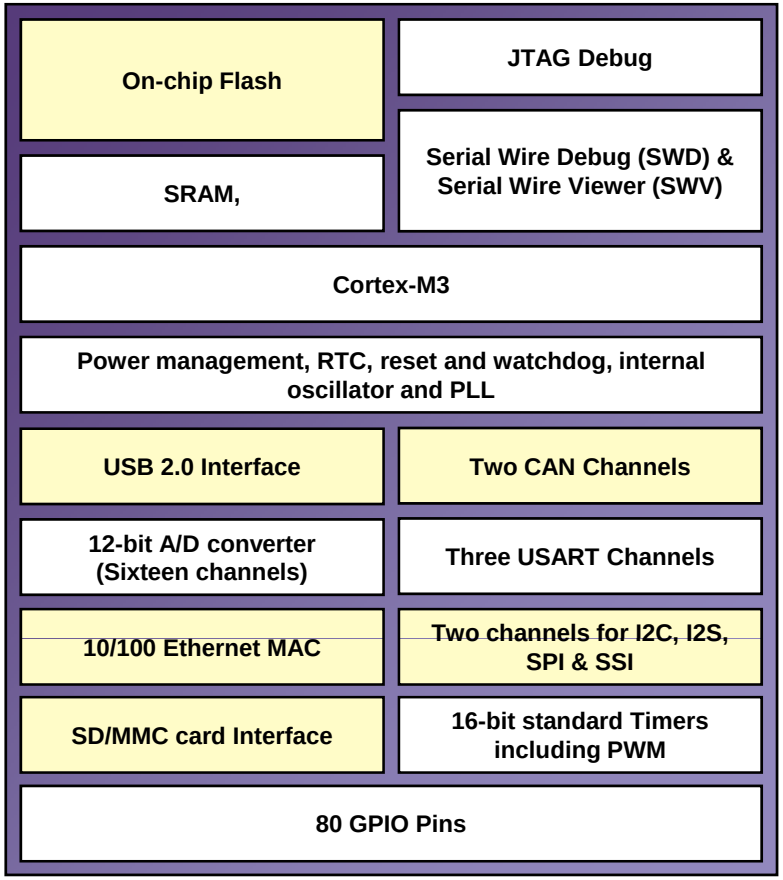
Middleware for Cortex-M based Microcontroller

KEIL™ Tools by ARM

Today's Microcontroller Selection

- Microcontroller have
  - Processor
  - On-chip Memory
  - Interrupt System
  - Rich peripheral set
    - I/O Pins, Timers, PWM
    - A/D and D/A converters
    - UART, SPI, I2C
    - Complex communication peripherals (CAN, USB, Ethernet)



Block Diagram of a Standard Microcontroller

Embedded Connectivity Challenges

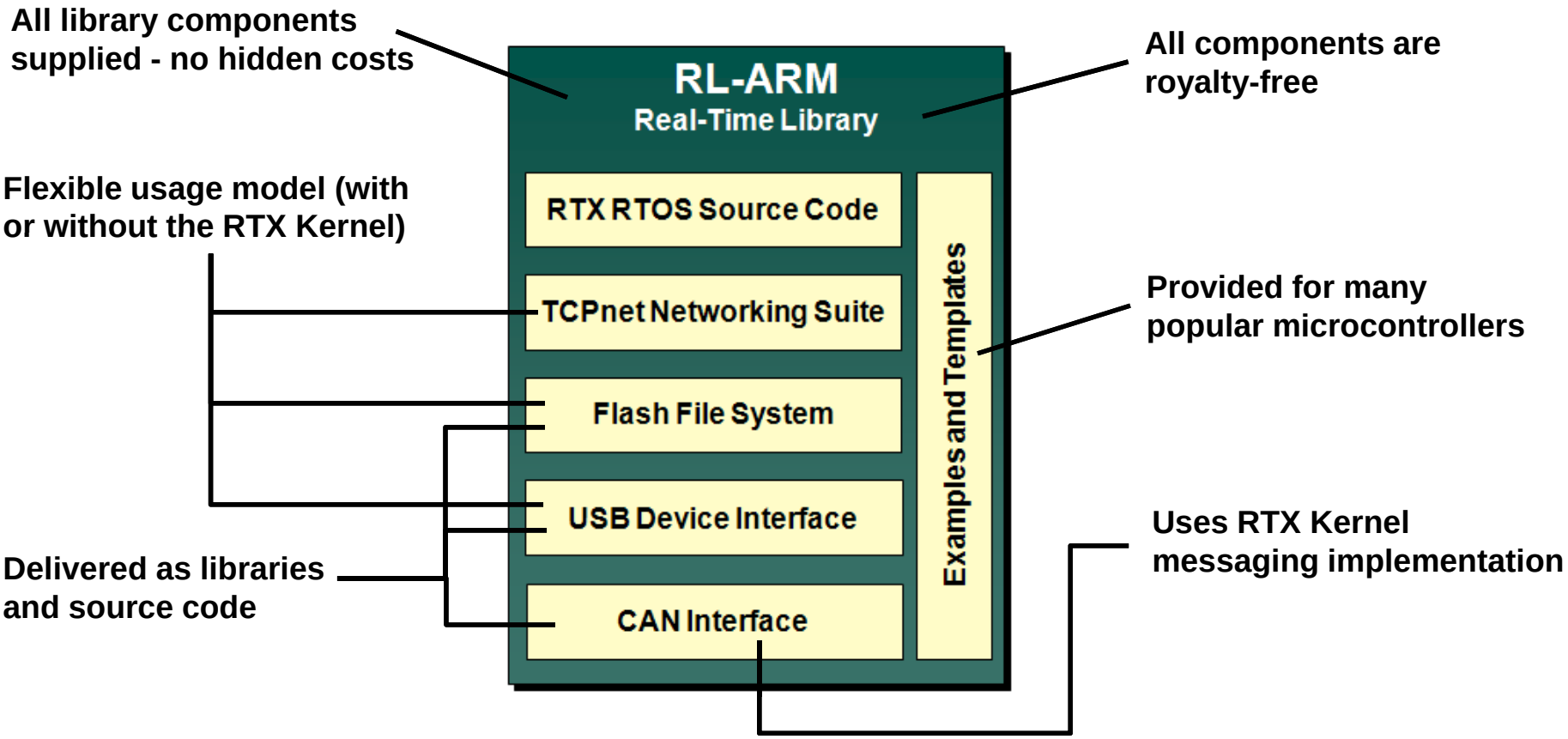
- Embedded devices are used everywhere
  - Need to support many different interfaces…
    - CAN, USB, SD/MMC, Ethernet
  - …and different protocols
    - HTTP, FTP, SMTP...
- Customers demand ease of use
  - Today's embedded devices need to support plug and play compatibility
- Developers need more functionality
  - Ability to support a wide range of interfaces
  - Need better development and debug tools for this task

RL-ARM

RTX Real-Time Kernel
TCPnet Networking Suite
Flash File System
USB Device Interface
CAN Interfaces

What is RL-ARM?

- A collection of resources for solving these challenges
  - Middleware components created and used by ARM engineers



Where is RL-ARM used?

- Everywhere that embedded devices are connected
  - It supports traditional embedded functions
    - For example CAN in industrial applications
  - And emerging applications for embedded devices
    - Web-based and mass storage products

- In simple and complex applications
  - Optimized routines give fast performance from a small code footprint
  - Component libraries can be used stand-alone or integrated with other resources and optional RTX kernel
  - Templates and examples are provided for all applications on many popular microcontrollers

How does RL-ARM work for me?

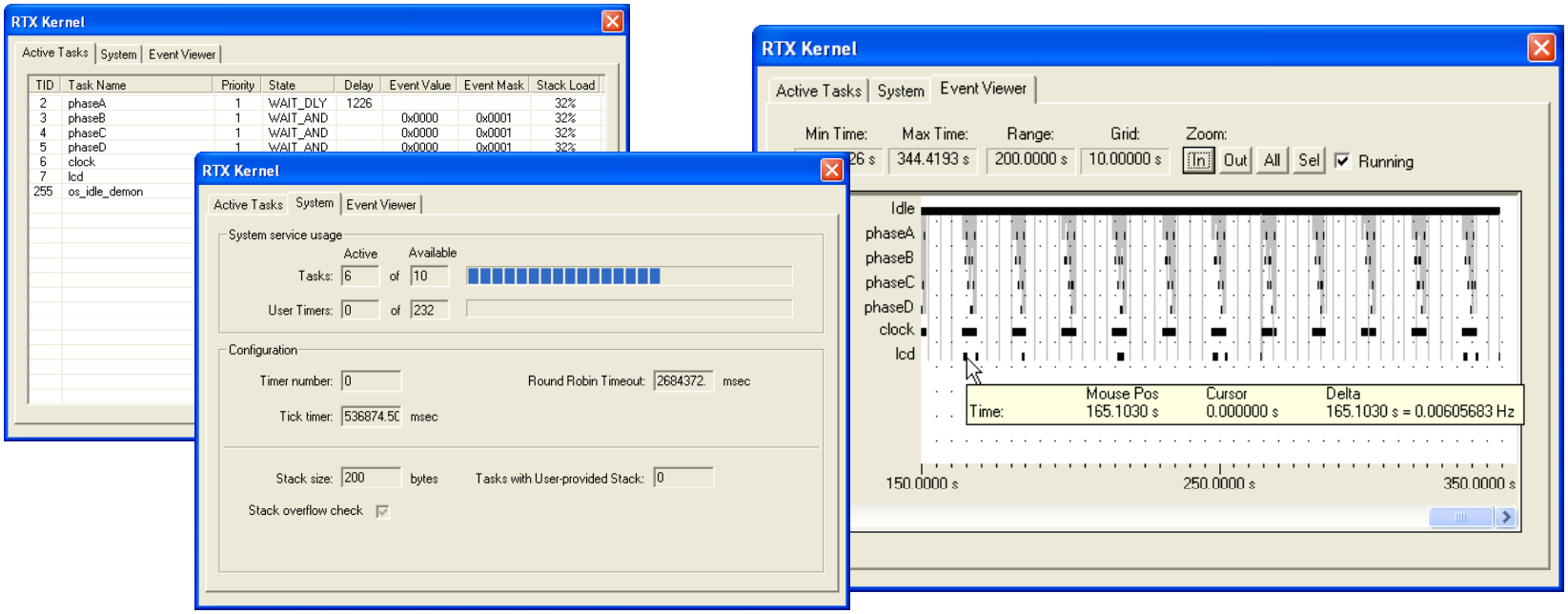
- Integrated solution
  - Developed with MDK-ARM, the tools and middleware are guaranteed to work together
  - ARM engineers can support every part of your project

- Cost effective
  - Allows you to focus effort on developing the important parts of your application
  - Provides tested and optimized components

- Proven and reliable
  - Thousands of designs using RL-ARM in the field today
  - Trusted in applications by ARM and its Partners

Using RL-ARM with MDK-ARM

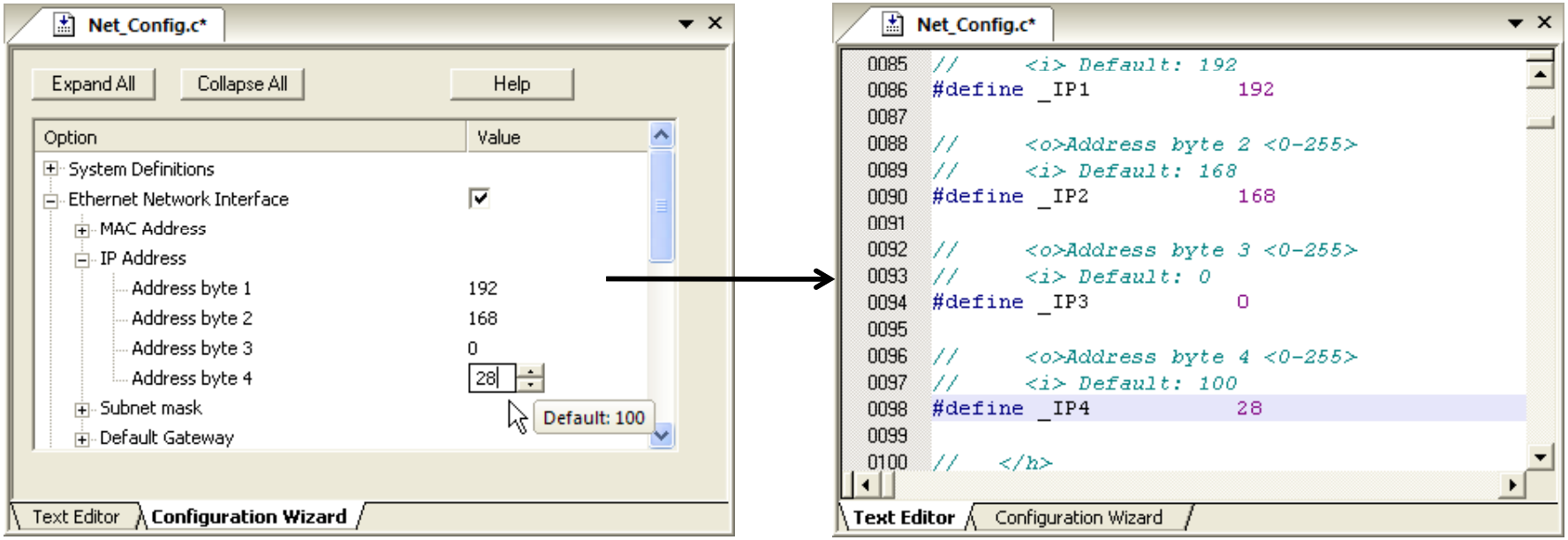
- MDK includes dedicated support for RL-ARM functionality
  - Examples supplied as µVision projects – ready to build
  - Build options include settings for RL-ARM resources
  - µVision supports RTX Kernel-aware debugging



Detailed view of system status from µVision IDE

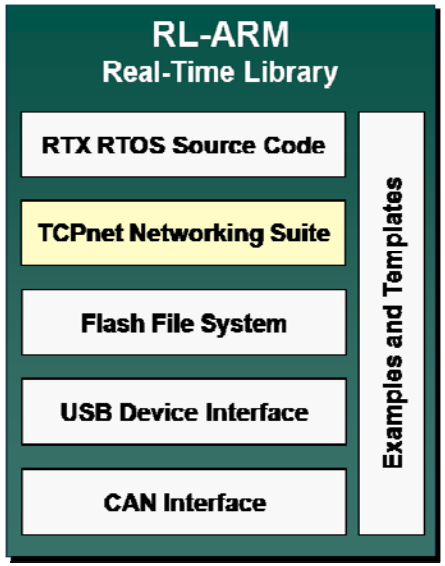
µVision Configuration Wizard

- User-friendly way to adjust settings
  - No need to search for relevant source code sections
    - All useful parameters are instantly accessible
  - Less risk of making mistakes
    - Simple checking of selected values



