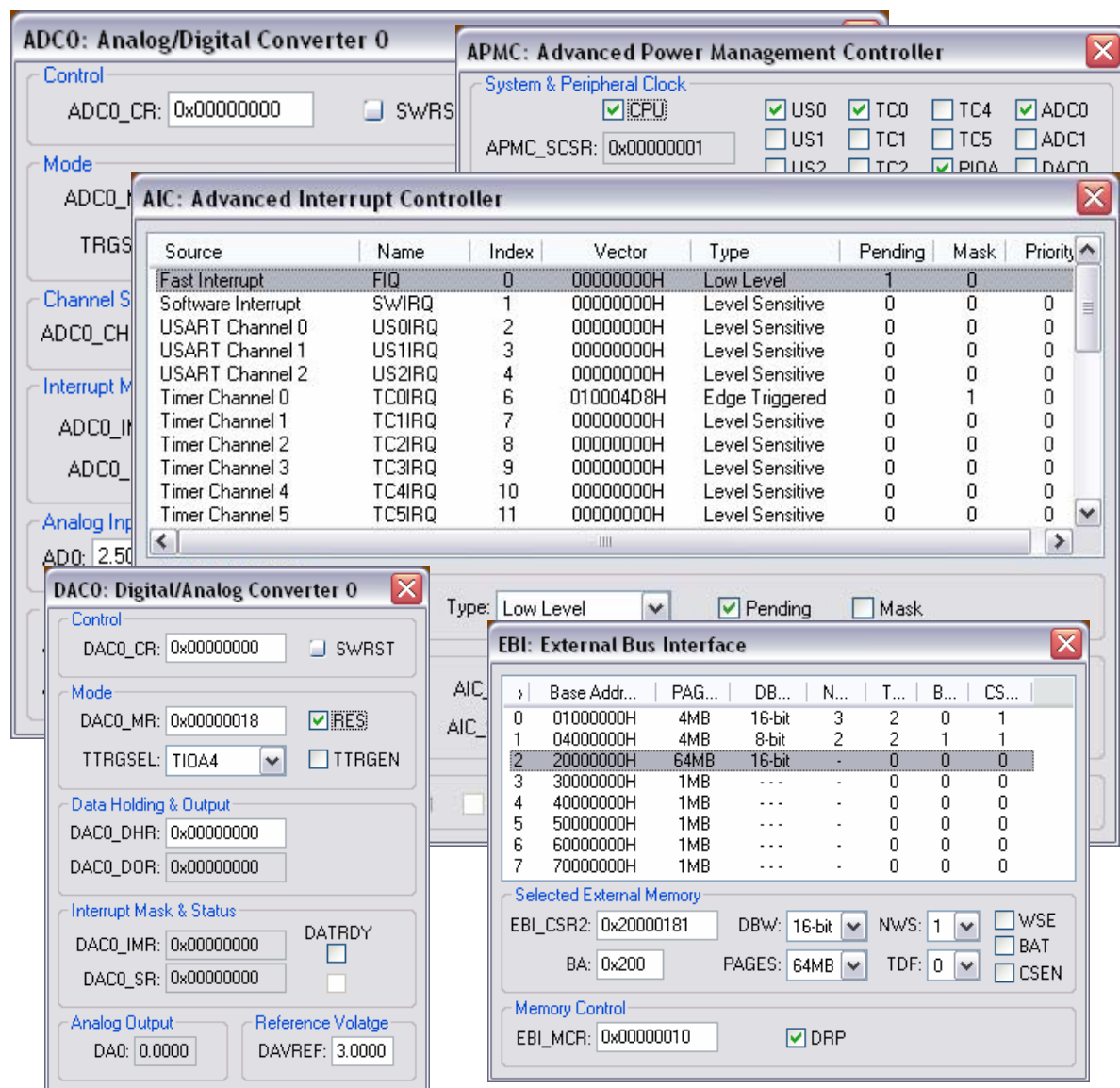
## Complete Simulation

- ▶ Many Standard Devices
- ▶ Covers complex peripherals:  
ADC, DAC, EBI  
Timers, and more.

## Peripheral Dialogs

- ▶ Simulation
- ▶ JTAG Debugging

## Expandable with User DLL's



# µVision Code Coverage



## Execution Statistics

- ▶ Always Active for Complete Project
- ▶ Coverage Statistic at Module, C Source, and Assembly Level
- ▶ Instruction Status:
  - not executed (grey)
  - fully executed (green)
  - skipped (orange)
  - always taken (cyan)
- ▶ Multi-Session Coverage with Save / Restore

| Modules/Functions | Execution percentage                                      |
|-------------------|-----------------------------------------------------------|
| init_task         | 100% of 49 instructions                                   |
| clock_task        | 52% of 50 instructions                                    |
| get_escape        | 100% of 13 instructions, 1 condjump(s) not fully executed |
| command_task      | 94% of 209 instructions, 1 condjump(s) not fully executed |
| measure_task      | 100% of 49 instructions                                   |
| interval_task     | 95% of 45 instructions, 1 condjump(s) not fully executed  |
| save_measurements | 69% of 39 instructions                                    |
| read_index        | 94% of 50 instructions, 3 condjump(s) not fully executed  |
| clear_records     | 100% of 15 instructions                                   |
| ?C?INIT           | 74% of 31 instructions                                    |

C:\Keil\ARM\ARTX\Examples\Measure\Measure.c

```
274      break;
275
276      case 'Q':
277          printf ("\n
278          startflag =
279          break;
280
281      case 'C':
282          printf ("\n
283          clear_recor
284          break;
285
286      default:
287          printf (ERR
288          printf (men
289          break;
290      }
```

Disassembly

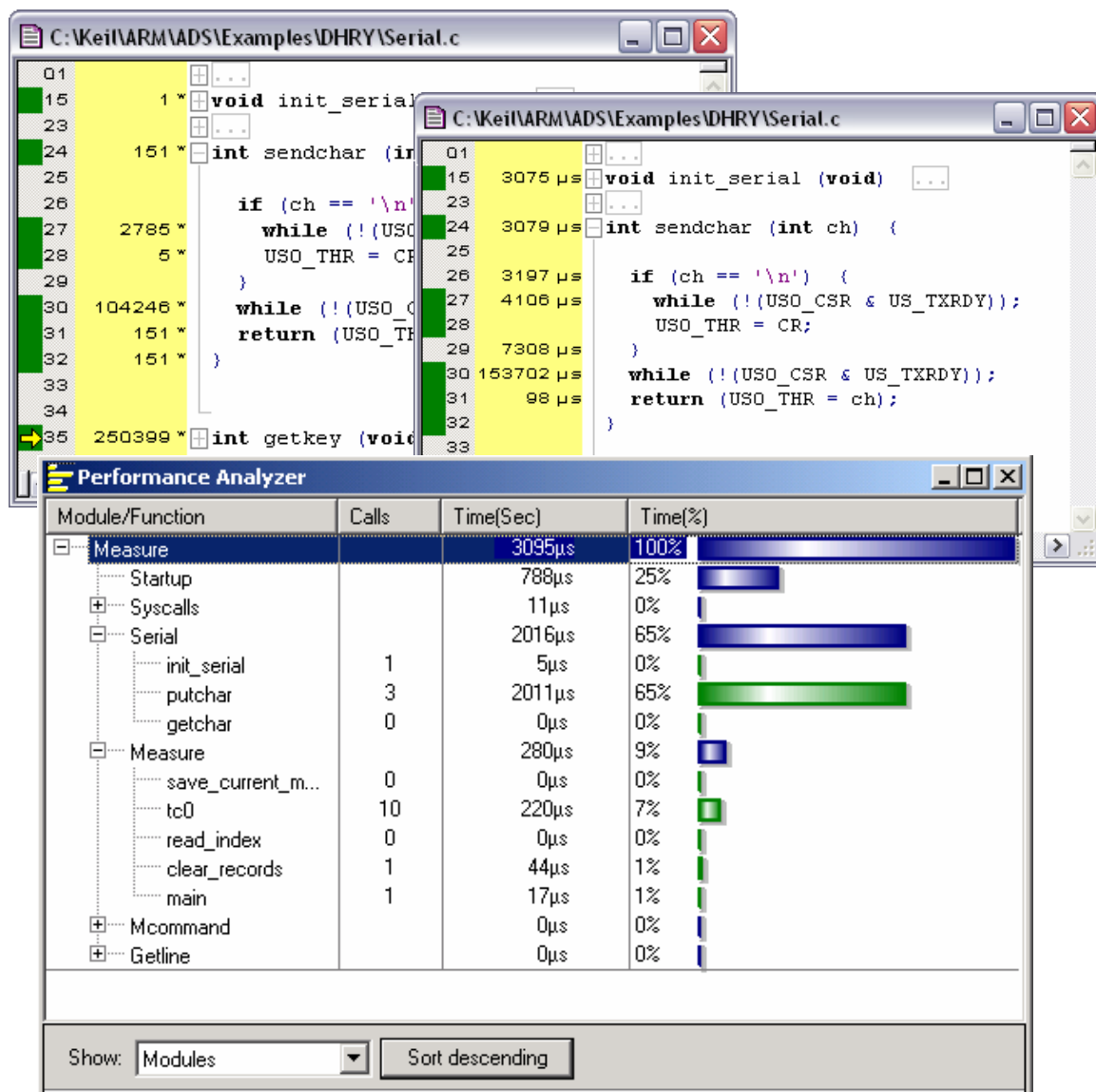
```
378:          SUB      R0, R0, #SVC_Stack_Size
379:
380: // Enter User Mode and set its Stack Pointer
0x000000D0 E2400020 SUB      R0,R0,#SVC_Stack_Size(0:
381:          MSR      CPSR_c, #Mode_USR
0x000000D4 E321F010 MSR      CPSR_c,#Mode_USR(0x000000
382:          MOV      SP, R0
383:
384: // Enter the C code
0x000000D8 E1A0D000 MOV      R13,R0
385:          LDR      R0,=?C?INIT
0x000000DC E59F0020 LDR      R0,[PC,#0x0020]
386:          TST      R0,#1 ; Bit-0 set:
0x000000E0 E3100001 TST      R0,#PLL_SETUP(0x00000000:
387:          LDREQ    LR,=exit?A ; ARM Mode
0x000000E4 059FE01C LDREQ    R14,[PC,#0x001C]
388:          LDRNE    LR,=exit?T ; Thumb Mode
0x000000E8 159FE01C LDRNE    R14,[PC,#0x001C]
389:          BX       R0
390:          ENDP
```

