SILICON LABS

Debugging Live Cortex-M Based Embedded Systems

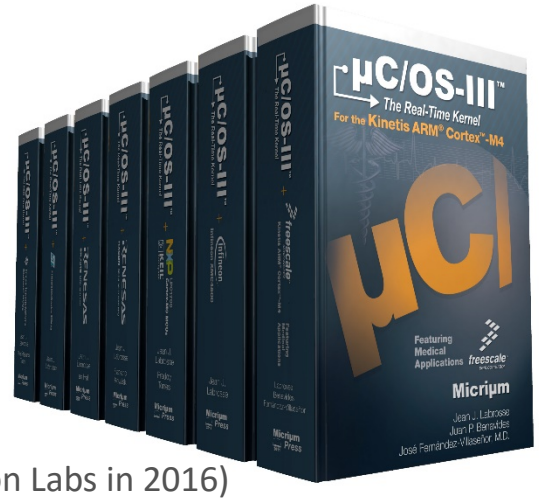
FEBRUARY 2018

JEAN J. LABROSSE

Introduction

Author

µC/OS series of software and books
Numerous articles and blogs



Lecturer

Conferences
Training

Entrepreneur

Micrium founder (acquired by Silicon Labs in 2016)

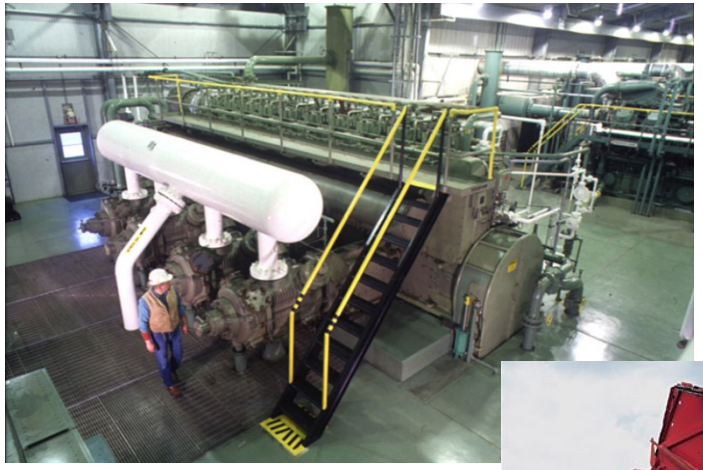
Embedded Systems Innovator

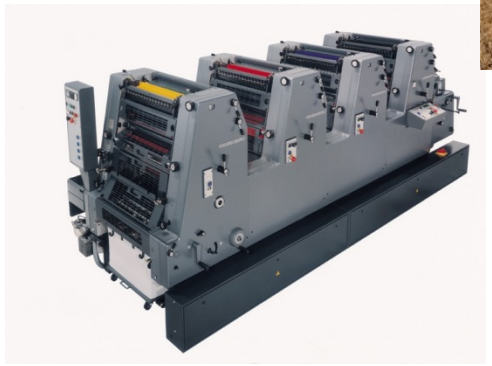
Embedded Computer Design Innovator of the Year award (2015)

Jean.Labrosse@SiLabs.com

www.SiLabs.com/EW2018

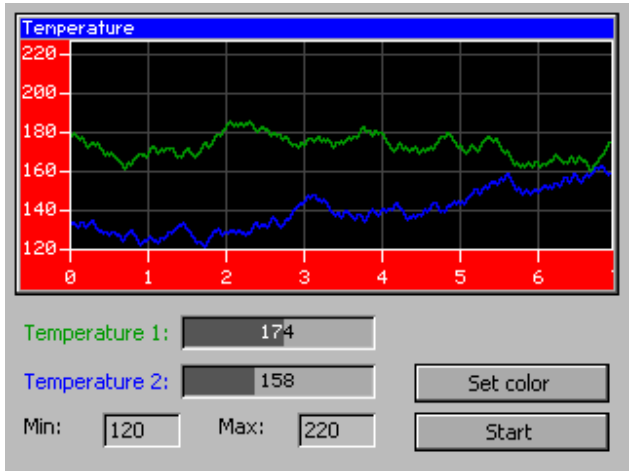
Debugging Live Systems





- You can’t always ‘single step’ through code!
  - Engine control
  - Printing presses
  - Food processing
  - Flight management
  - Chemical reactions
  - Agricultural equipment
  - Etc.
- Stopping these systems can have disastrous and/or costly consequences
  - Must be tested and debugged live

How Do You ‘See’ inside These Systems?