RL-TCPnet

TCP/IP Networking Suite



TCPnet Networking Suite

- Add network support to your projects quickly and easily
  - Libraries support common network protocols
  - Supplied with templates and examples ready to port to any target
  - Take advantage of standard networking applications

Email, SMTP

Modem, PPP

Remote Access, Telnet

Serial, SLIP

Web interface, HTTP

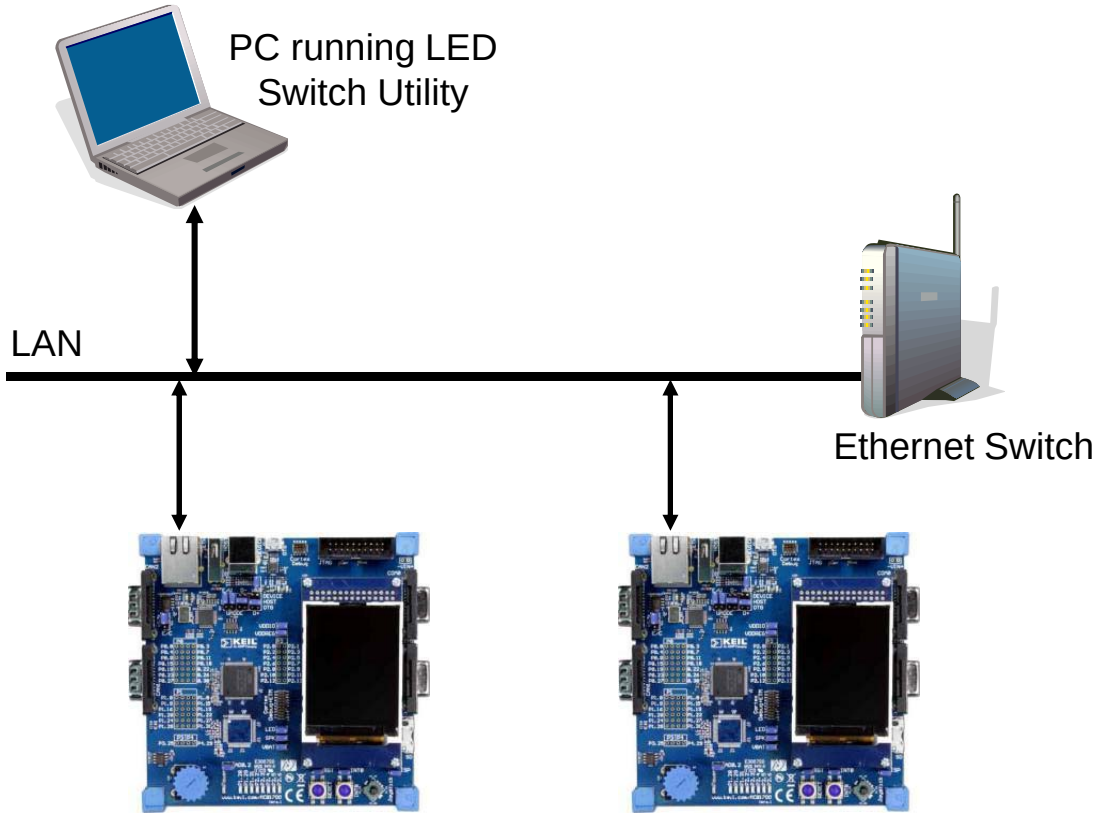
ARP, IEEE 802.xx network

TCPnet Networking Suite

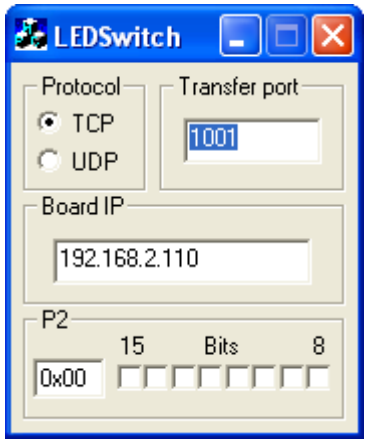
HTTP Server		Telnet Server		SMTP Server	
CGI Scripting		FTP Server		DNS Resolver	
TCP	UDP	ARP	DHCP	PPP	SLIP
Ethernet		Modem UART		Debug UART	

Example – using Network Devices

- Control LEDs from a remote PC or another board
- Example implementations of TCP and UDP



Evaluation Boards with LED Switch Client



LEDSwitch Utility
C++ PC application with source code

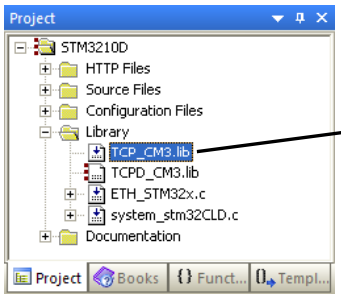
Example – using a HTTP Server

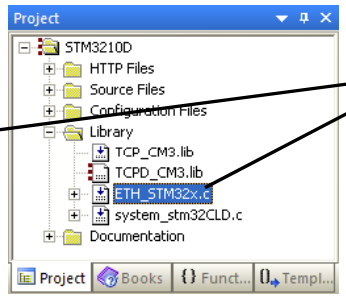
- Access the board from a browser
  - Control LEDs & LCD
  - View board status, switches, & inputs
- TCPnet includes a HTTP server
  - Typically used to host websites
  - Also provides a web-style interface to your application
- C interface to CGI scripts

```
/* Parse all returned parameters. */
dat = http_get_var (dat, var, 40);
if (var[0] != 0) {
   /* Parameter found, returned string is non 0-length. */
   if (str_scomp (var, "led0=on") == __TRUE) {
      P2 |= 0x01;
   }
   else if (str_scomp (var, "led1=on") == __TRUE) {
      P2 |= 0x02;
```

Using TCPnet to enable Ethernet

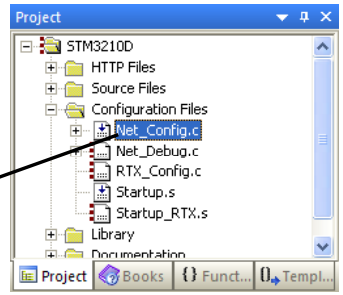
- Two items must be added to the project
  - Both supplied with RL-ARM



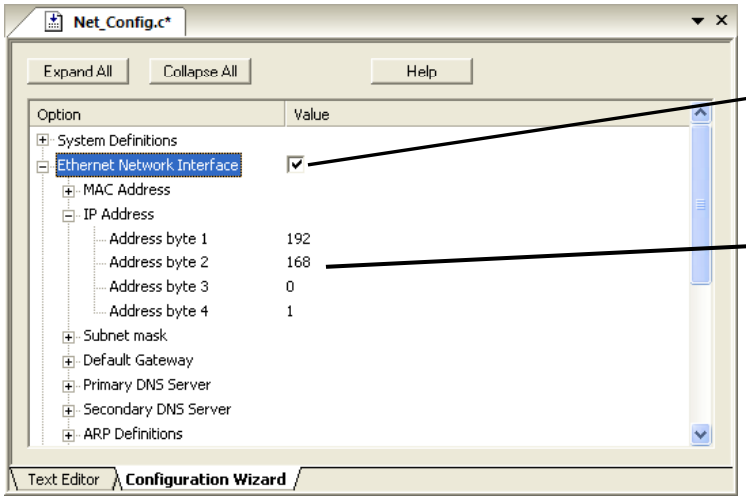


Library containing TCP stack and hardware drivers

Configuration settings for network components



- Ethernet can be enabled and parameters selected graphically

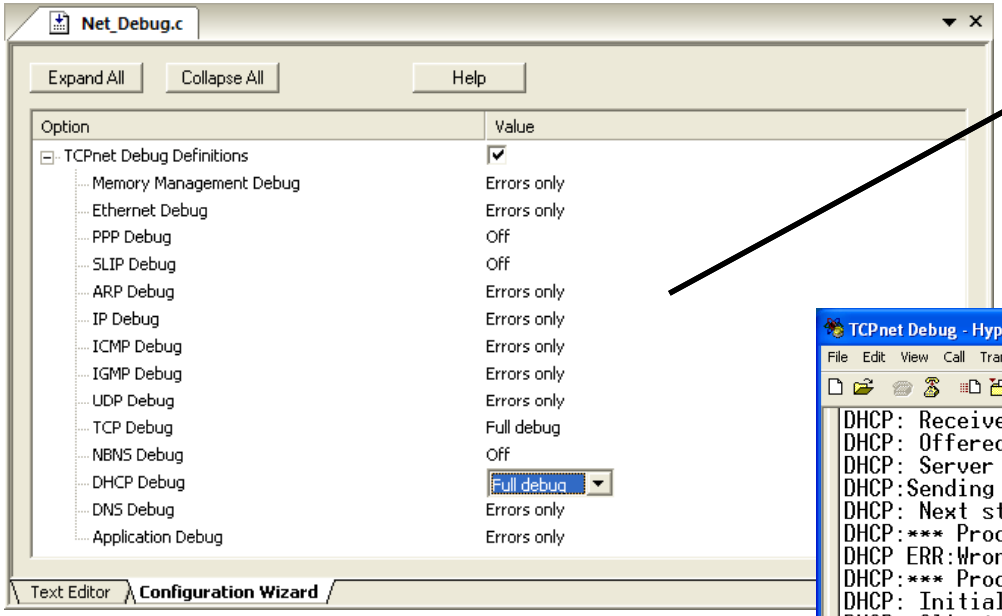


Check-boxes enable desired network components

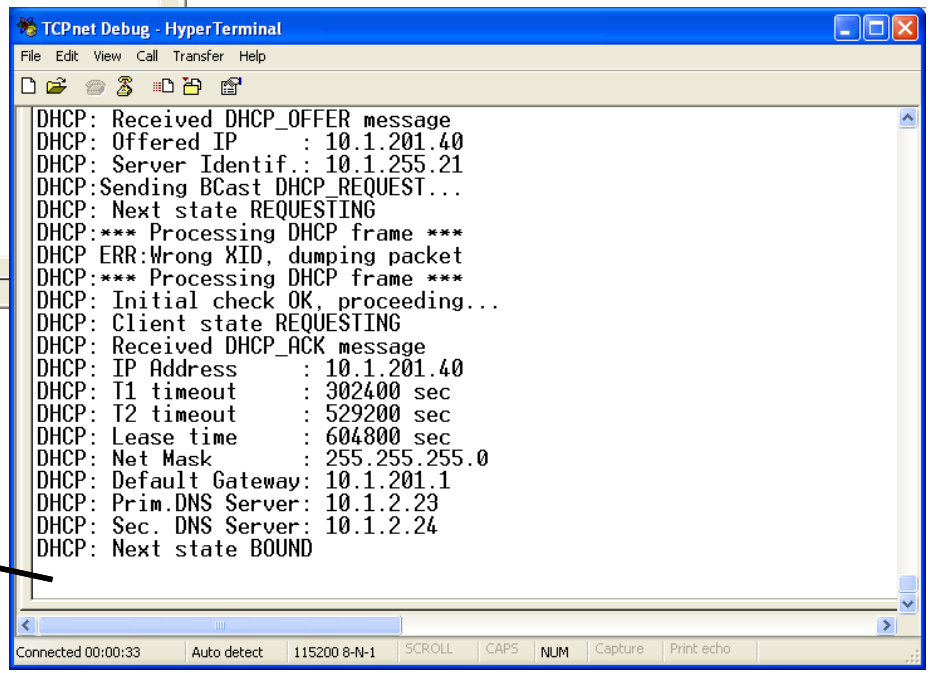
Configurable options instantly accessible via configuration wizard

TCPnet built-in Debug Support

- TCPnet provides optional debug information



Control the debug level
for each network component



View network activity via log
files or terminal window

TCPnet Performance

	UDP		TCP	
Packet size	Packets / sec	kByte / sec	Packets / sec	kByte / sec
10	19,790	176	7,540	74
200	21,370	4,164	6,450	1,272
400	17,490	6,820	5,600	2,202
600	14,230	8,330	4,730	2,782
800	11,950	9,360	4,210	3,300
1000	10,370	10,090	3,736	3,652
1200	9,120	10,670	3,322	3,894
1400	8,140	11,130	3,082	4,215

Examples shown using Cortex M3 device at 96MHz, 100 Mbps full duplex

Using CMSIS compatible Ethernet drivers

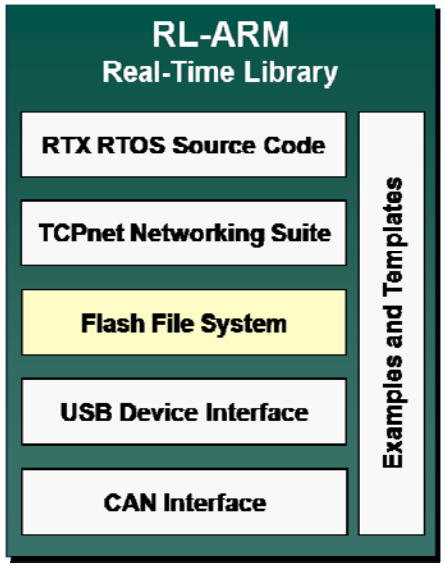
TCPnet footprint – 5 sockets enabled

Demo Example	Total ROM Size	Total RAM Size
HTTP Server (without RTX Kernel)	41,984 Bytes	20,112 Bytes
HTTP Server (with RTX Kernel)	45,240 Bytes	21,776 Bytes
Telnet Server	22,312 Bytes	20,112 Bytes
TFTP Server	34,996 Bytes	24,320 Bytes
SMTP Client	16,736 Bytes	19,600 Bytes
LED Switch Server	11,220 Bytes	19,568 Bytes
LED Switch Client	15,328 Bytes	19,576 Bytes
DNS Resolver	15,328 Bytes	19,776 Bytes

- HTTP Server: Web Server supporting dynamic Web pages and CGI Scripting
- Telnet Server: with command line interface, authorization etc
- TFTP Server: for uploading files (for example Web pages to a Web Server)
- SMTP Client: for sending automated emails
- LED Switch Server and Client: shows basic TCP/IP and UDP communication
- DNS Resolver: used to resolve IP address from the host name
- Further TCP sockets require an approximate 2kB additional space