# μVision Execution Profiling



## Timing Statistics

- ▶ Always Active for Complete Project
- ▶ Shows how many times each line of code has executed
- ▶ Shows Total Execution Time
- ▶ Complete Statistics in Performance Analyzer Window



## Timing Analysis

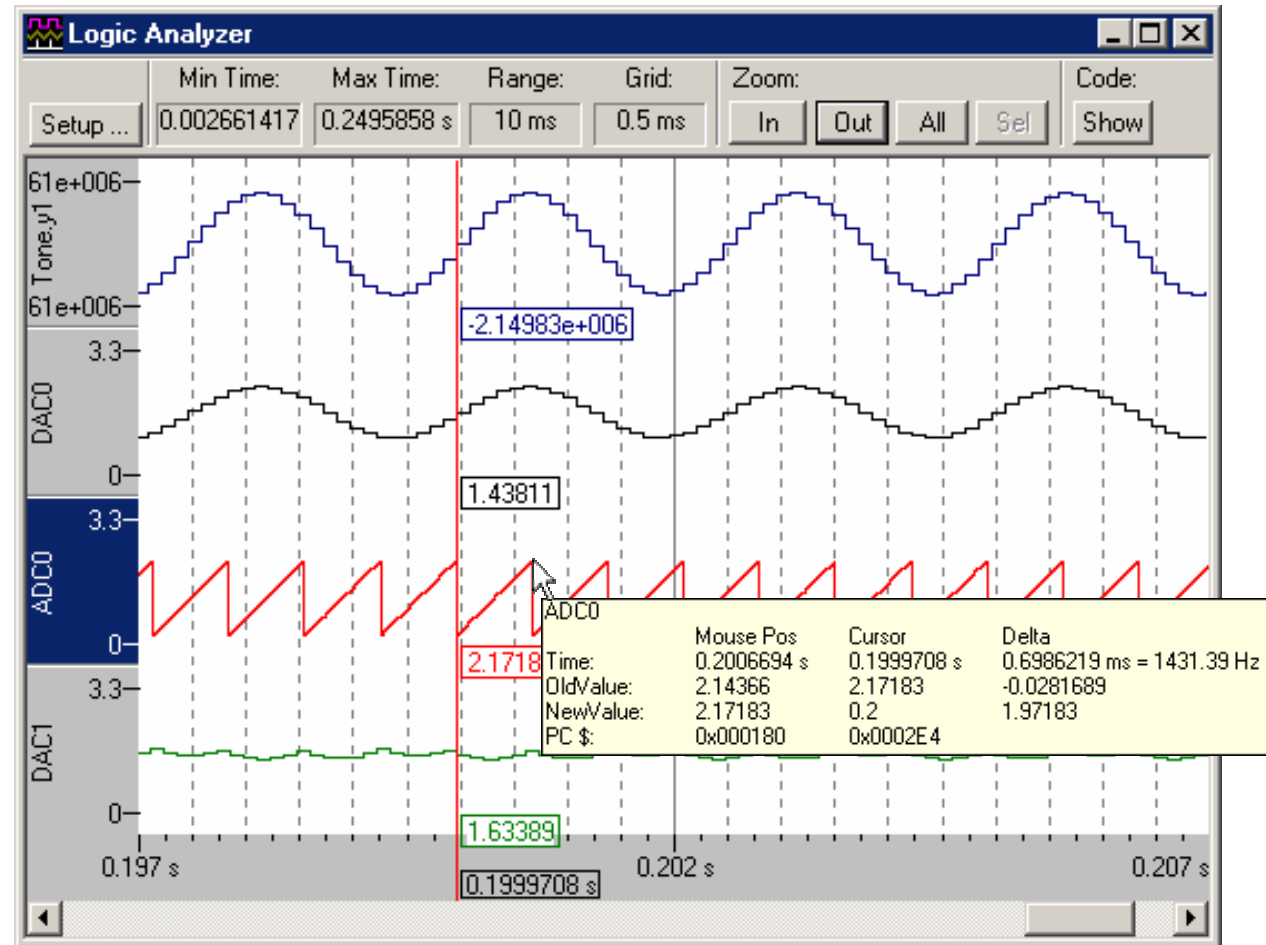
- ▶ Analog & Digital I/O Pins and Signals
- ▶ Internal Variables

## Exact Timing

- ▶ Using Cursor Line
- ▶ Tool-Tip Delta Information

## Display Styles

- ▶ Analog
- ▶ Bit
- ▶ State Change



# Debug & Signal Functions



## Expand Command Set

- ▶ C Macro Language
- ▶ Trigger by Breakpoints
- ▶ Output Analysis
- ▶ Input Pattern Generation

## Signal Functions

- ▶ Background Execution
- ▶ Time or Access triggered execution delays
- ▶ Input Signal Generation

```
// Generate Sine Wave Signal on A/D Channel 0

signal void ADC0_Sine (void) {
    float volts;           // peak-to-peak voltage
    float frequency;       // output frequency in Hz
    float offset;          // voltage offset
    float duration;        // duration in Seconds
    float val;
    long i, end;

    volts      = 1.4;
    offset     = 1.6;
    frequency  = 1800;
    duration   = 0.1;

    printf ("Sine Wave Signal on AD Channel 0.\n");

    end = (duration * 100000);
    for (i = 0 ; i < end; i++) {
        val = __sin (frequency * (((float) STATES) / CLOCK)
                    * 2 * 3.1415926);
        AIN0 = (val * volts) + offset;
        swatch (0.00001);           // in 10 uSec steps
    }
}
```

## Built-In Functions

- ▶ Text Output
- ▶ User Queries
- ▶ Analog Signal Generation

*µVision Simulation Template: A/D Input - Sine Wave Signal*

# Virtual Simulation Registers



Virtual Simulation Registers (VTREG) simulate CPU pins or communication channels. VTREG specify signals for a simulated piece of hardware. The command `DIR VTREG` lists all Virtual Simulation Registers.

| VTREG                               | Description                                                                                                                                                                                                                                       |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PORTx                               | A group of I/O pins for a port on the chip. For example, PORTA refers to all pins of PORTA. These registers allow you to simulate port I/O.                                                                                                       |
| AINx                                | Analog input pin on the chip which is typically an A/D converter input.                                                                                                                                                                           |
| DAx                                 | Analog output pin on the chip which is typically a D/A converter output.                                                                                                                                                                          |
| xxVREF                              | Input for voltage reference pins                                                                                                                                                                                                                  |
| SxIN                                | Input register of serial interface x. SxIN contains 0xFFFF when a value is processed.                                                                                                                                                             |
| SxOUT                               | Output register of serial interface x.                                                                                                                                                                                                            |
| SxTIME                              | Enable baudrate timing. When 1, $\mu$ Vision simulates exact transmission timing with the current baudrate. When 0, serial transmission instant with infinite baudrate.                                                                           |
| CLOCK                               | The CPU clock frequency of the simulated device in Hz.                                                                                                                                                                                            |
| XTAL                                | The crystal frequency of the device as defined in the dialog <b>Options – Target</b> .                                                                                                                                                            |
| SSPxx,<br>SPIxx,<br>CANxx,<br>I2Cxx | For communication peripherals such as UART, SSP, SPI, CAN, or I <sup>2</sup> C the simulator does not toggle the physical I/O pins. Instead the transmission is reflected in VTREGs which allows you to write signal functions for automated I/O. |

# VTREGs for CAN I/O Simulation



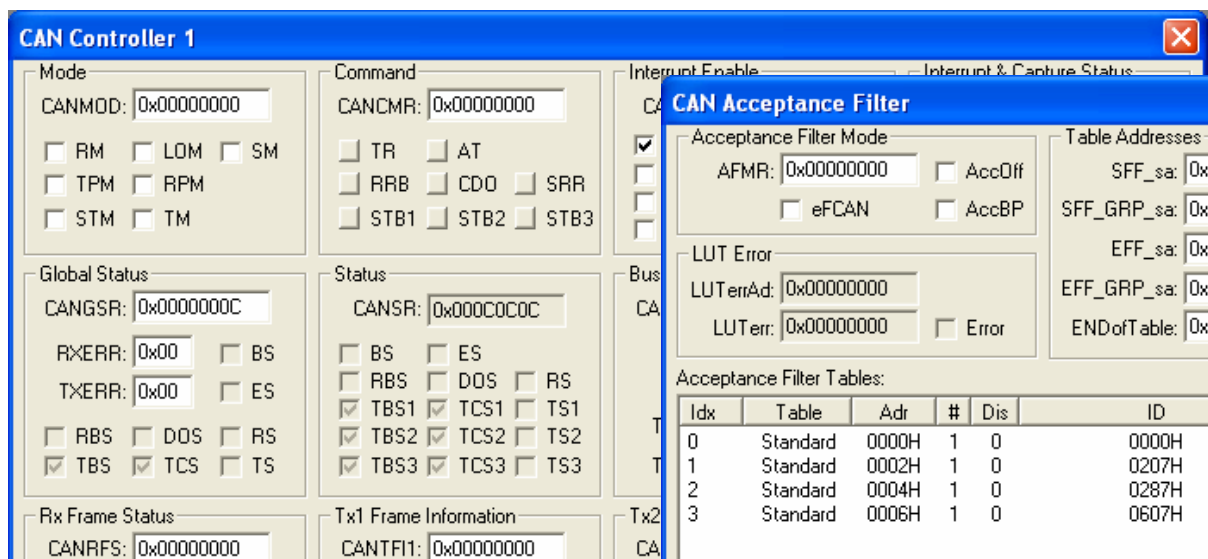
µVision provides VTREGs for controlling of serial communication (CAN, I<sup>2</sup>C, SSP, SPI). CAN I/O is simulated using the following VTREGs.