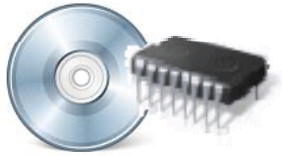
RL-Flash

Flash File System



Flash File System (RL-Flash)

- Enables industry-standard file system compatibility
  - Accessed via standard I/O function calls
- Two file system implementations provided
  - Small & fast file system for internal RAM and ROM
  - FAT32/16/12 for external storage – SPI Flash & SD/MMC cards



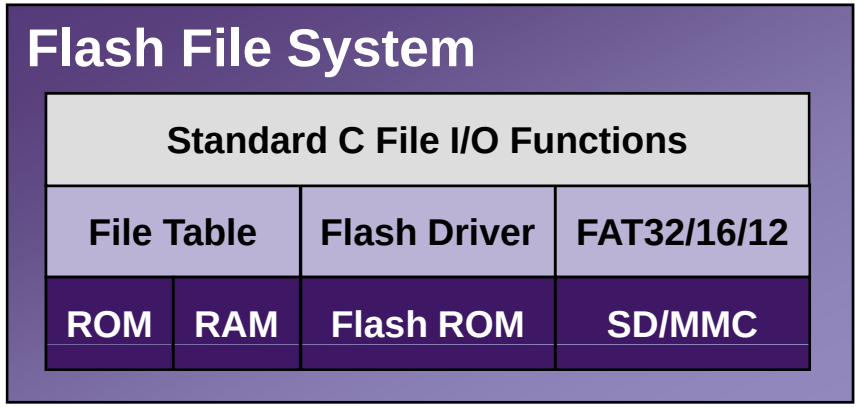
ROM/RAM, data access

SD/MMC, storage

Sub-directories, folder support

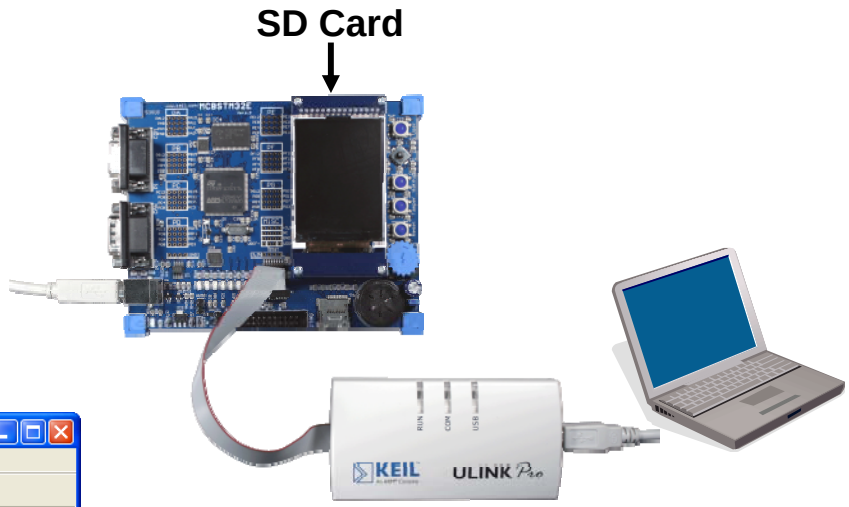
```
Directory of C:\tmp

27/05/2009  08:52    <DIR>                    .
27/05/2009  08:52    <DIR>                    ..
27/05/2009  08:52                14 LONGFI~1.TXT Long Filename.txt
               1 File(s)             14 bytes
               2 Dir(s)  143,698,612,224 bytes free
```

8.3 and long filenames, royalty-free option available



RL-Flash Example

- Standard file I/O with SD Card
- Command line interface
- Interfaces with UART or RTX



STM3210D - HyperTerminal

File Edit View Call Transfer Help

```
+-----------------------------------------------------------------+
|                SD/MMC Card File Manipulation example            |
+ command -------------+ function ---------------------------------+
| CAP fname [/A]       | captures serial data to a file           |
|                      |  [/A option appends data to a file]      |
| FILL fname [nnnn]    | create a file filled with text           |
|                      |  [nnnn - number of lines, default=1000]  |
| TYPE fname           | displays the content of a text file      |
| REN fname1 fname2    | renames a file to 'fname2'               |
| COPY fin [fin2] fout | copies a file 'fin' to 'fout' file       |
|                      |  ['fin2' option merges 'fin' and 'fin2']|
| DEL fname            | deletes a file                           |
| DIR [mask]           | displays a list of files in the directory |
| FORMAT [label]       | formats Flash Memory Card                |
| HELP  or  ?          | displays this help                       |
+----------------------+------------------------------------------+


Cmd>
```

Disconnected | Auto detect | 115200 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

STM3210D - HyperTerminal

File Edit View Call Transfer Help

```
Cmd> dir

File System Directory...
M1.TXT                    1.000.014   01.11.2006  12:00
M2.TXT                    2.000.028   01.11.2006  12:00
M4.TXT                    4.000.056   01.11.2006  12:00
M8.TXT                    8.000.112   01.11.2006  12:00
M16.TXT                  16.000.224   01.11.2006  12:00
M32.TXT                  32.000.448   01.11.2006  12:00
M64.TXT                  64.000.896   01.11.2006  12:00
        7 File(s)       127.001.778 bytes
                        863.469.568 bytes free.

Cmd>
```

Disconnected | Auto detect | 115200 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

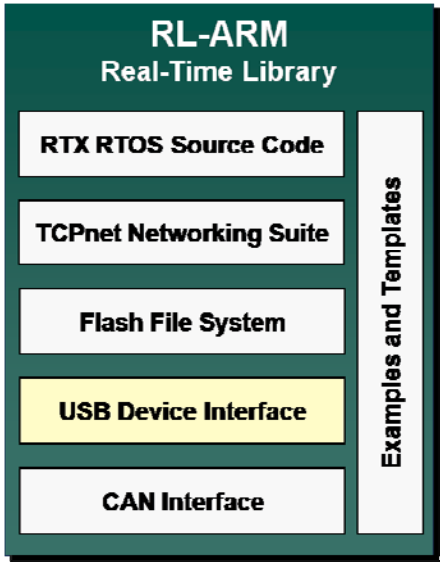
RL-Flash Performance

Board	Device	CPU Core	CPU [MHz]	Card Interface	Write [KB/s]	Read [KB/s]
MCBSTM32	ST STM32	Cortex-M3	72	SPI at 18MHz	711.1	758.1
LM3S8962	Luminary LM3S8962	Cortex-M3	50	SPI at 12.5MHz	537.8	607.6
LM3S6965	Luminary LM3S6965	Cortex-M3	50	SPI at 12.5MHz	539.2	603.6
LM3S3768	Luminary LM3S3768	Cortex-M3	50	SPI at 12.5MHz	539.5	603.8
AT91SAM9G 20-EK	Atmel AT91SAM9G20	ARM9	99	SD4 at 25MHz	4,083.8	5,403.7
MCB2400	NXP LPC2468	ARM7	48	SD4 at 24MHz	4,084.3	5,525.9
MCB2300	NXP LPC2368	ARM7	48	SD4 at 24MHz	3,946.3	5,330.6
MCB2140	NXP LPC2148	ARM7	60	SPI at 7.5MHz	299.4	313.4
MCBSTR9	ST STR912	ARM9	48	SPI at 12MHz	355.2	357.1
MCBSTR750	ST STR750	ARM7	60	SPI at 15MHz	402.2	416.1

Figures shown were achieved working with 4MB of data in 4KB blocks

RL-USB

USB Device Interface



USB Device Interface (RL-USB)

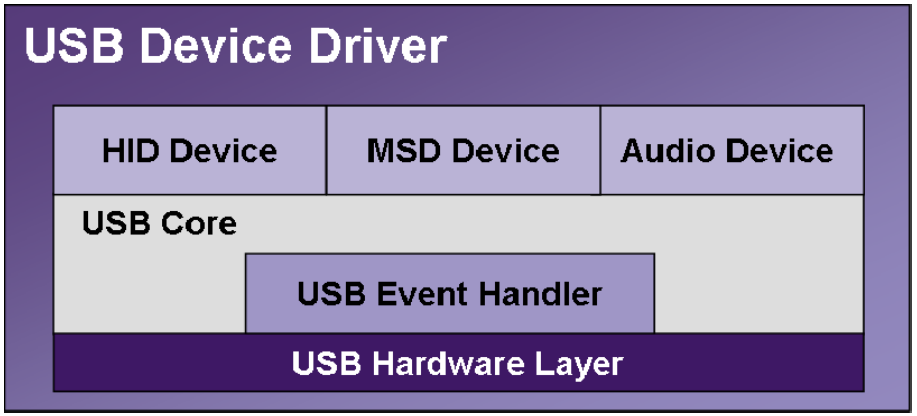
- Offers plug-and-play compatibility for your design
- Enables interfaces for standard USB device classes
  - Uses native drivers provided for Windows 2000/XP/Vista

Human interface devices

Audio, entertainment & communications

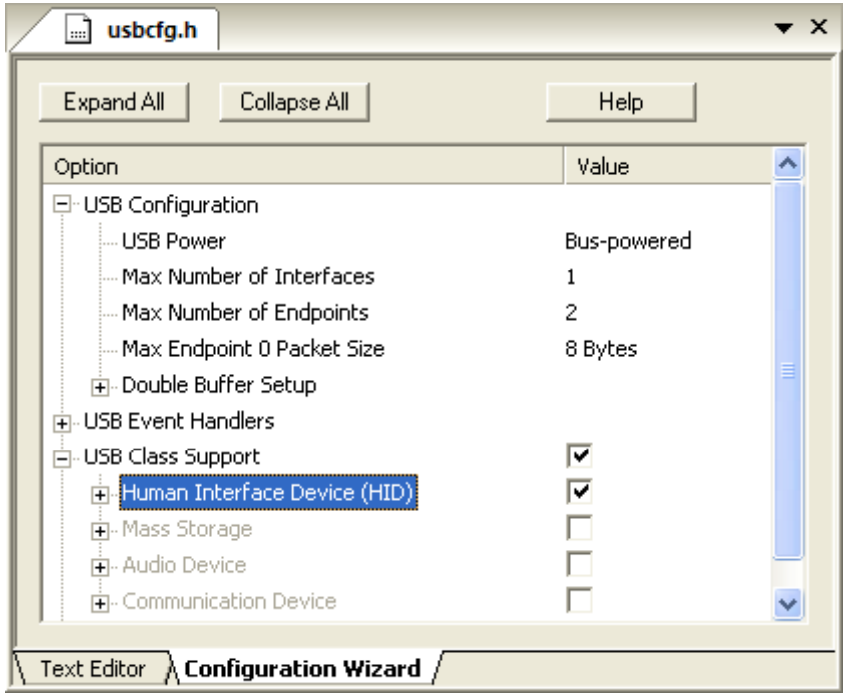
Mass storage, drives, cameras...

Comms devices, telephone modems...



RL-USB Configuration

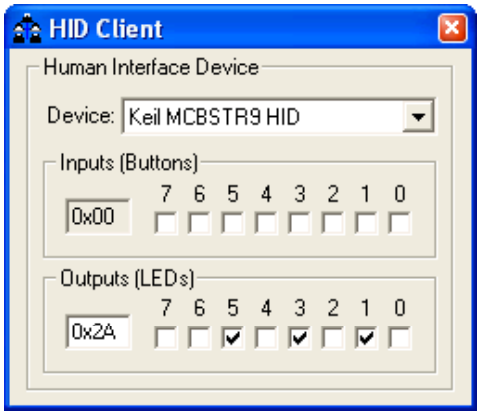
- Device configuration settings are easy to access
  - User must select the appropriate settings for their device
- Start with a standard USB template
  - Adjust USB Core Parameters
  - Update the Device Descriptors
  - Extend the USB Event Handlers
- Composite devices
  - Single device with multiple functions
    - e.g. keyboard with mousepad
  - Configure each function in turn
  - Implement USB Class Code
  - Add USB Class Code from the related USB Template
  - Re-assign USB Event Handlers



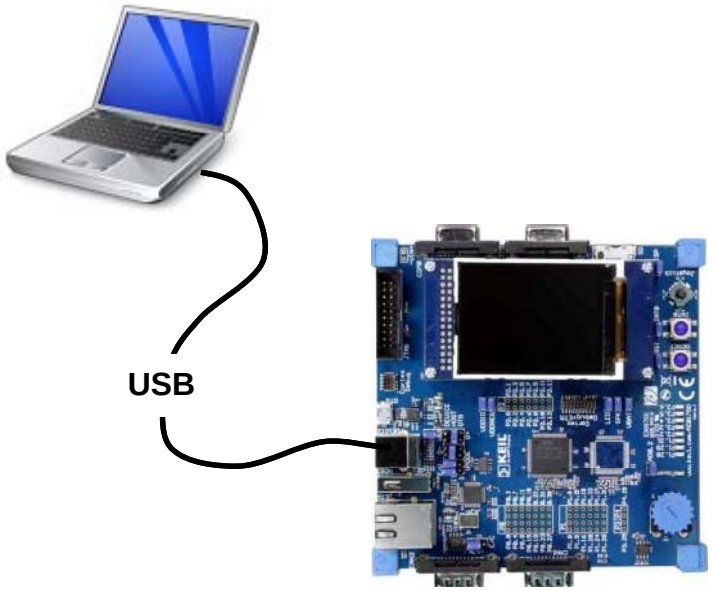
USB Configuration using the µVision Configuration Wizard

RL-USB Example – HID Template

- Human Interface Device
  - Connects to PC without driver
  - LEDs controlled from PC application
  - Switches reported to PC application



HID Client Application supplied with source code



- Example USB templates include:

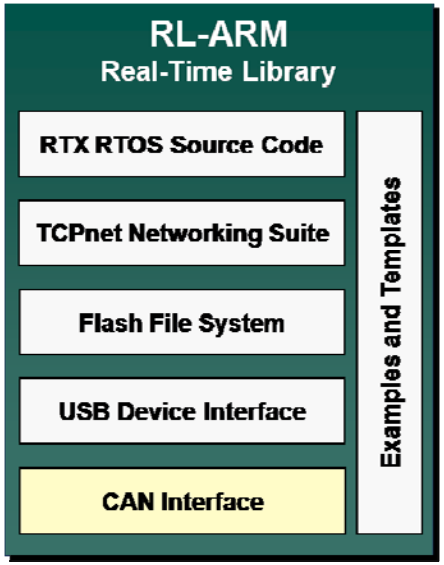
Audio, PC speaker

Storage, memory stick

CDC, virtual COM port

RL-CAN

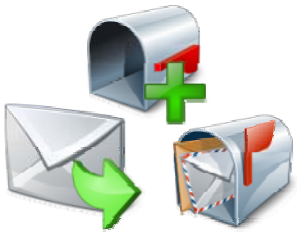
CAN Interface



CAN Interface (RL-CAN)

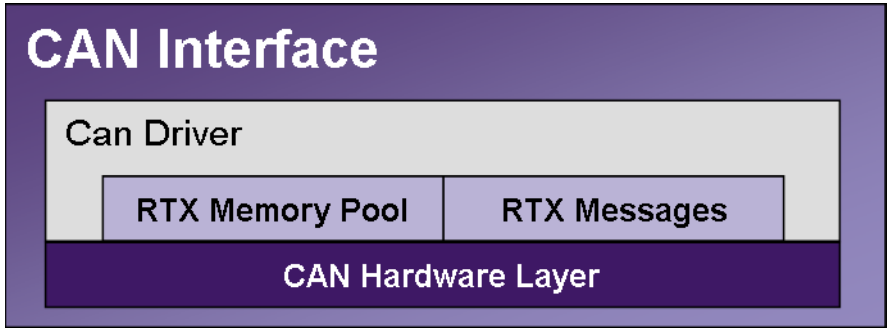
- Generic CAN driver with hardware adaptations
  - Interrupt-driven hardware layer
  - Supports several ARM-based microcontrollers
- Common API for access to many CAN controllers
  - Including Atmel, NXP, ST, Luminary, TI, and Toshiba

Configure and initialize devices



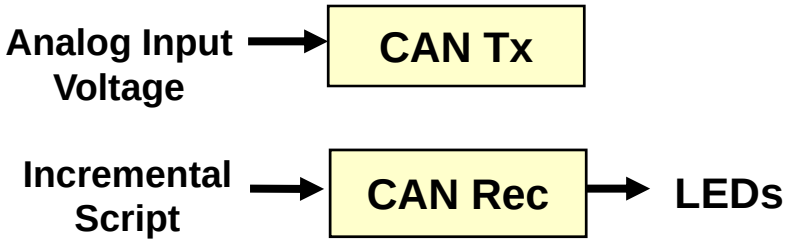
Send, request and receive messages

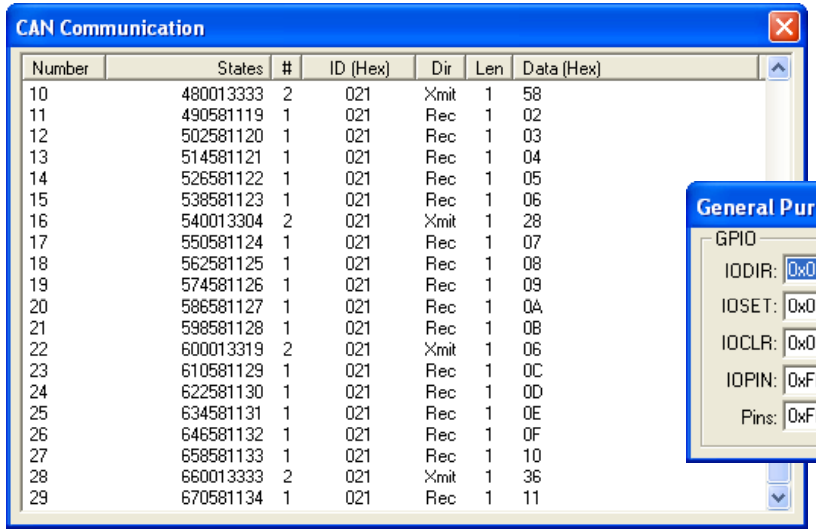
- Implemented using RTX Kernel
  - Memory Pool
  - Message Passing



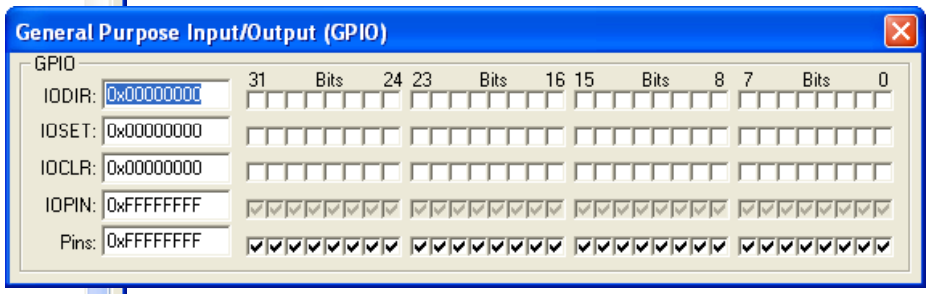
RL-CAN Examples

- Hardware
  - A/D Converter gets input voltage from Potentiometer
  - Input Voltage sent every second via CAN2
  - Message received via CAN is shown on LEDs via CAN1
- Using µVision Simulation
  - Script generates A/D input voltage
  - Messages received via CAN2





Number	States	#	ID (Hex)	Dir	Len	Data (Hex)
10	480013333	2	021	Xmit	1	58
11	490581119	1	021	Rec	1	02
12	502581120	1	021	Rec	1	03
13	514581121	1	021	Rec	1	04
14	526581122	1	021	Rec	1	05
15	538581123	1	021	Rec	1	06
16	540013304	2	021	Xmit	1	28
17	550581124	1	021	Rec	1	07
18	562581125	1	021	Rec	1	08
19	574581126	1	021	Rec	1	09
20	586581127	1	021	Rec	1	0A
21	598581128	1	021	Rec	1	0B
22	600013319	2	021	Xmit	1	06
23	610581129	1	021	Rec	1	0C
24	622581130	1	021	Rec	1	0D
25	634581131	1	021	Rec	1	0E
26	646581132	1	021	Rec	1	0F
27	658581133	1	021	Rec	1	10
28	660013333	2	021	Xmit	1	36
29	670581134	1	021	Rec	1	11



RL-CAN Virtual Simulation Registers

- µVision provides VTREGs
  - Allows control of communication (CAN, I2C, SSP, SPI)
  - CAN I/O can be simulated and scripted using these registers

VTREG	Description
CANxID	Is an 11-bit or 29-bit identifier of the message currently transferred. The ID size is specified with the values in the CAN0IN or CAN0OUT VTREG. For short 11-bit identifiers only the 11 LSB bits are used of this VTREG.
CANxL	The data length of the CAN message. Valid values for CAN0L are 0...8.
CANxB0 ...CANxB7	The data bytes of the CAN message. 8 data bytes are implemented to access the individual message bytes of the current object.
CANxIN	Is set by the user or within debug functions to simulate incoming messages. The following values are possible: ▪ CANxIN = 1 receive the current message using a 11-bit identifier. ▪ CANxIN = 2 receive the current message using a 29-bit identifier. ▪ CANxIN = 3 request a remote frame from the application program with matching 11-bit identifier. ▪ CANxIN = 4 request a remote frame from the application program with matching 29-bit identifier. ▪ CANxIN = 0xFF simulate BUSOFF mode of the CAN controller.CANxIN is set to 0 by the simulator when the message has be processed by the µVision3 simulator.
CANxOUT	Is set by the simulator when transmitting a message by the application program. The following values are possible: ▪ CANxOUT = 1 transmit the current message using a 11-bit identifier. ▪ CANxOUT = 2 transmit the current message using a 29-bit identifier. ▪ CANxOUT = 3 request a remote frame with matching 11-bit identifier from the user or debug function. ▪ CANxOUT = 4 request a remote frame with matching 29-bit identifier from the user or debug function.
CANxTIMING	Allows you to set a performance factor that controls the simulated communication timing within µVision3. This performance factor simulates a busy CAN network. With a performance factor of 0 an CAN network with infinite baudrate is simulated. With a factor of 1 the CAN messages are transferred in real-time taking care of the current selected communication baudrate. A factor of 2 simulates the performance that is identical to 50% of the communication baudrate. CANxTIMING is a floating point value that can be between 0 .. 1000.

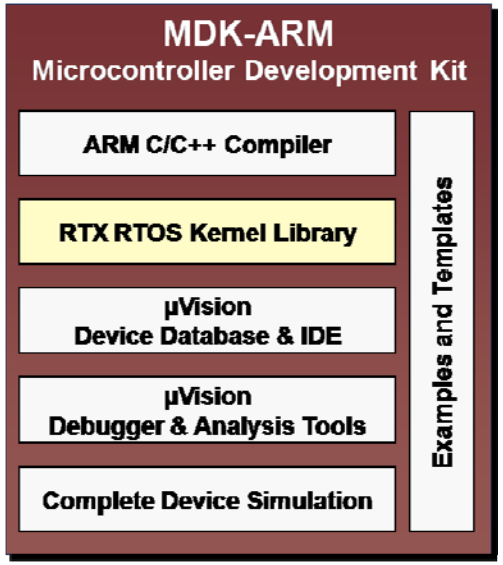
µVision Debug & Signal Functions

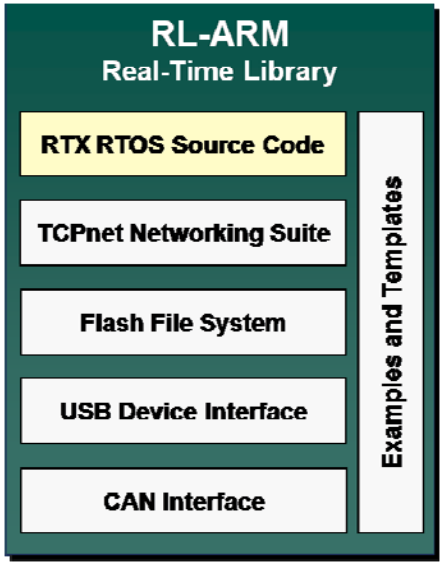
- Users can define and generate input functions as stimuli to simulation models
  - Scripts for CAN Input and Output Messages
  - Signal Functions
  - Automated Message Processing
  - Periodic CAN Messages

```
FUNC void SendCANmessage (void) {
  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
}
```

```
FUNC void Print_CANmessage (void) {
  switch (CAN0OUT) {
    case 1: printf("\nSend 11-bit ID=%X",   CANAID);
   break;
    case 2: printf("\nSend 29-bit ID=%X",   CANAID);
   break;
    case 3: printf("\nRequest 11-bit ID=%X", CANAID);
   return;
    case 4: printf("\nRequest 29-bit ID=%08X",
   CANAID); return;
  }
   printf("\nMessage Length %d, Data: ", CAN0L);
   printf("%X … %X",  CAN0B0, …, CAN0B7);
}
```

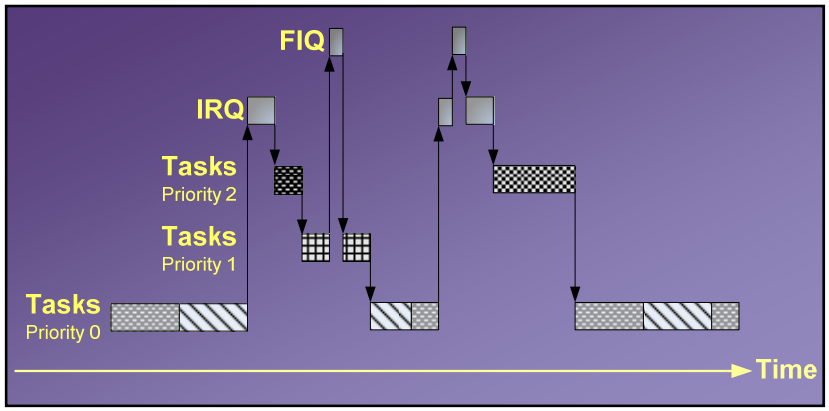
RTX Real-Time Kernel





Software Concepts for ARM

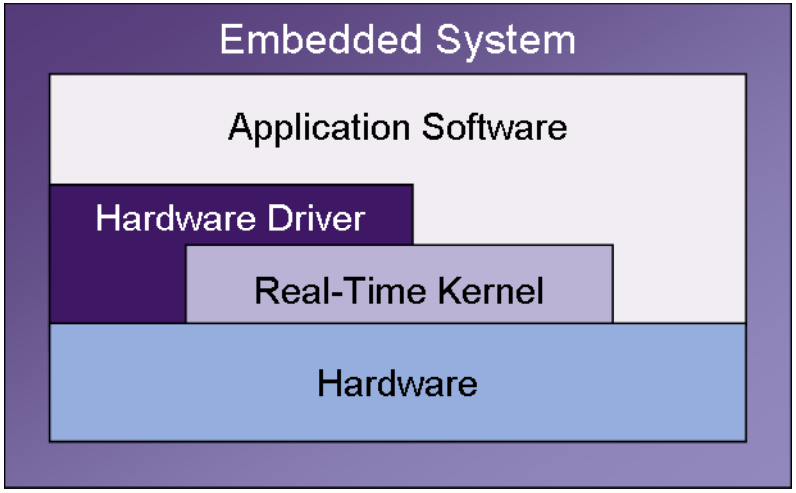
- Embedded applications typically have two design concepts
- ‘main’ as Infinite Loop
  - Each task called from main loop
  - Interrupts perform time-critical jobs
  - Stack usage un-predictable
  - User manages task interactions



- Using a Real-Time Kernel
  - Allows application to be separated into independent tasks
  - Message passing eliminates critical memory buffers
  - Each task has its own stack area
  - Interrupt communication with event flags and messages

Why use a Real-Time Kernel?

- Structured framework for embedded applications
  - Hardware interface layer
  - Easy expansion of system software
  - Hardware independent
- Housekeeping
  - Process scheduling
  - CPU resource management
  - Task communication



- Focus on Application Development
  - Leave basic system management to the RTOS kernel
  - Avoid re-writing resource management code that already exists
  - Reduce porting and testing overheads

What makes a Good RTOS?

- Performance
  - Predictable behaviour
  - Low latency
  - High number of interrupt levels
- Ease of Use
  - Flexible API and implementation
  - Tool-chain integration
  - Scheduling options
    - Multitasking, Pre-emptive, Round Robin
- System Friendly
  - Consumes small amount of system resource
  - Proven kernel
  - Low cost

Real-Time?