| VTREG      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CAN_ID     | 11-bit or 29-bit CAN identifier of the currently transferred message. The ID size and message type is specified with the values in CAN_IN or CAN_OUT.                                                                                                                                                                                                                                                                                                                                                                                |
| CAN_L      | Data length of the CAN message. Valid values for CAN_L are 0 – 8.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| CAN_B0 – 7 | Represent maximal 8 data bytes of a CAN message for the current transferred object.                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| CAN_IN     | Set to the following values by user or debug functions to simulate incoming messages:<br>1: receive the current message using a 11-bit identifier.<br>2: receive the current message using a 29-bit identifier.<br>3: request a remote frame from the application program with matching 11-bit identifier.<br>4: request a remote frame from the application program with matching 29-bit identifier.<br>0xFF: simulate BUSOFF mode of the CAN controller.<br>0: cleared when the message has be processed by the µVision simulator. |
| CAN_OUT    | Set to the following values by the µVision simulator to simulate outgoing messages:<br>1: transmit the current message using a 11-bit identifier.<br>2: transmit the current message using a 29-bit identifier.<br>3: transmit a remote frame using a 11-bit identifier.<br>4: transmit a remote frame using a 29-bit identifier.                                                                                                                                                                                                    |
| CAN_TIMING | Set a performance factor that controls the simulated communication timing within µVision.                                                                                                                                                                                                                                                                                                                                                                                                                                            |

[www.keil.com/appnotes](http://www.keil.com/appnotes): several applications for CAN Bus Programming and Simulation

# CAN Peripheral Dialogs

**CAN Status →  
Controller  
Hardware**



**CAN Controller 1**

Mode: CANMOD: 0x00000000

Command: CANCMR: 0x00000000

Interrupt Enable: ☒ Interrupt & Capture Status

Global Status: CANGSR: 0x0000000C

RxERR: 0x00 BS

TxERR: 0x00 ES

Rx Frame Status: CANRFS: 0x00000000

Tx1 Frame Information: CANTFI1: 0x00000000

Acceptance Filter Mode: AFMR: 0x00000000

LUT Error: LUTerrAd: 0x00000000

LUTerr: 0x00000000

Table Addresses: SFF\_sa: 0x00000000

SFF\_GRP\_sa: 0x00000008

EFF\_sa: 0x00000008

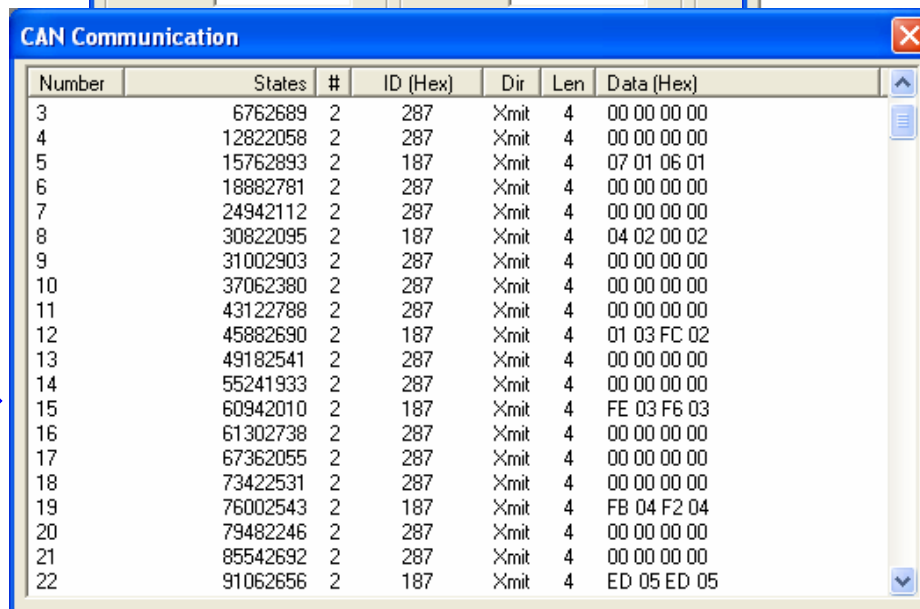
EFF\_GRP\_sa: 0x00000008

ENDofTable: 0x00000008

Acceptance Filter Tables:

| Idx | Table    | Adr   | # | Dis | ID    |
|-----|----------|-------|---|-----|-------|
| 0   | Standard | 0000H | 1 | 0   | 0000H |
| 1   | Standard | 0002H | 1 | 0   | 0207H |
| 2   | Standard | 0004H | 1 | 0   | 0287H |
| 3   | Standard | 0006H | 1 | 0   | 0607H |

**CAN Monitor →  
Shows Bus  
Traffic**



**CAN Communication**

| Number | States   | # | ID (Hex) | Dir  | Len | Data (Hex)  |
|--------|----------|---|----------|------|-----|-------------|
| 3      | 6762689  | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 4      | 12822058 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 5      | 15762893 | 2 | 187      | Xmit | 4   | 07 01 06 01 |
| 6      | 18882781 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 7      | 24942112 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 8      | 30822095 | 2 | 187      | Xmit | 4   | 04 02 00 02 |
| 9      | 31002903 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 10     | 37062380 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 11     | 43122788 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 12     | 45882690 | 2 | 187      | Xmit | 4   | 01 03 FC 02 |
| 13     | 49182541 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 14     | 55241933 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 15     | 60942010 | 2 | 187      | Xmit | 4   | FE 03 F6 03 |
| 16     | 61302738 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 17     | 67362055 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 18     | 73422531 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 19     | 76002543 | 2 | 187      | Xmit | 4   | FB 04 F2 04 |
| 20     | 79482246 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 21     | 85542692 | 2 | 287      | Xmit | 4   | 00 00 00 00 |
| 22     | 91062656 | 2 | 187      | Xmit | 4   | ED 05 ED 05 |

Rx Messages in Full CAN:

| RTR | ID | Rx Data (Hex) |
|-----|----|---------------|
|-----|----|---------------|

# CAN I/O Simulation Scripts



## Simulation eliminates complex Hardware by

- ▶ CAN Monitor Window: shows CAN Message I/O
- ▶ Scripts for CAN Input and Output Messages

## Signal Functions

- ▶ Automated Message Processing
- ▶ Periodic CAN Messages

## Available Soon

- ▶ PEAK CAN Interface connects to real BUS

```
>CAN0ID = 0x4500 // message ID = 0x4500
>CAN0L = 2 // message length 2 bytes
>CAN0B0 = 0x12 // message data byte 0
>CAN0B1 = 0x34 // message data byte 1
>CAN0IN = 2 // send message with 29-bit ID to application
```

*Send CAN Message to Application Program*

```
/*
 * Print the last message sent by the controller
 */
FUNC void CANmessage (void) {
    switch (CAN0OUT) {
        case 1: printf("\nSend 11-bit ID=%04X", CANAID); break;
        case 2: printf("\nSend 29-bit ID=%08X", CANAID); break;
        case 3: printf("\nRequest 11-bit ID=%04X", CANAID); return;
        case 4: printf("\nRequest 29-bit ID=%08X", CANAID); return;
    }
    printf("\nMessage Length %d, Data: ", CAN0L);
    printf("%X %X %X %X ", CAN0B0, CAN0B1, CAN0B2, CAN0B3);
    printf("%X %X %X %X\n", CAN0B4, CAN0B5, CAN0B6, CAN0B7);
}

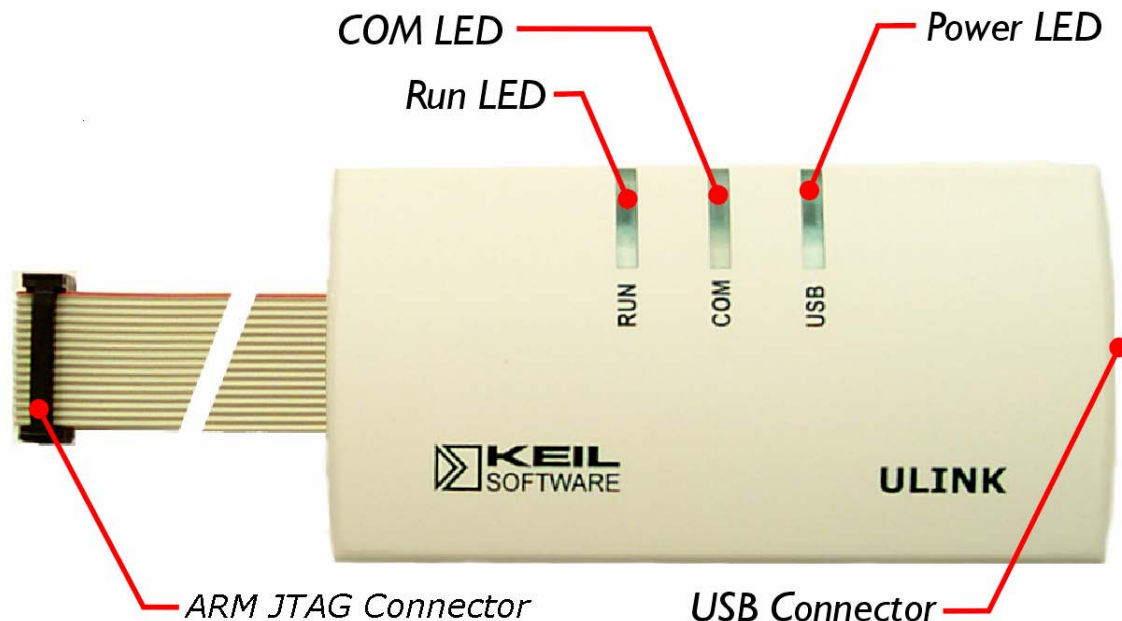
// Set Breakpoint on CAN Output VTREG
BS WRITE CAN0OUT, 1, "CANmessage()"
```