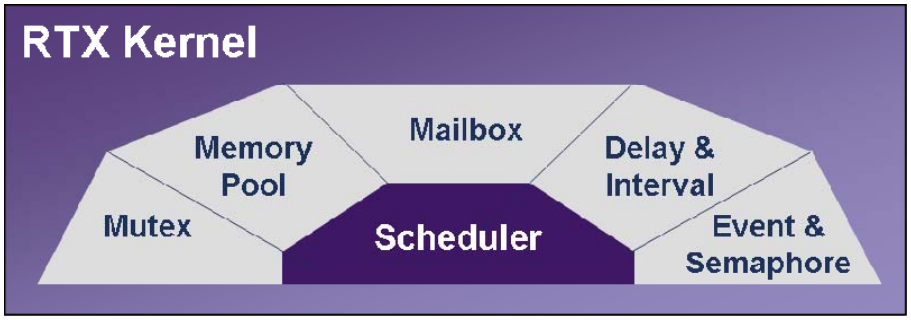
- Real-Time does not simply mean High Speed
  - Not all tasks are ‘Most Urgent’
  - Tasks need to complete before deadline and other tasks
  - Real-Time OS not to be confused with high-speed requirements
- Real-Time, not mission critical
  - Varying levels of Real-Time
    - RTOS not confined to critical systems
  - Deterministic behaviour is often most important

- A Real-Time OS is a framework
  - RTOS provides good multitasking environment
  - Reliable and scalable management of housekeeping tasks

RTX Real-Time Kernel

- Full-featured Real-Time kernel designed to meet the challenges of Embedded System Design

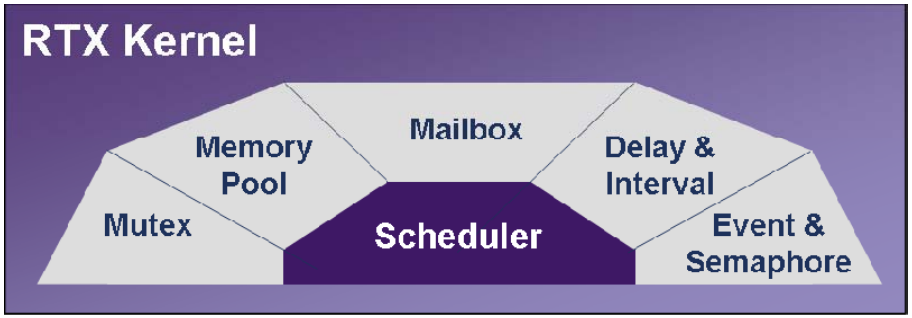
- Process Management
  - Create and delete tasks
  - Change task priorities
  - Event flag management
  - Interrupt functions
  - CPU resources
- Multi-Tasking
  - Pre-emptive context switching
  - Scheduling
  - Semaphore management
- Real-Time Control
  - Deterministic behaviour



- Inter-task Communication
  - Mailbox management
  - Interface to interrupt functions
- Memory Allocation
  - Thread-safe (usage even in ISR)

RTX Real-Time Kernel

- Full-featured real-time kernel for embedded systems
- Process Management
  - Create and delete tasks, change task priorities
  - Manage event flag and CPU resources
- Multi-Tasking
  - Pre-emptive context switching, scheduling, and semaphores
- Real-Time Control
  - Deterministic behaviour
- Inter-task communication
  - Mailbox management
  - Interface to interrupt functions
- Memory allocation
  - Thread-safe (usage even in ISR)



RTX Specifications

- Provides all real-time kernel requirements
  - Multi-Tasking – Round Robin, Pre-emptive, Cooperative
  - Unlimited – User Timers, Semaphores and Mailboxes
  - Royalty free

Task Specifications	
Priority Levels	256
No. of Tasks Defined	Unlimited
No. of Tasks Active	256
Context Switch	< 300 Cycles
Interrupt Latency	< 100 Cycles

Memory Requirements	Bytes
CODE Space (depending on used functionality)	1.5K – 5K
RAM Space (each active task requires its own stack space)	< 500

RTX Performance

Task Specifications	ARM7TDMI	Cortex-M3
CPU Clock Speed	60MHz	72MHz
Initialize system, start task	46.2µS	22.1µS
Create defined task, (no task switch)	17.0µS	8.1µS
Create defined task, (with task switch)	19.1µS	9.3µS
Delete Task	9.3µS	4.8µS
Task Switch	6.6µS	3.9µS
Set event (no task switch)	2.4µS	1.9µS
Send semaphore	1.7µS	1.6µS
Send message	4.5µS	2.5µS
Max Interrupt lockout for IRQ ISR's	3.1µS	n/a

Enabling RTX in MDK-ARM

C:\Keil\ARM\RL\RTX\Examples\Measure\example.c

```
311  *----------------------------------------------------------------------
312 int main (void) {                          /* program exec. starts here   */
313   initialise();                            /* init                        */
314   while (1) {                              /* loop forever                */
315     task = get_response();
316     switch (task){
317       case 'M': measure_task();
318       break;
319
320       case 'S': save_measurement
321       break;
```

C:\Keil\ARM\RL\RTX\Examples\Measure\exampleRTOS.c

```
309 /*----------------------------------------
310  *        Main: Initialize and start RTX
311  *----------------------------------------
312 int main (void) {                          /* p
313   os_sys_init_prio (init_task,50);         /* i
314 }
315
316 /*----------------------------------------
317  * end of file
318  *----------------------------------------
319
```

Infinite while loop in main() is replaced by an OS initialisation call

Core application duties are defined as RTOS tasks

Graphical configuration of RTOS settings

C:\Keil\ARM\RL\RTX\Examples\Measure\example.c

```
064
065 void init_task (void);                     /* Function prototypes */
066 void clock_task (void);
067 void get_escape (void);
068 void command_task (void);
    void measure_task (void);
    void interval_task (void);

    /*----------------------------------------
     *                Save curre
     *
```

C:\Keil\ARM\RL\RTX\Examples\Measure\exampleRTOS.c

```
064
065 __task void init_task (void);                /* !
066 __task void clock_task (void);
067 __task void get_escape (void);
068 __task void command_task (void);
069 __task void measure_task (void);
070 __task void interval_task (void);
071
072 /*----------------------------------------
073  *                Save current measurements in
074  *----------------------------------------
```

C:\Keil\ARM\RL\RTX\Examples\Measure\RTX_Config.c

Expand All | Collapse All | Help

Option	Value
Task Definitions	
Number of concurrent running tasks	6
Number of tasks with user-provided stack	1
Task stack size [bytes]	2
Check for the stack overflow	
Number of user timers	0
System Timer Configuration	
RTX Kernel timer	Timer
Timer clock value [Hz]	12000
Timer tick value [us]	4000
Round-Robin Task switching	

Define max. number of tasks that will run at the same time. Default: 6

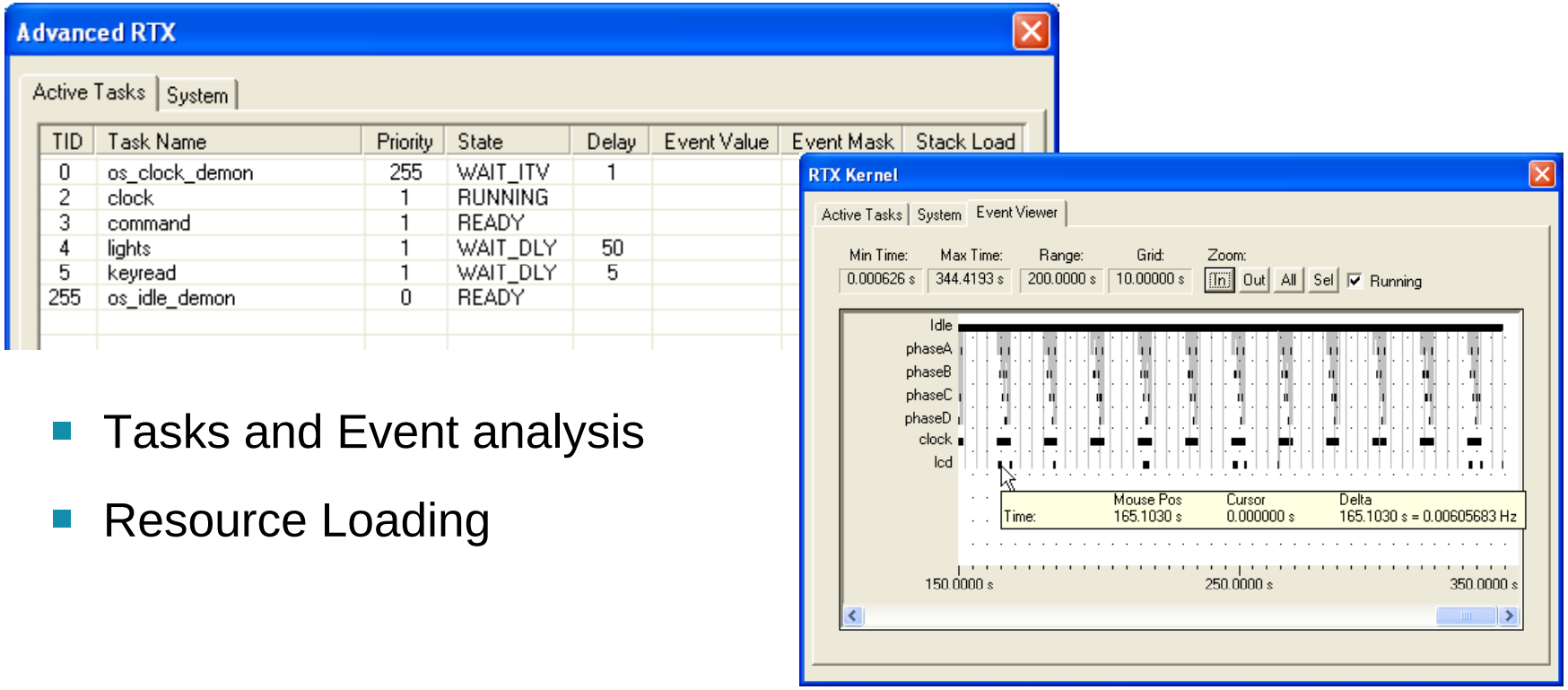
Project Workspace
- Simulator
  - Main Files
  - Configuration Files
    - RTX_Config.c
    - Startup.s
  - Documentation

Text Editor | Configuration Wizard

KEIL Tools by ARM

Kernel Aware Debugging

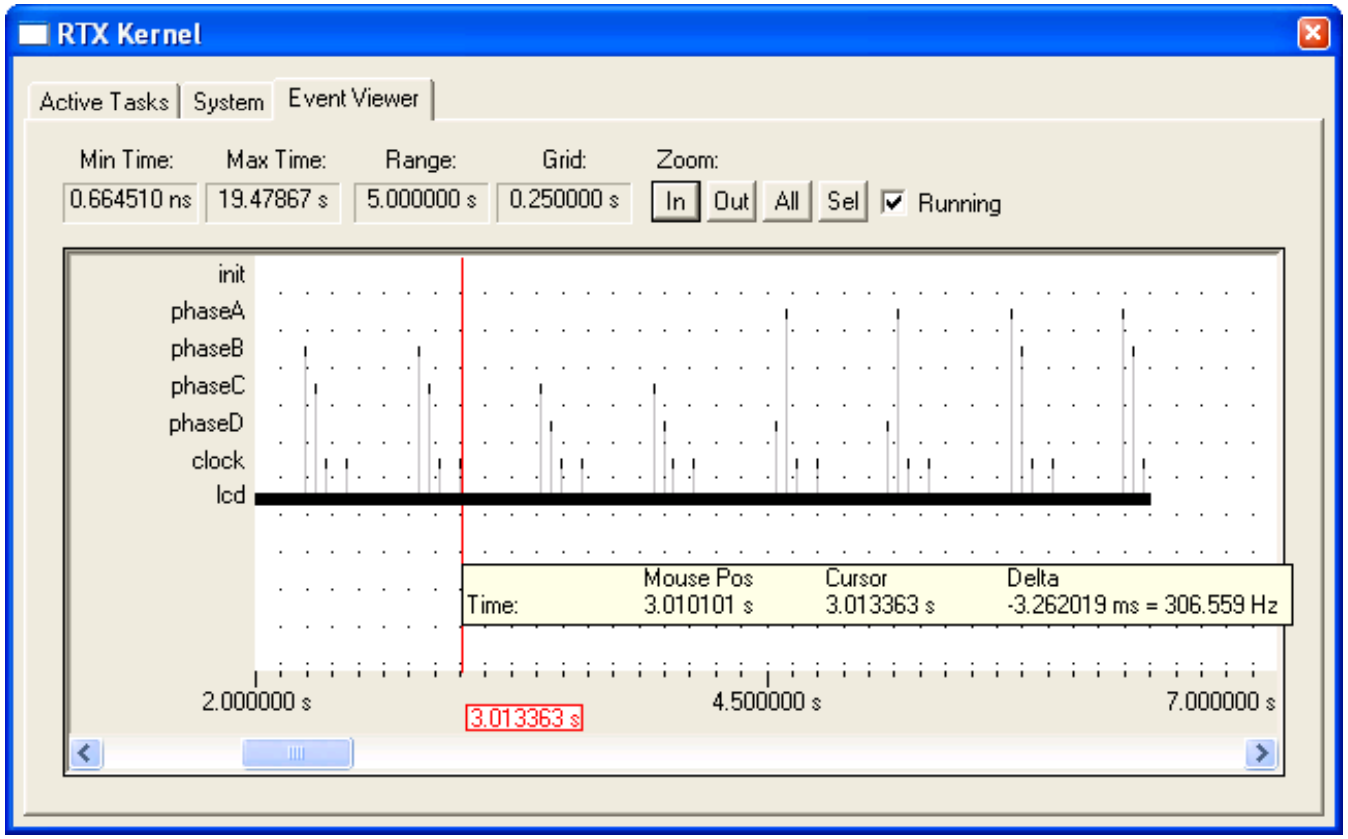
- RTX and µVision are tightly integrated
  - Kernel status information is easily visible



  - Tasks and Event analysis
  - Resource Loading

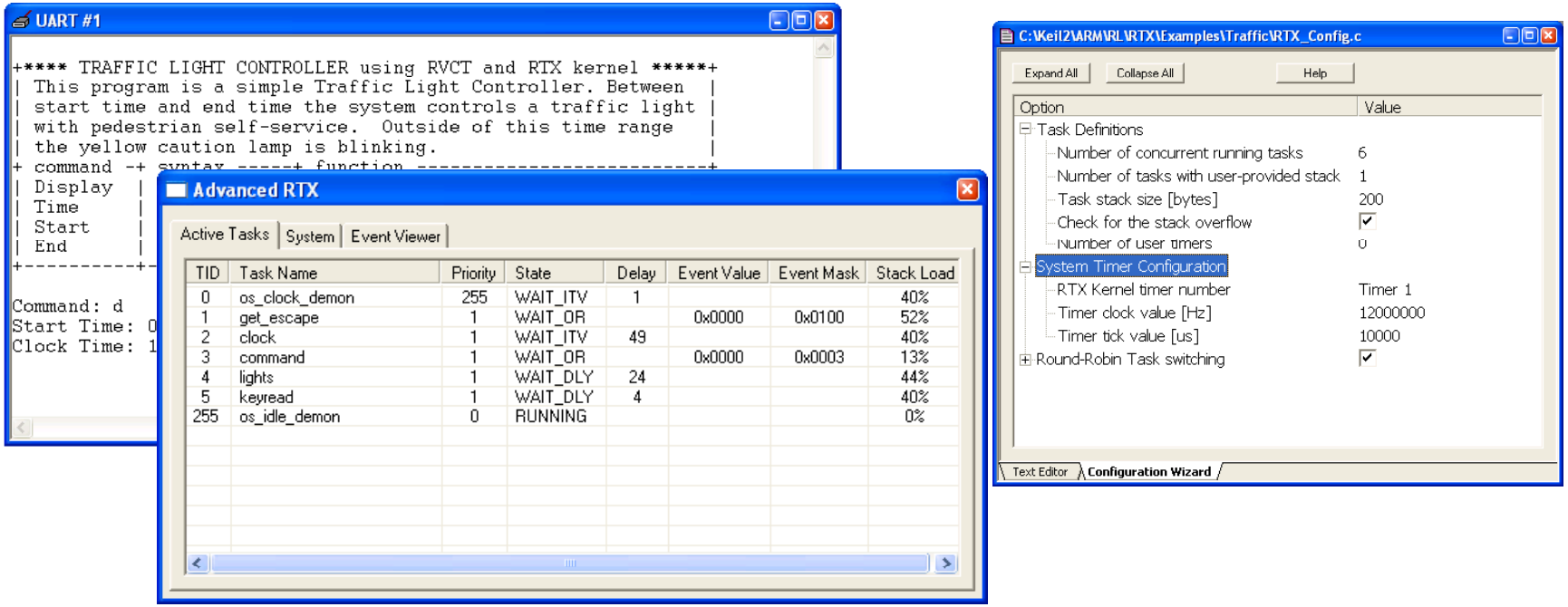
RTX Event Viewer

- Displays task switching and events of a running RTX system
  - Available on running Cortex-Mx devices or using µVision simulation



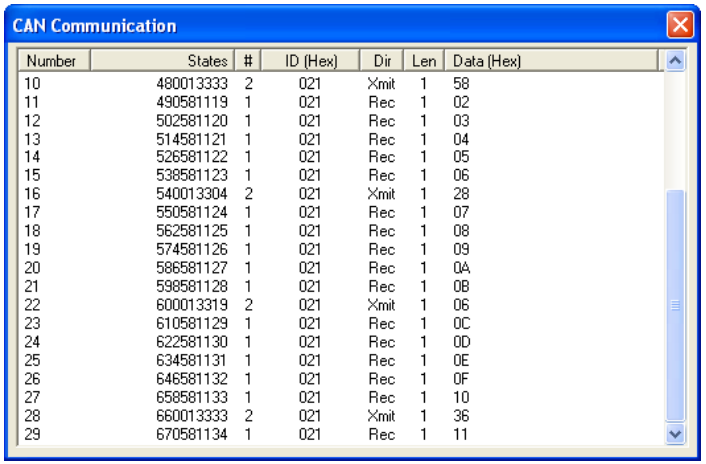
RTX Examples

- Traffic Light
  - LEDs are timed or controlled by push button
  - Uses interrupt control, event management, and multitasking capabilities of RTX Kernel



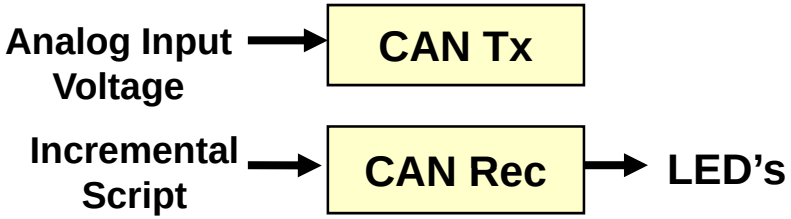
RTX Examples

- CAN Example using RTX
  - Mailbox and event handling
  - CAN Send (Tx) – shows automatic data handling capabilities
  - CAN Rec – message checking with instant message receipt
    – task wait and return
    – almost impossible without Real-Time Kernel



CAN Communication

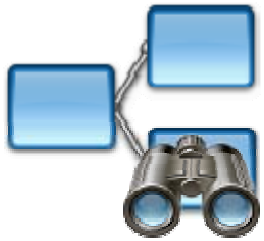
Number	States	#	ID (Hex)	Dir	Len	Data (Hex)
10	480013333	2	021	Xmit	1	58
11	490581119	1	021	Rec	1	02
12	502581120	1	021	Rec	1	03
13	514581121	1	021	Rec	1	04
14	526581122	1	021	Rec	1	05
15	538581123	1	021	Rec	1	06
16	540013304	2	021	Xmit	1	28
17	550581124	1	021	Rec	1	07
18	562581125	1	021	Rec	1	08
19	574581126	1	021	Rec	1	09
20	586581127	1	021	Rec	1	0A
21	598581128	1	021	Rec	1	0B
22	600013319	2	021	Xmit	1	06
23	610581129	1	021	Rec	1	0C
24	622581130	1	021	Rec	1	0D
25	634581131	1	021	Rec	1	0E
26	646581132	1	021	Rec	1	0F
27	658581133	1	021	Rec	1	10
28	660013333	2	021	Xmit	1	36
29	670581134	1	021	Rec	1	11



How does RL-ARM work for me?

- Develop robust and powerful applications fast
  - The RTX kernel and sources, gives you all the resources you need to create and control multi-threaded, real-time applications that can be tailored to your system.

- Ensure you only do what you have to
  - RL-ARM enables USB, CAN, TCP/IP networking, and file-system support. Use existing resources to ensure you focus on the important parts of your application.

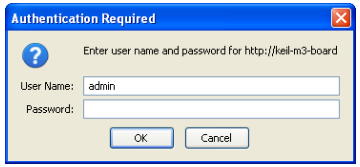


- Take advantage of the expertise of others
  - RL-ARM is designed, tested, and optimized by ARM engineers. Documentation and examples make it easy to re-use the work done by our experts.

New Features coming to RL-ARM

- Next release – September 2009

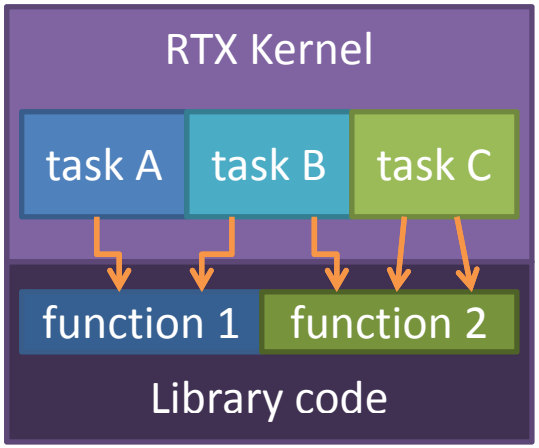
RL-Flash FAT FS will tolerate power-failures



User/admin access control for HTTP login

FTP client and host support

- *Now!*

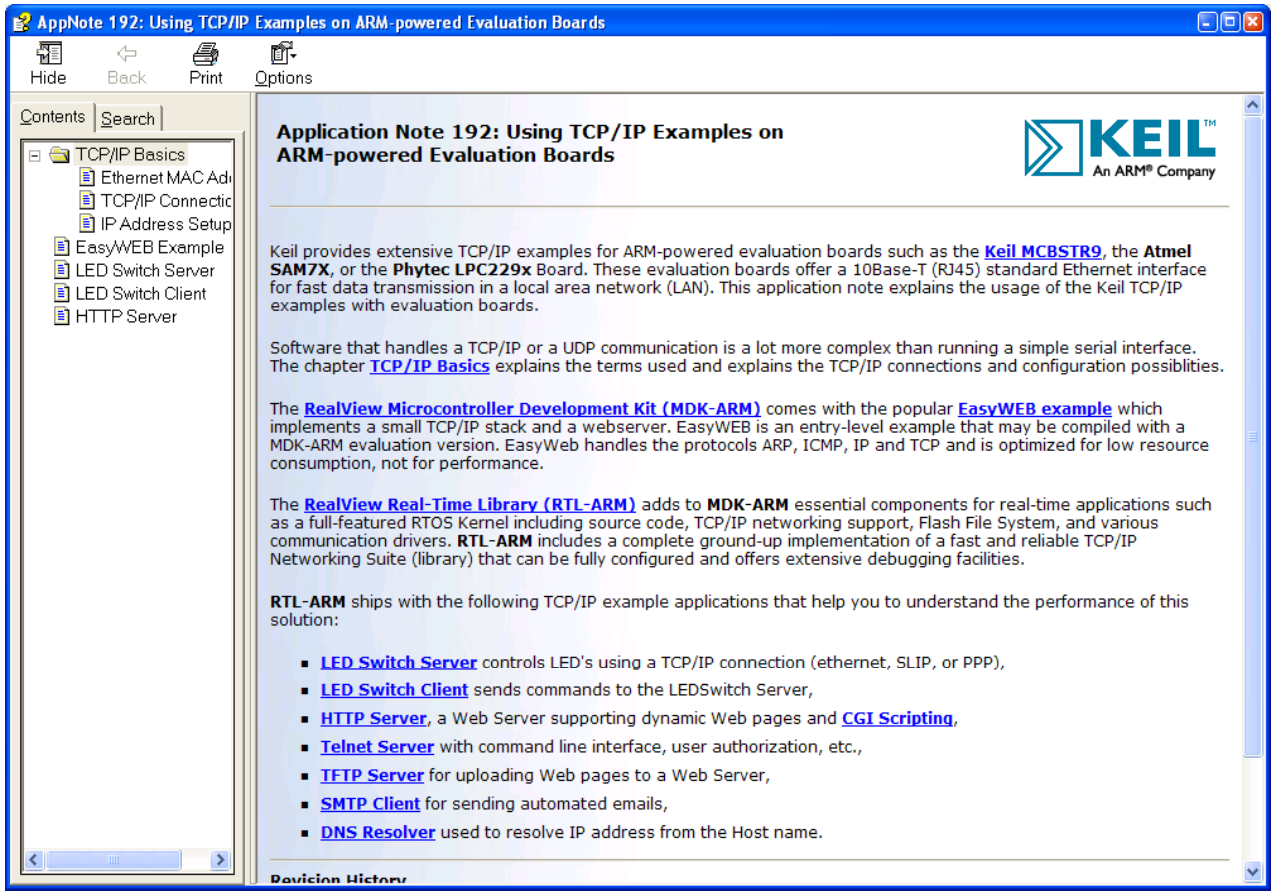


Full thread-safe implementation of all features

- 2010
  - New lightweight graphics library
  - CMSIS compliant components
  - Enhanced USB support – Host, Hub, & OTG

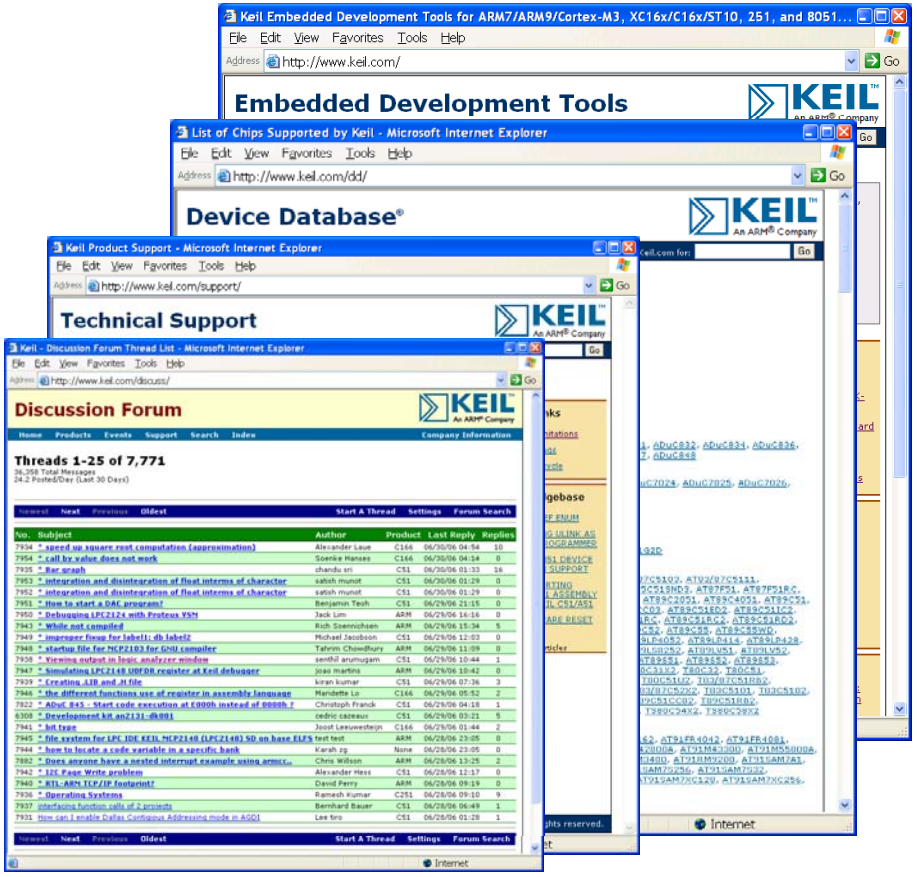
Need More Help?