*Print CAN Messages Send from Application Program*

# ULINK – USB / JTAG Adapter



|       |    |    |                |
|-------|----|----|----------------|
| VCC   | 1  | 2  | VCC (optional) |
| TRST  | 3  | 4  | GND            |
| TDI   | 5  | 6  | GND            |
| TMS   | 7  | 8  | GND            |
| TCLK  | 9  | 10 | GND            |
| RTCK  | 11 | 12 | GND            |
| TDO   | 13 | 14 | GND            |
| RESET | 15 | 16 | GND            |
| N/C   | 17 | 18 | GND            |
| N/C   | 19 | 20 | GND            |



## Program Debugging

- ▶ Single Stepping & Real-Time Execution
- ▶ Examine Memory and CPU Registers
- ▶ Flash and Software Breakpoints
- ▶ Memory Access Breakpoints

## Flash Programming

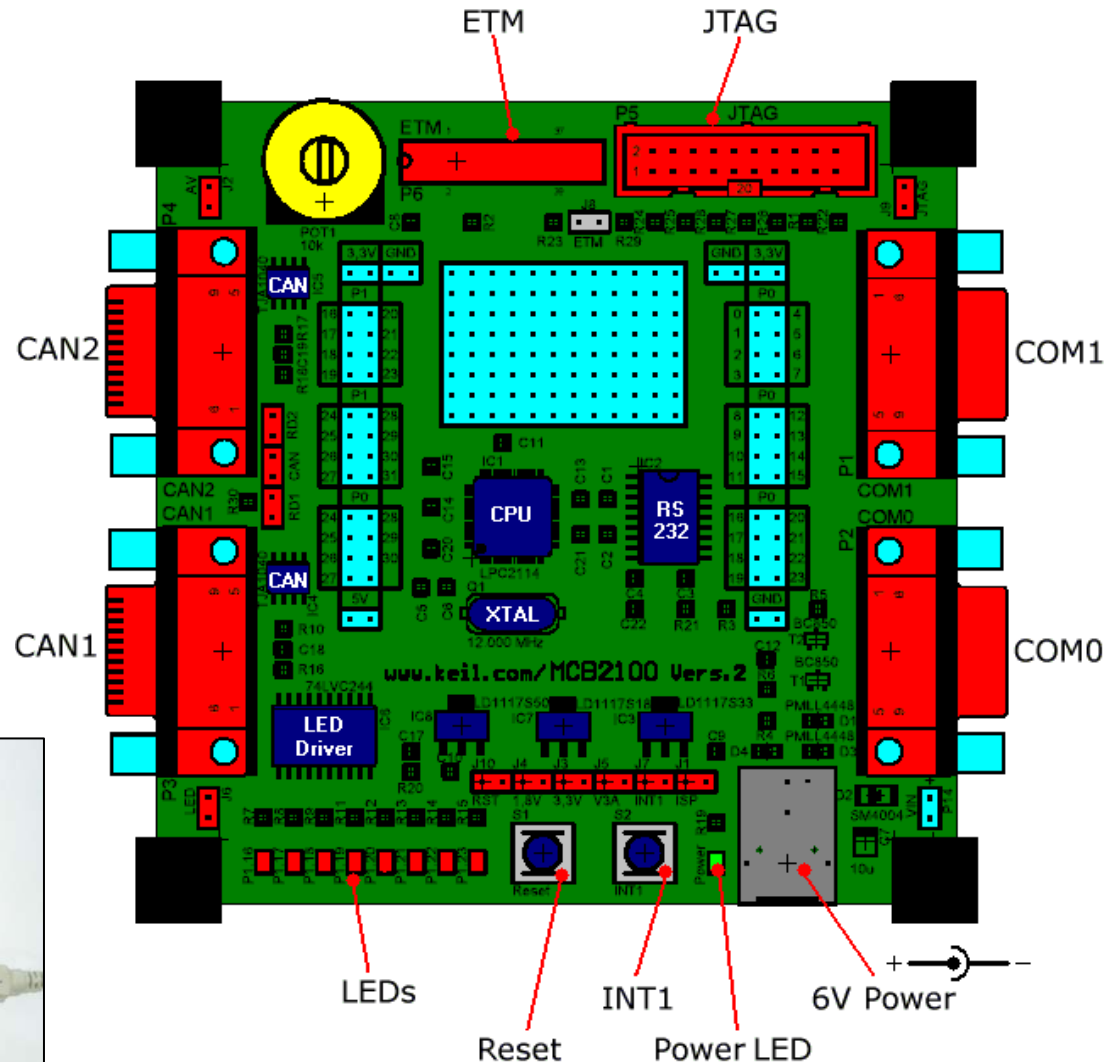
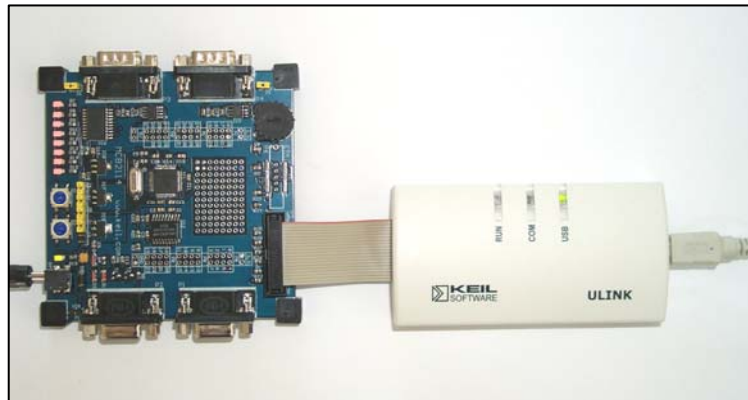
- ▶ Download, Verify, and Execute
- ▶ Fast Download: 20KB / Second
- ▶ Customized Programming Algorithms

# MCB2100 Evaluation Board

Flexible and Easy-to-Use  
Evaluation Board with:

- ▶ **LPC 2129** in Single Chip Design
- ▶ **Dual CAN Interface**
- ▶ **Dual UART Interface**
- ▶ **JTAG Interface**
- ▶ **ETM optional**
- ▶ **8 Status LED's**
- ▶ **Analog Inputs**