- **Application Notes on www.keil.com/appnotes**
  - 192: Using TCP/IP Examples on ARM Powered Evaluation Boards
  - 195: Developing HID USB Device Drivers For Embedded Systems



Get More Information

- Customers use www.keil.com on a daily basis to obtain
  - Program examples
  - Latest technical information
  - Application Notes
  - Program Examples
  - Device Database
  - Support Knowledgebase
  - Discussion Forum



Middleware

SEGGER

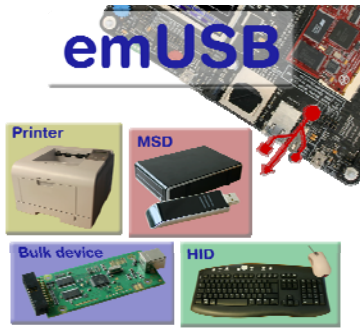
Middleware Product overview

Middleware
emWin
embOS
embOS/IP
emFile
emUSB

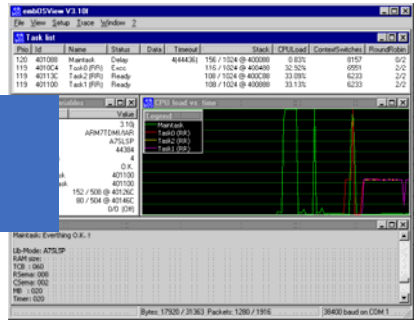
Licensing

GUI (emWin)

USB (emUSB) Host + Device



RTOS (embOS)

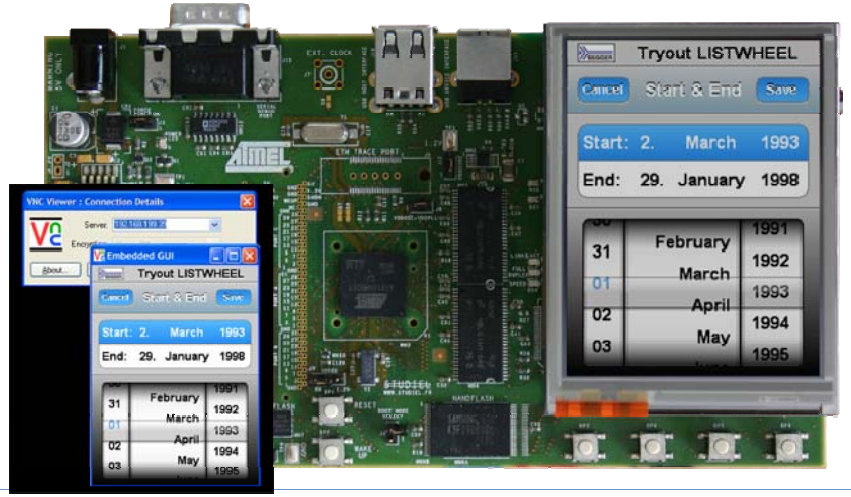


TCP/IP Stack (embOS/IP)

File System (emFile)

emWin®
Graphics Software and GUI

- Any CPU, any LCD, any LCD controller
- Very fast drawing routines
- Small footprint
- Unicode font support
- Font converter available
- Touch screen support
- Simulation included
- Virtual display support
- Multiple layer / multi display support
- Child windows
- Customizable Widgets
- Alpha blending
- Support for transparent windows
- PNG, JPEG support
- VNC Server available

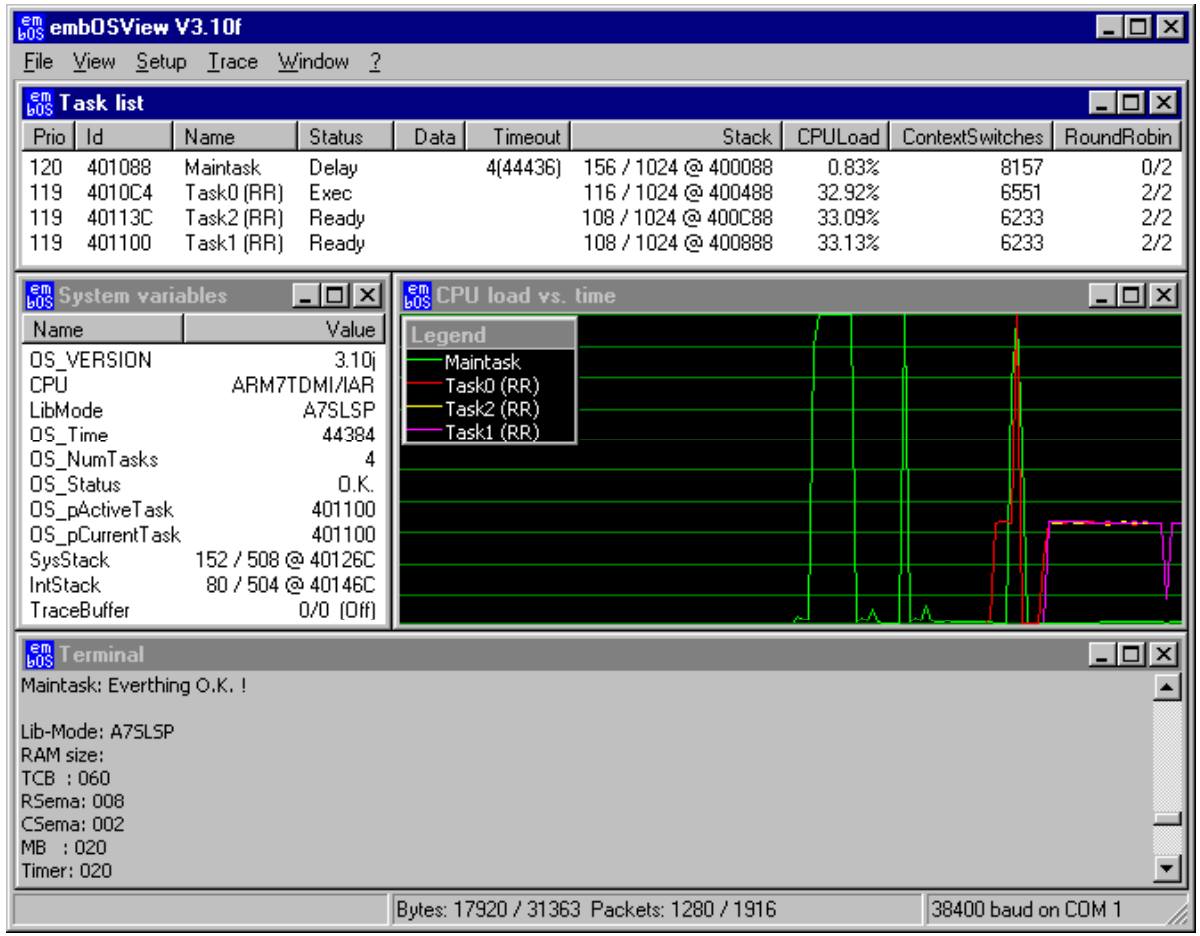


embOS®
real time operating system

- Preemptive scheduling, Round-robin for tasks at same priority
- Unlimited number of tasks
- Zero interrupt latency time
- Small footprint: ROM <3 kB, RAM <1kB (Kernel only, ARM7)
- Very easy to use, No configuration needed
- Start application and projects (BSPs) for an easy start.
- Fully dynamic task creation / termination
- Fast context switch (independent of number of tasks)
- High resolution time measurement
- High precision per task profiling
- Real time kernel viewer (embOSView) included

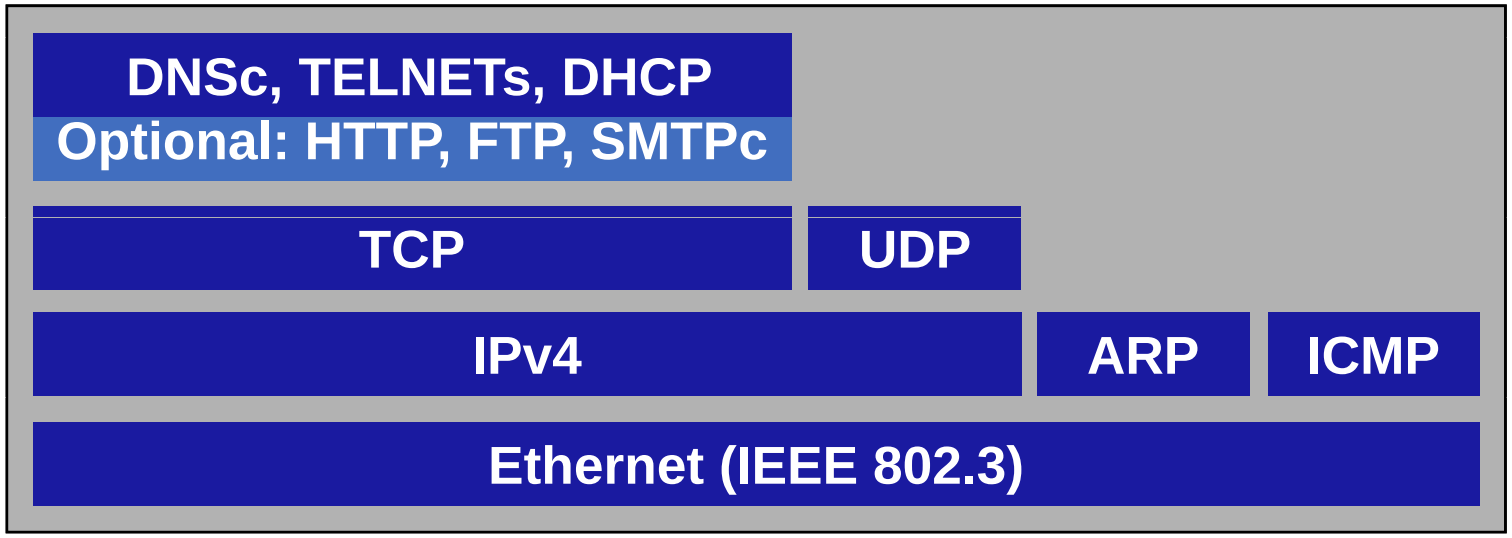
embOS® profiling with embOSView

- Included with embOS
- Real-time profiling
- via UART or JTAG (DCC)



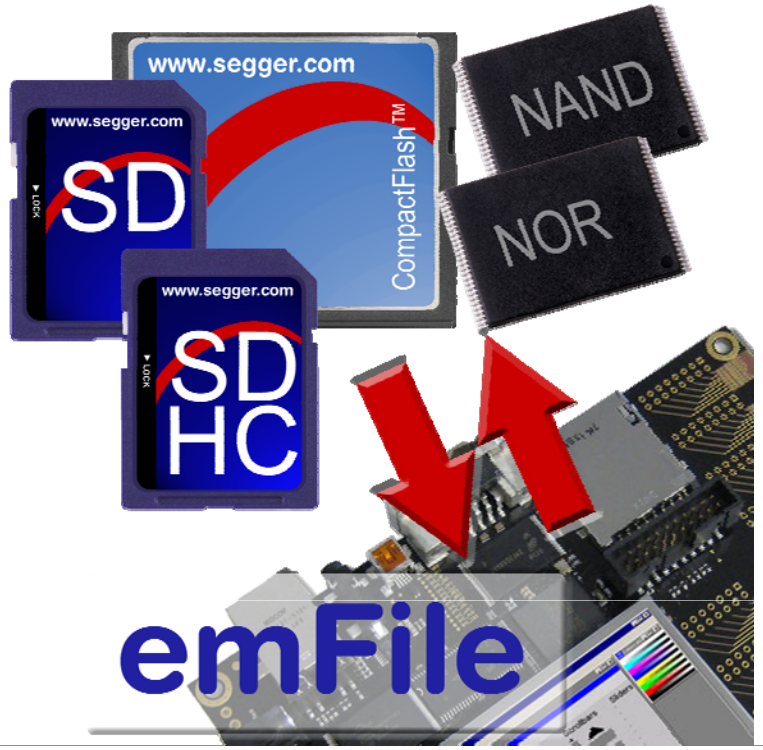
embOS/IP
TCP/IP stack for embOS

- Standard sockets interface
- High performance
- Small footprint
- No configuration required
- Zero data copy for ultra fast performance
- Non-blocking versions of all functions
- Connections limited only by available memory
- Optional drivers for the most common devices are available
- Support for driver side (hardware) checksum computation



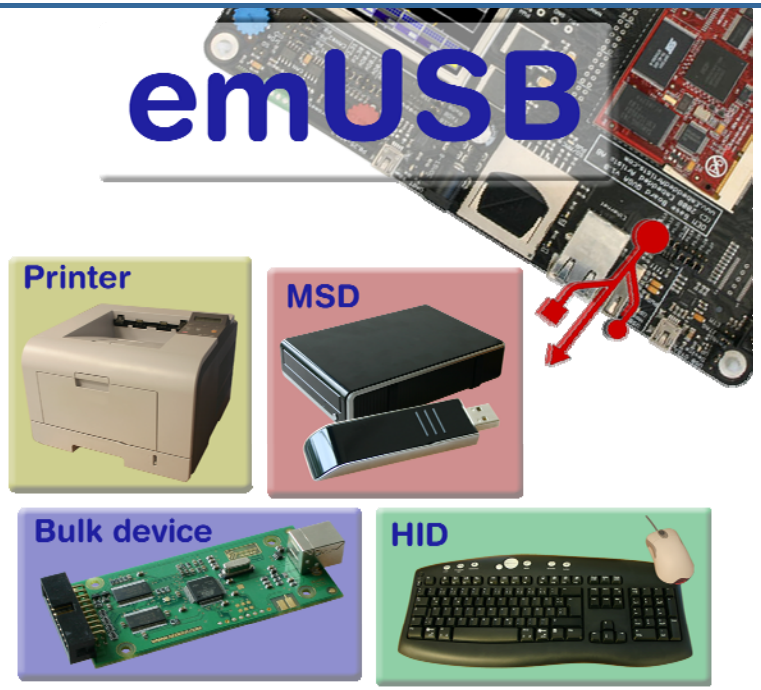
emFile

- Journaling File system for FAT
- Fail safe driver
- Very small footprint, starting at 8KB ROM / 1,6 KB RAM for a system using RAM disk driver
- Ready to use device drivers for: Media Cards (CF, SD, MMC), IDE and NOR/NAND flash
- Ready to use sample projects
- Highly configurable
- Complete source code included
- Any CPU can be used
- Portable OS support
- Check Disk function
- FAT 12, FAT 16 and FAT 32 support



emUSB host stack

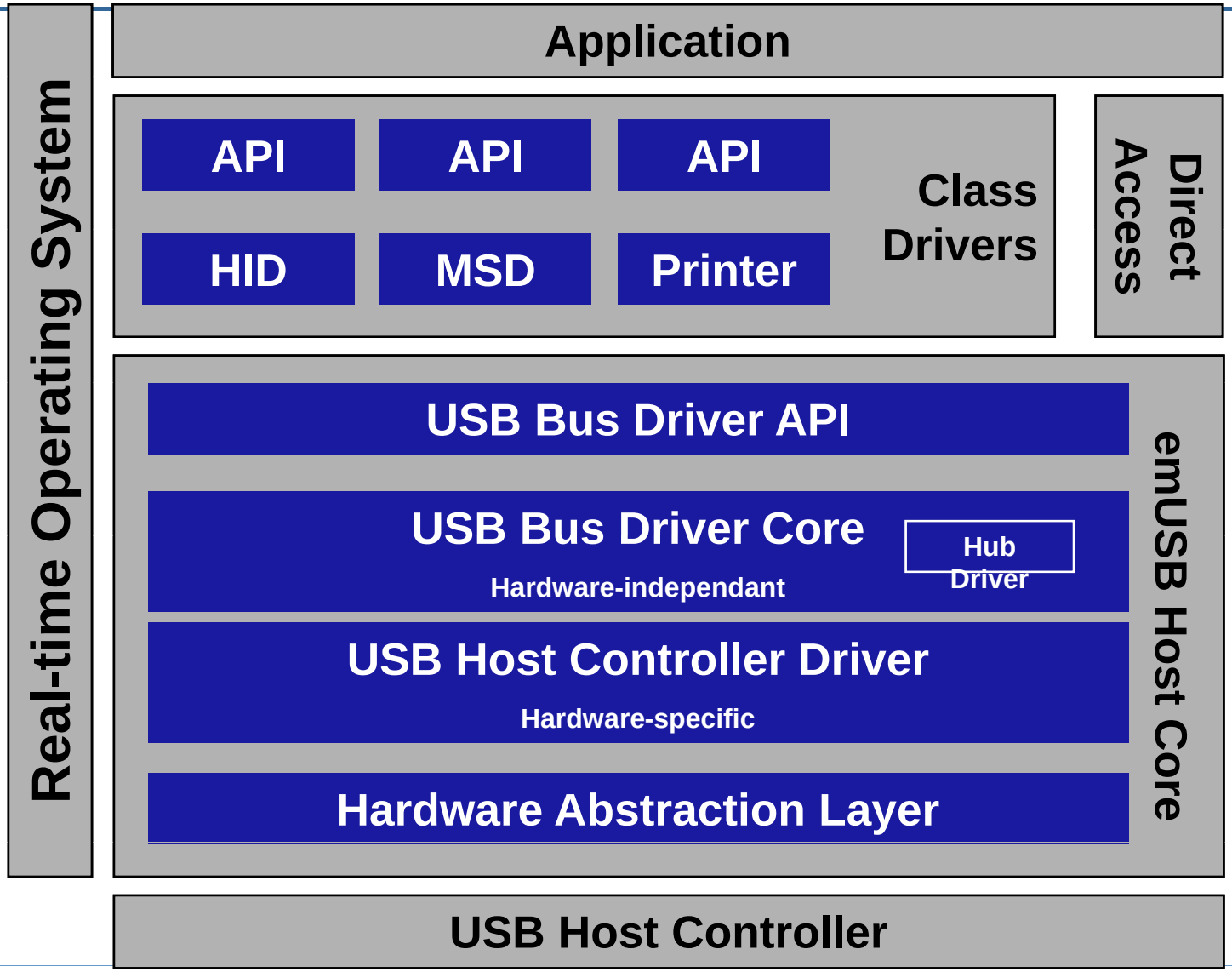
- Control, bulk and interrupt transfers
- Currently supported classes:
  MSD, HID, printer
- USB-2.0-High-speed
- Runs out of the box
- Easy to use
- Small memory footprint
- extended error recovery
  during device enumeration
- Hot plug and hot removal detection
- dynamical attachment/removal of a
  USB Host Controller Driver



emUSB host stack

Middleware
emWin
embOS
embOS/IP
emFile
emUSB

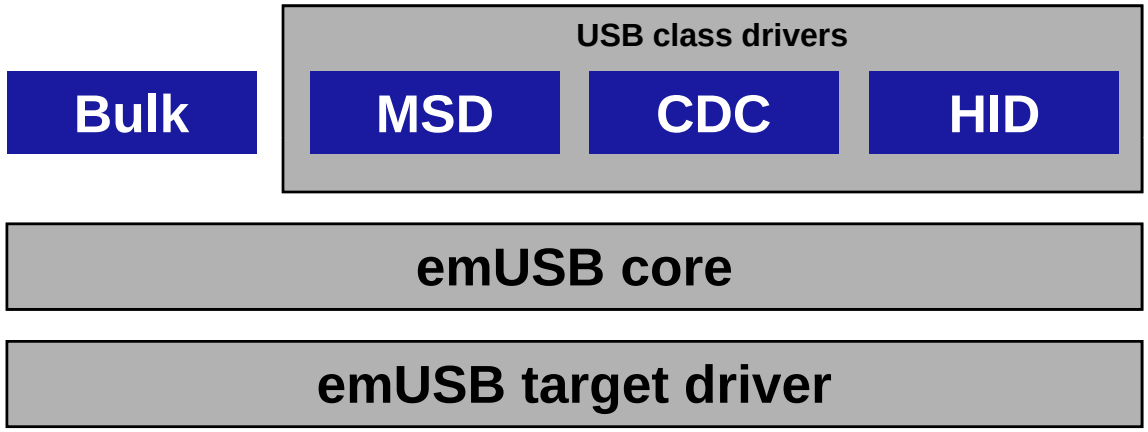
Licensing



emUSB device stack

- Bulk communication component with Windows kernel mode driver available
- MSD Component available
- MSD-CDROM support
- MSD supports any type of storage incl. NAND
- CDC component available
- HID component available

- Supports USB 1.1/2.0 devices
- ISO/ANSI C source code
- Full/High speed support
- Target drivers for most popular USB devices
- Start/test applications supplied
- No royalties



Middleware

MICRIUM

Micriµm

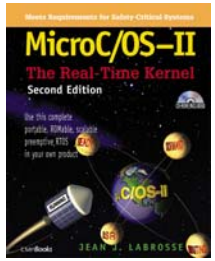
Product Portfolio

- **µC/OS-II**
  - Embedded RTOS
  - µC/OS-MMU
  - µC/OS-MPU
- **µC/OS-III**
  - Embedded RTOS
  - Unlimited objects
  - Preemptive and Round-Robin Scheduling
- **µC/TCP-IP**
  - Embedded TCP/IP v4 stack
  - DHCPc, DNSc, FTP, HTTPs, POP3c, SMTPc, SNTPc,
  - TELNETs, Shell
- **µC/USB-Host**
  - HID, CDC, MSD, Audio, Printer Classes
  - OHCI, EHCI, UHCI
- **µC/USB-Device**
  - Bulk-device stack
  - HID, MSD, CDROM Classes
- **µC/USB-OTG**
- **µC/Bluetooth**
  - SPP, RFCOMM, L2CAP and HCI layers
- **µC/CAN**
  - CAN Framework
  - µC/CANopen I (Small Sensor Slave)
- **µC/Modbus**
  - Master and/or Slave, RS-232C or RS-485
  - ASCII and/or RTU
- **µC/GUI**
  - Embedded Graphical User Interface
- **µC/FS**
  - Embedded File System
  - FAT32, Long Filenames
  - Many media drivers
- **µC/BuildingBlocks**
  - Software time-of-day clock (µC/CLK)
  - Character-based LCD (µC/LCD)
  - Shell (µC/Shell)
  - CRC Calculation (µC/CRC)
  -
- **µC/Probe**
  - Run-time data monitor
  - Any CPU (8-, 16-, 32-, 64-bit or DSP)
  - Any compiler/linker (ELF or IEEE695)
  - Any interface (RS-232C, TCP/IP, USB, etc.)
  - With or without an RTOS

- **Preemptive Multitasking**
- **Supports up to 255 tasks and 255 priorities**
- **Written in ANSI C**
- **Highly portable**
  - Ported to over 45 different CPU architectures
- **Scalable and ROMable**
  - 5K to 20K bytes code, 1K to 3K bytes data
- **Provides standard 'services' to the application**
  - Semaphores, Mutexes, Queues, Task, Time, etc.
- **High Performance**

© 2009, Micriµm, All Rights Reserved

- **Used in 1000s of products all over the world**
- **Book describing internals**



- **Adopted by 100s of Colleges & Universities**
- **Supports most of the MISRA C rules**
- **Third Party Certified**
  - FAA - DO178B Level A
  - FDA - 510(k)
  - IEC - 61508

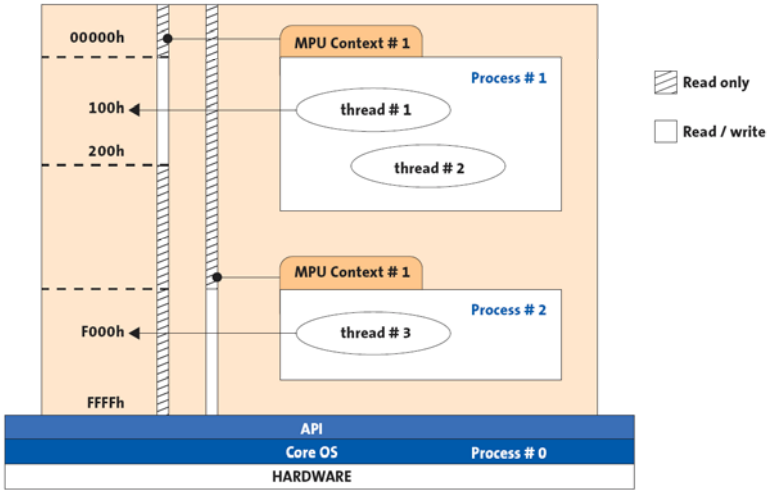
DO-178B Software Certification Levels

Level A	Software that could cause or contribute to the failure of the system resulting in a catastrophic failure condition.
Level B	Software that could cause or contribute to the failure of the system resulting in a hazardous or severe failure condition.
Level C	Software that could cause or contribute to the failure of the system resulting in a major failure condition.
Level D	Software that could cause or contribute to the failure of the system resulting in a minor failure condition.
Level E	Software that could cause or contribute to the failure of the system resulting in no effect on the system.

© 2009, Micriµm, All Rights Reserved

µC/OS-MPU
Memory Protection Unit Extension

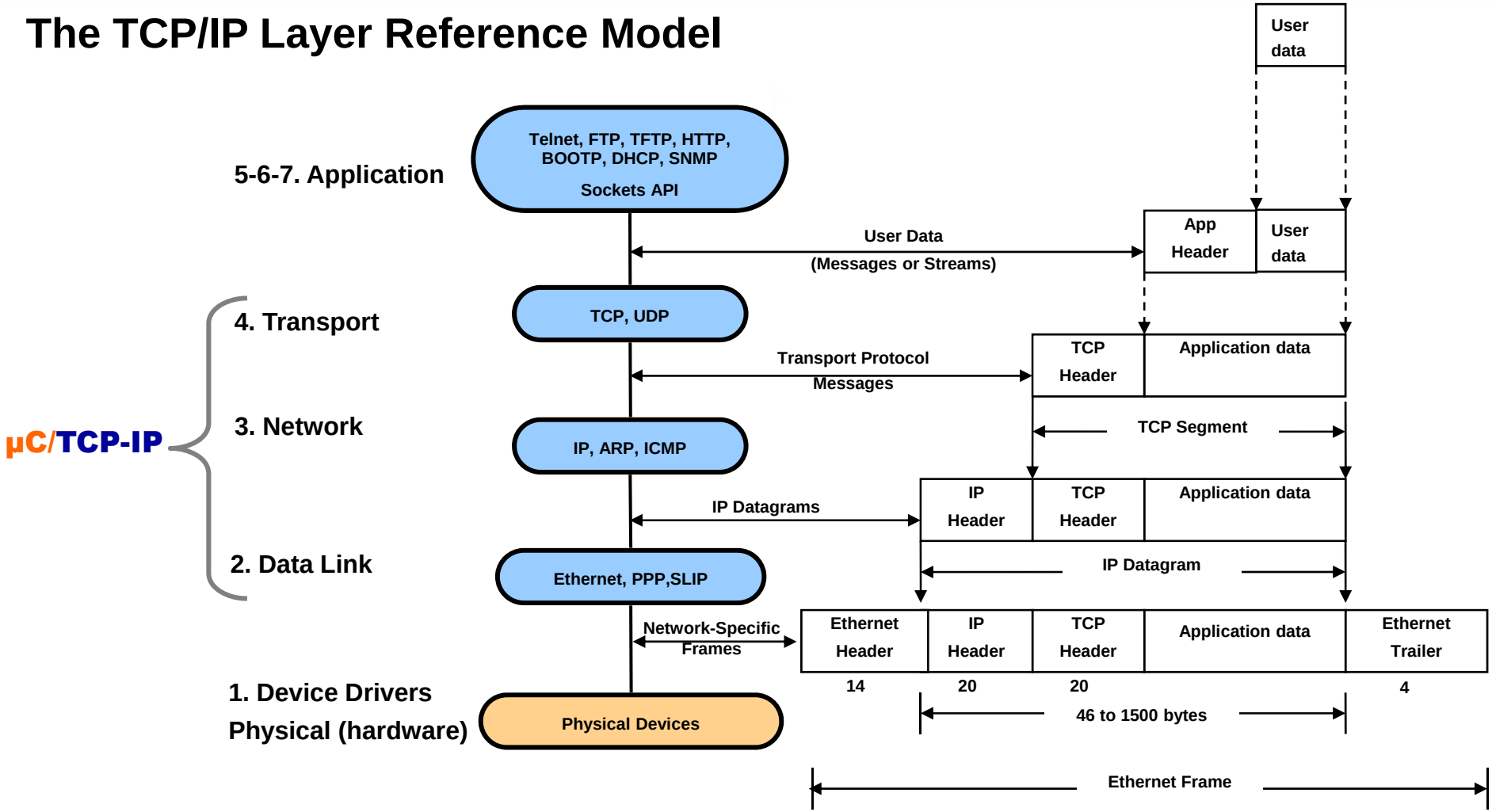
- **Task memory space protection**
  - Protect on task from corrupting another task's data
- **Existing µC/OS-II applications can be adapted easily**
- **Used for Safety Critical Systems**
  - DO178B, 510(k) and IEC-61508 certifications in progress



© 2009, Micriµm, All Rights Reserved

TCP/IP stack model

The TCP/IP Layer Reference Model



© 2009, Micriµm, All Rights Reserved

- **Clean room design**
  - Based on RFCs
  - ANSI C
  - Use RTOS Services for performance guarantee

- **Application layer components:**
  - DHCP (client), DNS (client),
  - FTP (client and server), TFTP (client and server)
  - HTTP (web server), SMTP (client), POP3 (client),
  - SNTP (client), Telnet (server), Shell, Clock
  - More to come

© 2009, Micriµm, All Rights Reserved

- **Easy to Port:**
  - Works with any RTOS (but needs an RTOS)
  - Works with most 32 bit CPUs
  - Works with any NIC
  - Free drivers
- **Small Footprint (Scalable: 75 to 120 Kbytes)**
- **Efficient:**
  - Critical sections kept to a minimum
  - Zero copy
  - Small and Large packet buffers
    - Efficient use of memory
    - Small and Large buffer size adjustable at compile time

© 2009, Micriµm, All Rights Reserved

END

Thank you