Also available MCB2130  
with LPC2138

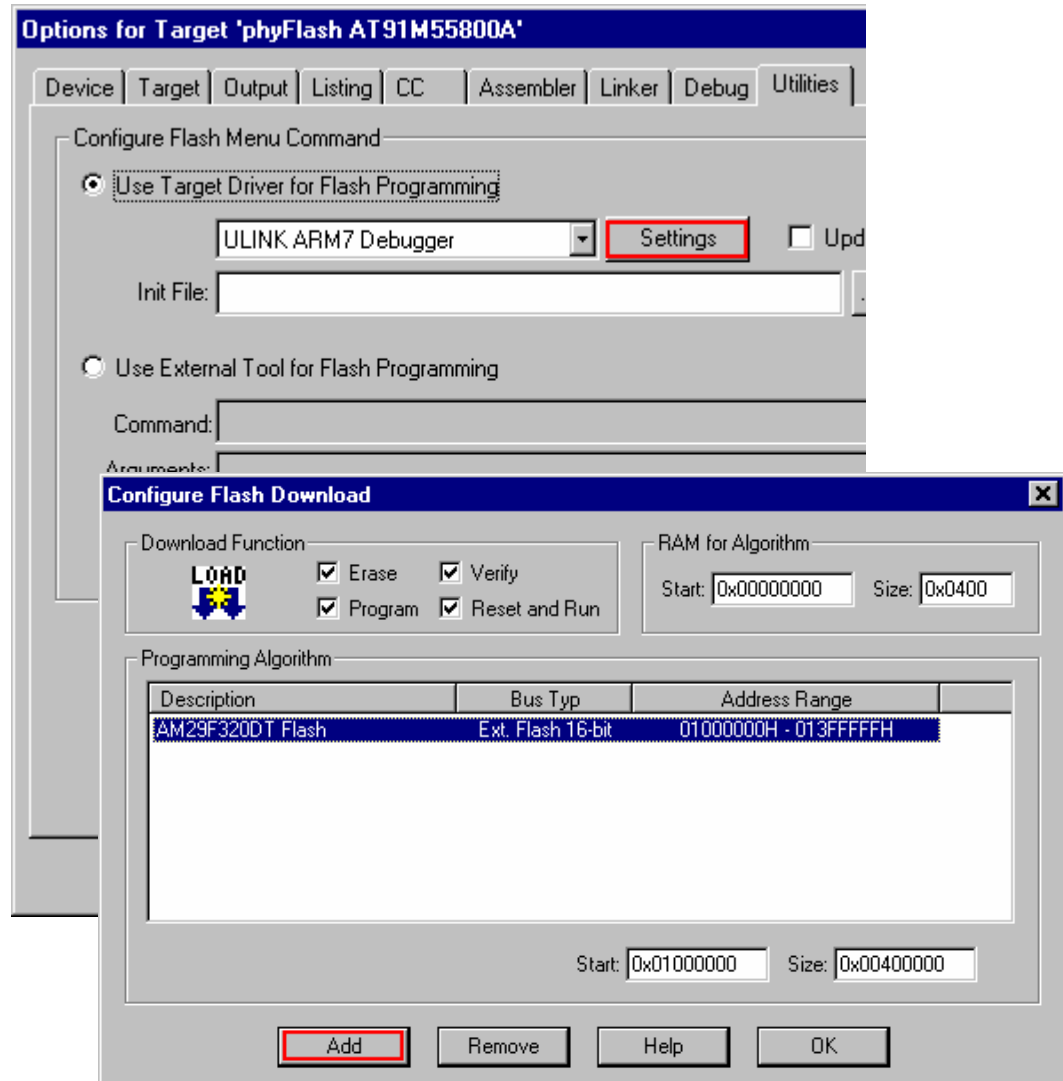


# ULINK Flash Programming



## Configure under Options - Utilities

- ▶ Fully Integrated into Build Process
- ▶ Preconfigured Algorithms
  - on-chip Flash ROM
  - selected Boards
- ▶ User may add own Program Algorithms



# ARM Architecture Overview

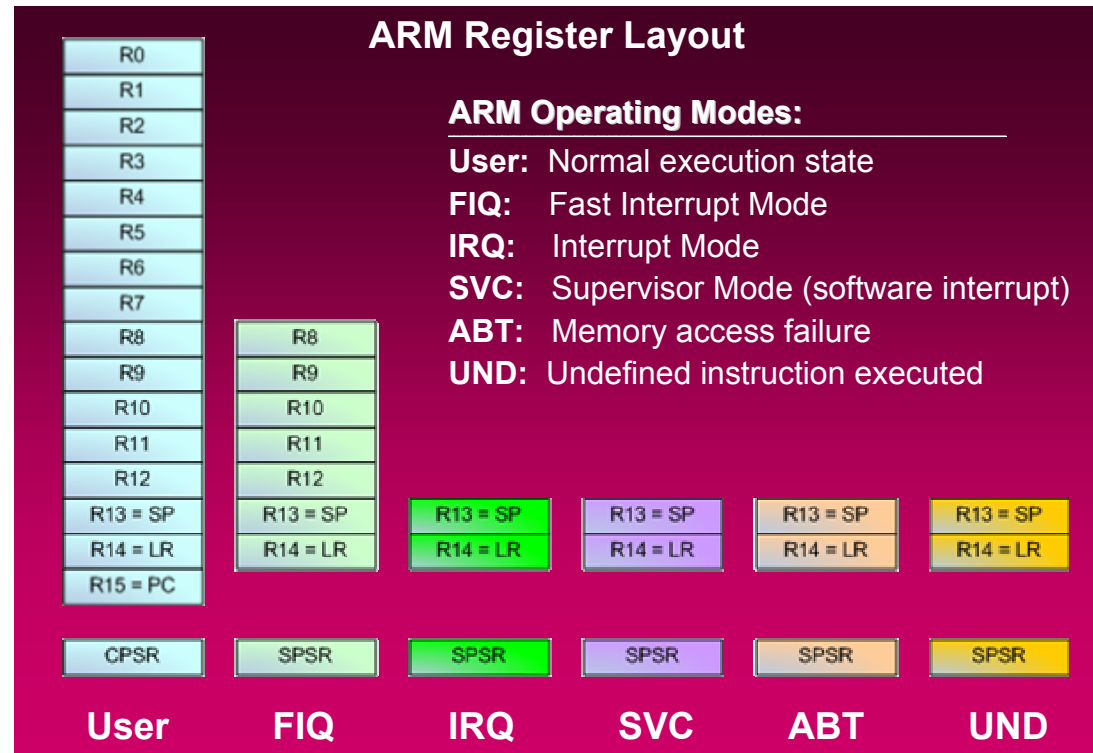


# ARM

## Industry Standard 32-Bit Architecture used by many Silicon Vendors

## CPU Highlights:

- 4 MByte Address Space
- Two interrupt levels
- Fast Interrupt with Registers
- CPU modes with own stack
- Allows fast context switch for Real Time Operating Systems
- Peripherals are mapped into linear address space
- On Philips devices: Vectored Interrupt Controller (VIC)
- Load Store architecture with efficient pointer addressing
- Compact 16-bit Instruction Set (Thumb)



The Keil CARM Compiler is optimized for Thumb mode. Compared to ARM native mode, Thumb code is 65% of the code size and executes 160% faster from a 16-bit memory system. Thumb mode execution is even faster on most on-chip Flash ROM Microcontrollers.

## CARM Overview:

- Function In-lining removes call / return overhead and optimizes parameter passing.
- In-line assembly with full variable access supports special ARM instructions in C.
- RAM functions enable high-speed interrupt code and In-System Flash Programming.
- Automatic ARM/Thumb interworking gives outstanding code density and CPU performance.
- Locate variables and functions at C level to absolute memory addresses.
- Function attributes for IRQ, FIQ, and Task (RTOS support) built into the C compiler.
- Software Interrupt support built into the C compiler enables atomic instruction sequences.

```
// ARM function in RAM
void fastfunc (void) arm ram {
    flash[I] = buffer[I];
}

void mycode (void) thumb {
    fastfunc ();
}
```

*Automatic ARM / Thumb Functions Interworking*

```
/*
 *   Atomic AND operation: *val &= mask
 */

U32 AImask (U32 *val, U32 mask) arm swi(8) {
    asm {MSR CPSR_c, #0xd3}      // disable FIQ
    return (*val &= mask);
}
```

*SWI Functions are protected by IRQ interrupts*

# Keil CARM Function In-Lining



The Keil CARM Compiler performs optimizations at the Linker Level. Function in-lining and register allocation is optimized across complete applications. This optimization technology removes the overhead involved with Hardware Abstraction Layers (as used by UML).

```
void change_state (void)  {
    LED (5, ON);
    LED (12, OFF);
}
```

*Function that changes LED status*

```
extern inline void LED (unsigned int no, int v)  {
    if (no < 8)  PortOut (IO1, 1<<(no), v);
    else        PortOut (IO0, 1<<(no-8), v);
}
```

*LED interface to physical I/O ports*

```
extern inline void PortOut (int io, int pos, int v) {
    switch (io)  {
        case IO0:  if (v == OFF)  IOCLR0 |= pos;
                    else          IOSET0 |= pos;  break;

        case IO1:  if (v == OFF)  IOCLR1 |= pos;
                    else          IOSET1 |= pos;  break;
    }
}
```

*Generic physical I/O port routine*

Code with  
Function Inlining →

```
void change_state (void)  {
    LED (5, ON);
    IOSET1 |= pos;
        MOV     R2,#0x20
        LDR     R0,[R15,#36] ; IOSET0
        LDR     R1,[R0,#0x0]
        ORR     R1,R2
        STR     R1,[R0,#0x0]

    LED (12, OFF);
    IOCLR1 |= pos;
        MOV     R2,#0x10
        LDR     R0,[R15,#20] ; IOCLR1
        LDR     R1,[R0,#0x0]
        ORR     R1,R2
        STR     R1,[R0,#0x0]
        BX      R14
}
```

*Generated Code*

Global Function In-lining saves call overhead and reduces code size.

# Keil CARM Compiler Warnings



The Keil CARM Compiler performs extensive program flow checking and syntax checking. CARM generates detailed warning messages for C source constructs that are likely to fail which helps you to minimize sporadic run-time problems.

## CARM generates Warnings for:

- Mismatches in data types, pointers and function arguments.
- Use of un-initialized variables.
- Dead code and dead assignments.
- Unused variables, labels, functions.
- Constant expressions in conditional statements.
- Missing return values.

The Keil LARM Linker performs cross module data type checking and deletes unused functions from your program.

**µVision even integrates PC-LINT for extensive syntax checking.**

```
struct t { int i; } *pt;
#define TSTVAL(v) ((v)&0x10)

int myfunc (unsigned int c1, int *p2) {
    int error, val;
    *** warning: 'val': unreferenced local variable

    if (TSTVAL(c1) & 0x20) {
        return -1;
    }
    *** warning C177: unreachable code
    switch (c1) {
        case 1: error = 0; break;
        case 2: error = 1; break;
    }
    if (p2) error = getstring ();
    pt = p2;
    *** warning: '=': pointer to different objects
    return error;
    *** warning: value of 'error' possibly undefined
}
```

*Warnings generated by Keil CARM*

# Keil CARM Benchmarks

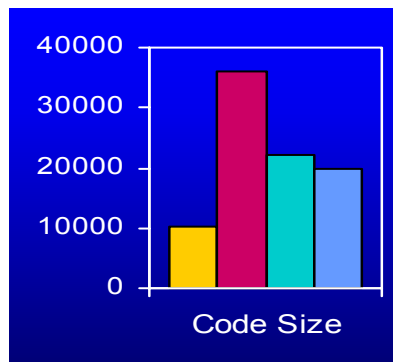


The Run-Time Library of the Keil CARM Compiler is written mainly in Thumb mode and fits therefore embedded systems with limited memory resources. Most library functions are re-entrant which is essential for interrupt and multi-threading (RTOS) support.

## Dhrystone 2.1 (Integer and String Operations)

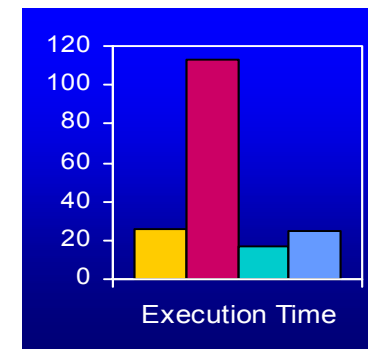
Program Size

| Compiler  | Code Size   | Data Size   |
|-----------|-------------|-------------|
| Keil CARM | 10330 Bytes | 10256 Bytes |
| GNU       | 36004 Bytes | 11912 Bytes |
| ARM ADS   | 22266 Bytes | 10256 Bytes |
| IAR       | 19892 Bytes | 10269 Bytes |



Execution Time

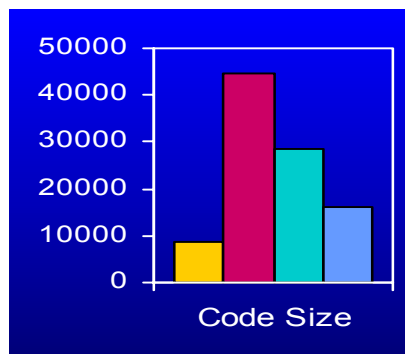
| Compiler  | μSeconds |
|-----------|----------|
| Keil CARM | 25.4     |
| GNU       | 112.9    |
| ARM ADS   | 16.8     |
| IAR       | 24.4     |



## Whetstone (Floating Point Operations)

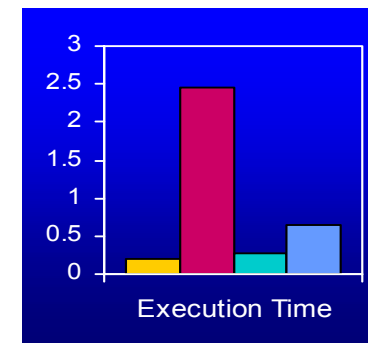
Program Size

| Compiler  | Code Size   | Data Size  |
|-----------|-------------|------------|
| Keil CARM | 8492 Bytes  | 72 Bytes   |
| GNU       | 44428 Bytes | 4768 Bytes |
| ARM ADS   | 28516 Bytes | 76 Bytes   |
| IAR       | 16316 Bytes | 72 Bytes   |



Execution Time

| Compiler  | Seconds  |
|-----------|----------|
| Keil CARM | 0.195308 |
| GNU       | 2.461430 |
| ARM ADS   | 0.268623 |
| IAR       | 0.635633 |

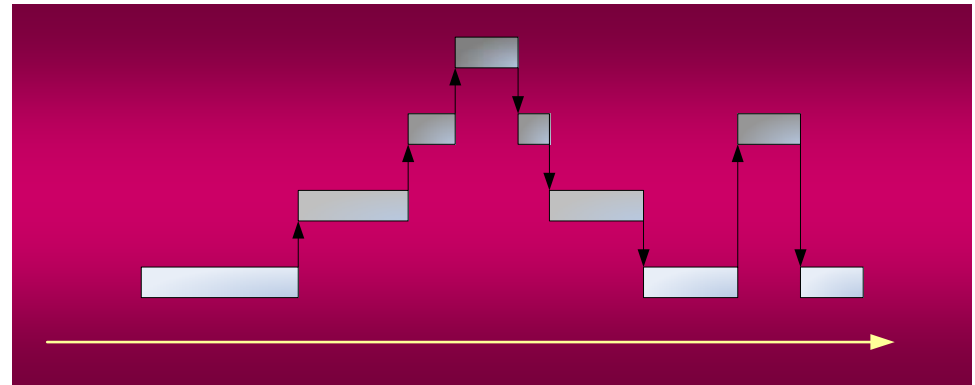


# ARM Software Concepts

The ARM core requires a different mindset for embedded applications. ARM has just two interrupt levels **Standard (IRQ)** and **Fast (FIQ)** interrupt but provides **CPU modes** with separate interrupt stacks for predictable stack requirements.

## Running 'main' in End-less Loop

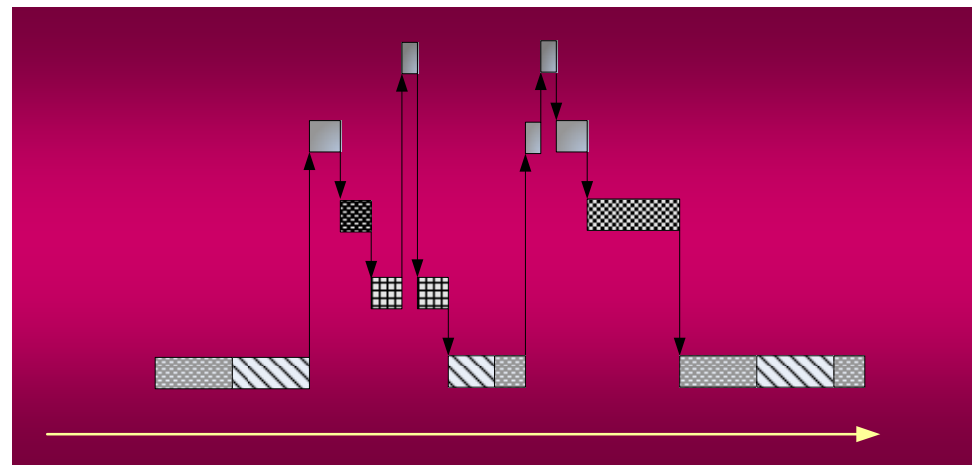
- Solution for simple Applications
- Usage together with powerful multi-level interrupt system
- Stack Usage un-predictable



*Classic Embedded Application with nested Interrupts*

## Using an Real-Time OS Kernel

- Allows to separate the application into tasks that run independent
- Message passing eliminates critical shared memory buffers
- Each Task has an own stack area
- Interrupt communication with event flags and messages.



*RTOS Application with Interrupts on ARM*

The Advanced RTX RTOS Kernel is a full-featured Real-Time Operating System with task priorities, round-robin, pre-emptive context switching, and support for multiple instances of the same task. Advanced RTX interfaces to I-Logix Rhapsody using the Willert UML Bridge.

## Advanced RTX RTOS Kernel - Function Overview

### Task Management

- Create and delete tasks
- Change Task Priorities

### Event Flag Management

- Wait for multiple Events
- Send Events from Tasks or Interrupt Functions

### Semaphore Management

- Efficient Resource Sharing

### System Functions

- Task Lock / Unlock

### Mailbox Management

- Intertask Communication
- Interface to Interrupt Functions

### Fixed Memory Block Allocation

- Thread-safe (usage even in ISR)
- Deterministic reaction times

### Time Management

- Task delays (timeout or interval)

### User Timer Management

- Timed invocation of Callback Functions

The TCP/IP networking layer of Advanced RTX is a ground-up implementation designed for embedded applications. The networking layer helps you to create Internet or Intranet applications with UDP or TCP/IP protocol using Ethernet, PPP, or SLIP interfaces.

## Advanced RTX Networking Layer - Function Overview

### Socket Interfaces

- TCP/IP
- UDP

### Physical Interfaces

- Ethernet
- PPP (serial connection)
- SLIP (dial-up line)

### Configuration

- Flexible networking function parameters
- Configuration with the  $\mu$ Vision Configuration Wizard

### Standard Applications

- HTTP Web Server with CGI Scripting
- Telnet Server
- TFTP Server
- SMTP Client
- DNS Resolver

### Debugging

- Multiple Debug Levels:
  - Errors only
  - Complete status information
- Outputs status information on UART interface

# Advanced RTX Examples



 C:\Keil\ARM\ARTX\Examples

Example Projects show basic configuration and help you to get started quickly.  
All examples run on Evaluation Boards.

## RTOS Kernel Examples

-  Artx\_ex1 → Use basic RTOS kernel features: timeouts & signals.
-  Artx\_ex2 → Show task priorities and signal passing.
-  Mailbox → Using the Mailbox and Memory Allocation functions
-  Traffic → Complete Traffic Light Controller with serial communication

## TCP/IP Networking Examples (Run on Phytex PhyCore LPC229x Eval Board)

-  Http\_demo → HTTP Server with Password Protection and CGI Scripting
-  Telnet\_demo → Telnet Server shows a simple IP based command line interface
-  DNS\_demo → Using the DNS resolver that connects to host names.
-  LEDSwitch → Controlling with TCP/IP, UCP via Ethernet, SLIP or PPP link.
-  SMTP\_demo → Shows sending of a dynamic message to an email address

# Advanced RTX Task Status



## Review Task Status During Program Execution

```
089 void tcp_poll (void) _task_ 4 _priority_ 2 ...
097 ...
098 void init (void) _task_ 1 _priority_ 2 {
099     init_TcpNet ();
100     system_init();
101     tsk1 = os_tsk_self ();           /* get own task identification */
102     tsk2 = os_tsk_create (2);        /* start task 2 */
103     tsk3 = os_tsk_create (3);        /* start task 3 */
104     tsk4 = os_tsk_create (4);        /* start task 4 */
105     os_tsk_delete_self();            /* stop init task (done) */
106 }
107
108 void main (void) {
109     os_sys_init (1);                 /* initialize Kernel start init task */
110 }
111
```

| TID | Task Name      | Priority | State    | Delay | Event Value | Event Mask | Stack load |
|-----|----------------|----------|----------|-------|-------------|------------|------------|
| 0   | os_clock_demon | 0        | INACTIVE |       |             |            | 100%       |
| 255 | os_idle_demon  | 0        | INACTIVE |       |             |            | 100%       |

System is not yet initialized

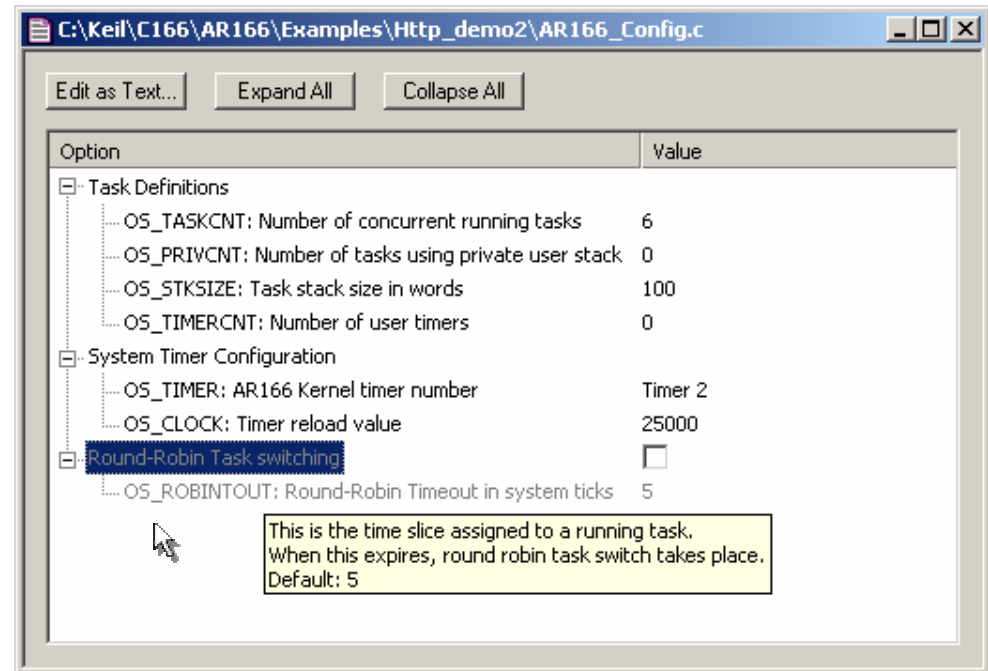
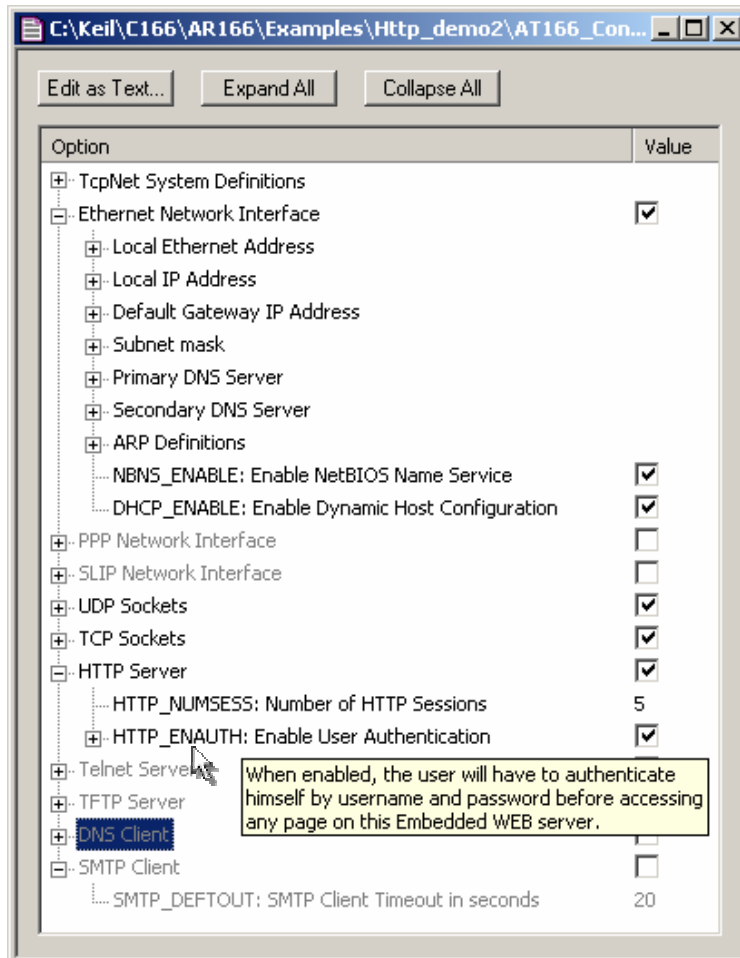
```
089 void tcp_poll (void) _task_ 4 _priority_ 2 ...
097 ...
098 void init (void) _task_ 1 _priority_ 2 {
099     init_TcpNet ();
100     system_init();
101     tsk1 = os_tsk_self ();           /* get own task identification */
102     tsk2 = os_tsk_create (2);        /* start task 2 */
103     tsk3 = os_tsk_create (3);        /* start task 3 */
104     tsk4 = os_tsk_create (4);        /* start task 4 */
105     os_tsk_delete_self();            /* stop init task (done) */
106 }
107
108 void main (void) {
109     os_sys_init (1);                 /* initialize Kernel start init task */
110 }
111
```

| TID | Task Name      | Priority | State    | Delay | Event Value | Event Mask | Stack load |
|-----|----------------|----------|----------|-------|-------------|------------|------------|
| 4   | tcp_poll       | 2        | READY    |       |             |            | 23%        |
| 3   | timer_poll     | 2        | READY    |       |             |            | 23%        |
| 2   | 0x00C0A796     | 2        | READY    |       |             |            | 23%        |
| 1   | init           | 2        | RUNNING  |       |             |            | 24%        |
| 0   | os_clock_demon | 255      | WAIT_ITV | 1     |             |            | 14%        |
| 255 | os_idle_demon  | 0        | READY    |       |             |            | 17%        |

Init() task was executed.  
Tasks and TCP/IP stack are  
initialized and running.

# Advanced RTX Setup

## Easy Parameter Setup using the $\mu$ Vision Configuration Wizard



The  $\mu$ Vision Configuration Wizard allows you to review and change all major parameters of the RTOS Kernel and the TCP/IP Layer.

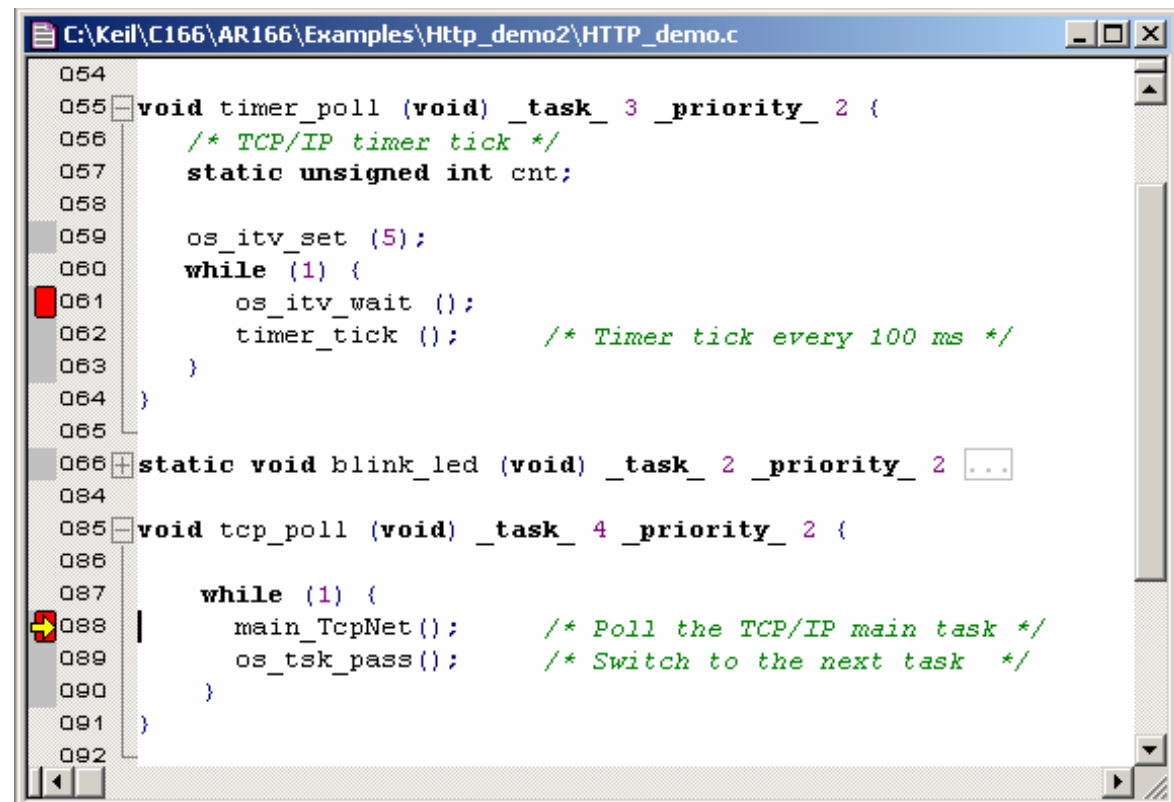
# Using the TCP/IP Layer

## TCP/IP may be used with and without the RTOS Kernel

- ▶ Call timer\_tick() in fixed intervals to generate TCP/IP timings
- ▶ Poll the main\_TcpNet function as often as possible

**timer\_tick () →**  
Called every  
100 ms using  
Interval wait

**main\_TcpNet () →**  
Called whenever no  
other task is ready

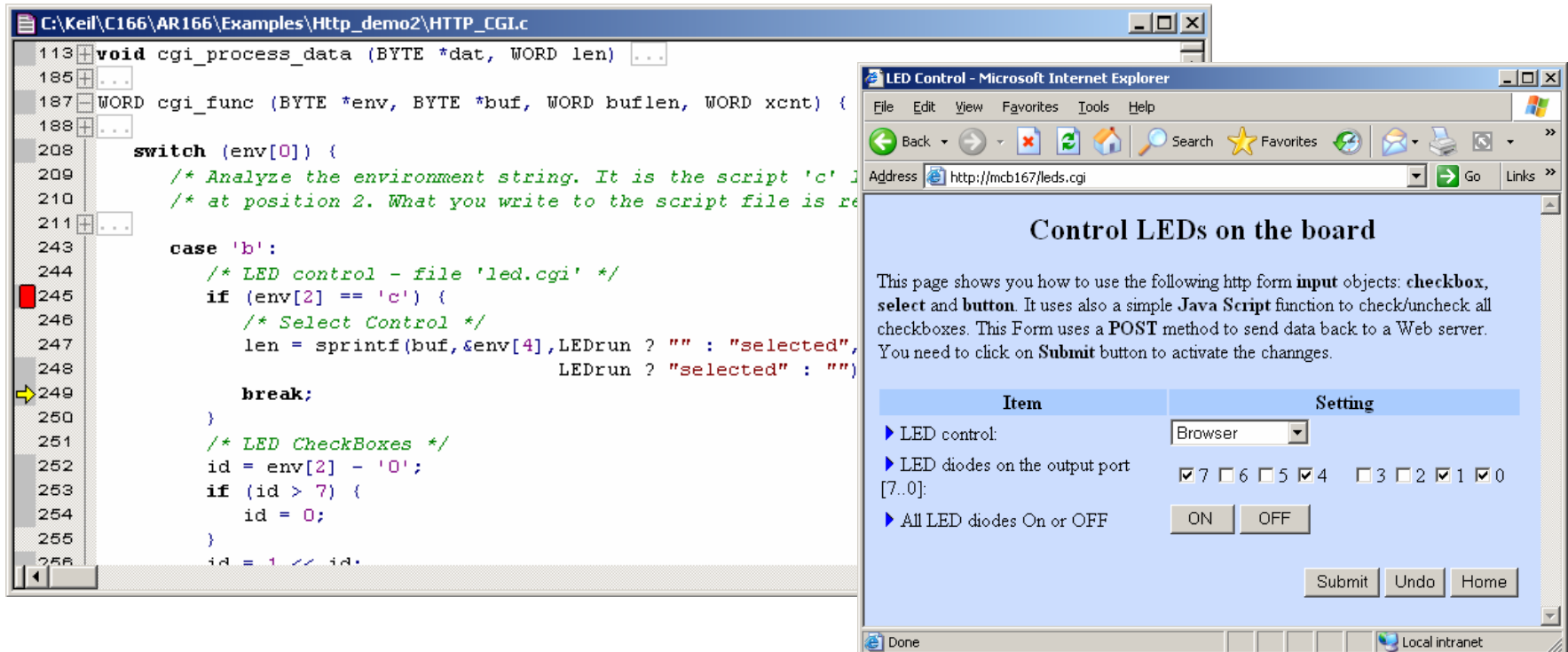


```
C:\Keil\C166\AR166\Examples\Http_demo2\HTTP_demo.c
054
055 void timer_poll (void) _task_ 3 _priority_ 2 {
056     /* TCP/IP timer tick */
057     static unsigned int cnt;
058
059     os_itv_set (5);
060     while (1) {
061         os_itv_wait ();
062         timer_tick ();    /* Timer tick every 100 ms */
063     }
064 }
065
066 static void blink_led (void) _task_ 2 _priority_ 2 ...
067
068 void tcp_poll (void) _task_ 4 _priority_ 2 {
069
070     while (1) {
071         main_TcpNet();    /* Poll the TCP/IP main task */
072         os_tsk_pass();    /* Switch to the next task */
073     }
074 }
075
```

# Advanced RTX Examples

## HTTP Server with CGI Interface

- ▶ The HTTP server provides authentication and allows multiple sessions at the same time.
- ▶ An CGI interface in the webserver allows to interact with your microcontroller hardware.



The screenshot shows the Keil IDE on the left with the file `C:\Keil\C166\AR166\Examples\Http_demo2\HTTP_CGI.c` open. The code defines a CGI interface for controlling LEDs. On the right, a Microsoft Internet Explorer window displays the "LED Control" web page at `http://mcb167/leds.cgi`.

**LED Control - Microsoft Internet Explorer**

Address: `http://mcb167/leds.cgi`

### Control LEDs on the board

This page shows you how to use the following http form input objects: **checkbox**, **select** and **button**. It uses also a simple Java Script function to check/uncheck all checkboxes. This Form uses a **POST** method to send data back to a Web server. You need to click on **Submit** button to activate the changes.

| Item                                    | Setting                                                                                                                                                                                                                                                             |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ▶ LED control:                          | <input type="text" value="Browser"/>                                                                                                                                                                                                                                |
| ▶ LED diodes on the output port [7..0]: | <input checked="" type="checkbox"/> 7 <input type="checkbox"/> 6 <input type="checkbox"/> 5 <input checked="" type="checkbox"/> 4 <input type="checkbox"/> 3 <input type="checkbox"/> 2 <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 0 |
| ▶ All LED diodes On or OFF              | <input type="button" value="ON"/> <input type="button" value="OFF"/>                                                                                                                                                                                                |

Done Local intranet

```
113 void cgi_process_data (BYTE *dat, WORD len) ...
185 ...
187 WORD cgi_func (BYTE *env, BYTE *buf, WORD buflen, WORD xcnt) {
188 ...
208 switch (env[0]) {
209     /* Analyze the environment string. It is the script 'c' ...
210     /* at position 2. What you write to the script file is re
211 ...
243 case 'b':
244     /* LED control - file 'led.cgi' */
245     if (env[2] == 'c') {
246         /* Select Control */
247         len = sprintf(buf, &env[4], LEDrun ? "" : "selected",
248                     LEDrun ? "selected" : "");
249         break;
250     }
251     /* LED CheckBoxes */
252     id = env[2] - '0';
253     if (id > 7) {
254         id = 0;
255     }
256     id = 1 << id;
```

# Get More Information



## www.keil.com

- ▶ Application Notes
- ▶ Program Examples
- ▶ Support Knowledgebase
- ▶ Device Database
- ▶ Discussion Forum

The collage displays five different pages from the Keil Software website, each shown in a Microsoft Internet Explorer browser window:

- Keil Software Application Notes (Listed by Category)**: A page listing various application notes categorized by product line.
- Keil Software Download File Area**: A page providing links to download software and documentation.
- ARM: WHY ARE MOST KEIL EXAMPLES USING THUMB MODE**: A technical article or support document.
- Philips LPC2138**: A page dedicated to the Philips LPC2138 microcontroller, including technical details and support resources.
- Discussion Forum**: A forum page titled "Threads 1-25 of 5,644" showing a list of user discussions. The forum includes navigation links (Home, Products, Events, Support, Search, Index) and a table of threads.

| No.  | Subject                                                                                 | Author           | Product | Last Reply     | Replies |
|------|-----------------------------------------------------------------------------------------|------------------|---------|----------------|---------|
| 5774 | <a href="#">* string definition</a>                                                     | Tam PT           | ARM     | 03/29/05 01:04 | 7       |
| 5764 | <a href="#">* code &amp; data allocation on flash of xc167</a>                          | A-w S            | C166    | 03/29/05 00:36 | 1       |
| 5779 | <a href="#">* No Peripherals in uVision 2 Debug C8051F016</a>                           | david marten     | C51     | 03/28/05 23:03 | 0       |
| 5778 | <a href="#">* I need the help of expert</a>                                             | hikaru sai       | None    | 03/28/05 19:49 | 0       |
| 5773 | <a href="#">* who can tell me some usefull websites or forums about Hardware,ele...</a> | he dingqiu       | C51     | 03/28/05 18:31 | 2       |
| 5707 | <a href="#">* Problem calling function from SRAM</a>                                    | M hameed         | ARM     | 03/28/05 18:30 | 4       |
| 5763 | <a href="#">* need help with 8515 microcontroller</a>                                   | MAN KKL          | None    | 03/28/05 15:12 | 3       |
| 5777 | <a href="#">* How do I select the appropriate memory model?</a>                         | Darrell Werries  | C166    | 03/28/05 15:04 | 0       |
| 5776 | <a href="#">* Keil CARM 16K size code restrictions</a>                                  | Daniel Benmelej  | ARM     | 03/28/05 14:50 | 1       |
| 5755 | <a href="#">* About function call</a>                                                   | Edison Lee       | C51     | 03/28/05 12:30 | 8       |
| 5753 | <a href="#">* I need to access two memory of xdata</a>                                  | Balaji Venugopal | C51     | 03/28/05 12:29 | 11      |