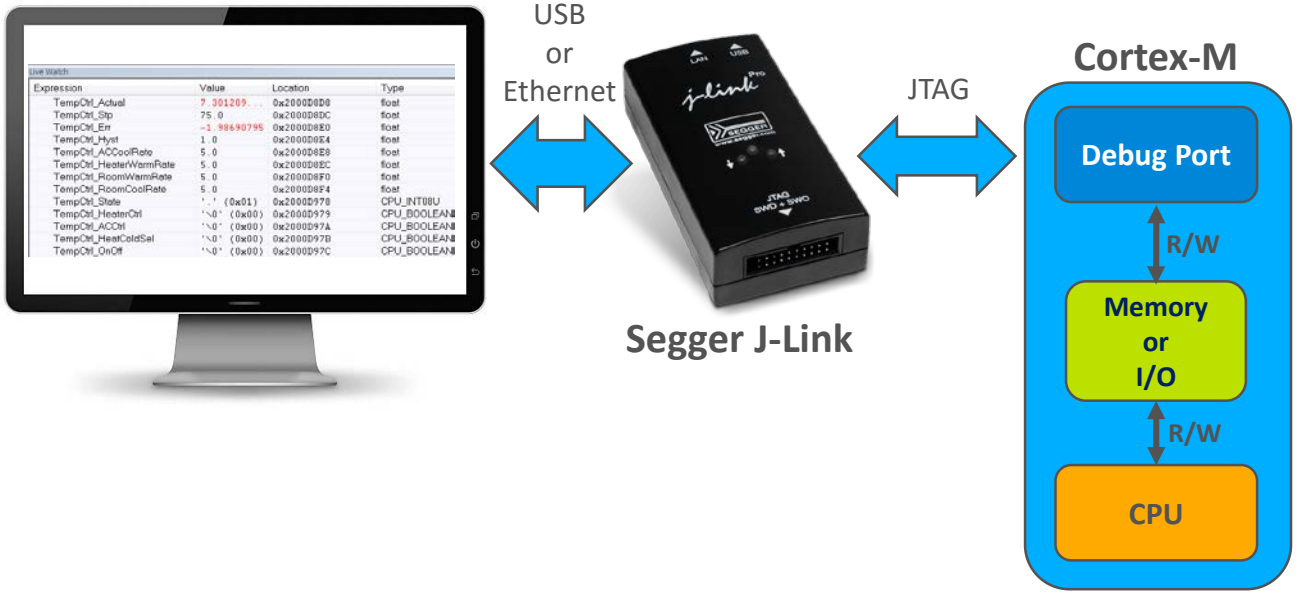


- Displaying values using:
  - LED annunciators
  - 7-Segment numeric displays
  - Bar graphs
  - Alphanumeric displays
  - Graphical user interfaces (GUIs)
  - `printf()` statements to a terminal
  - Debugger’s live watch … limited to numerical values
  - Etc.
- Drawbacks:
  - Display capabilities might be limited
  - All require target resident code
    - Heisenberg effect is often significant
  - Limited to what you can see/change
  - If you forget something …
    - Rebuild code
    - Download
    - Try to get back to the same test conditions

What if We Move the Display/Controls to a PC?

- Using COTS man-machine interfaces (MMIs)
  - e.g. Wonderware 'InTouch' (Schneider)
  - Much better at visualizing the process
  - Can monitor and/or change hundreds of values
  - Data logging capabilities
- Uses standard PLC protocols
  - e.g. Modbus, ProfiNet, DeviceNet, etc.
- Drawbacks:
  - Target needs a database of accessible variables
  - Requires target resident code
  - Adds overhead, complexity and cost
- COTS MMIs are typically for end use
  - Could be useful during development

The ARM Cortex-M Debug Port



- Core debugging:
  - Halting
  - Single stepping
  - Resume
  - Reset
  - Register accesses
- Up to 8 hardware breakpoints
- Up to 4 hardware watchpoints
- Optional ***instruction*** trace
- ***Data*** trace
- Instrumentation trace (printf() like) – 32 channels
- Profiling counters
- PC sampling
- **On-the-fly memory and I/O accesses**
  - Can be a security risk for deployed systems

Live Watch

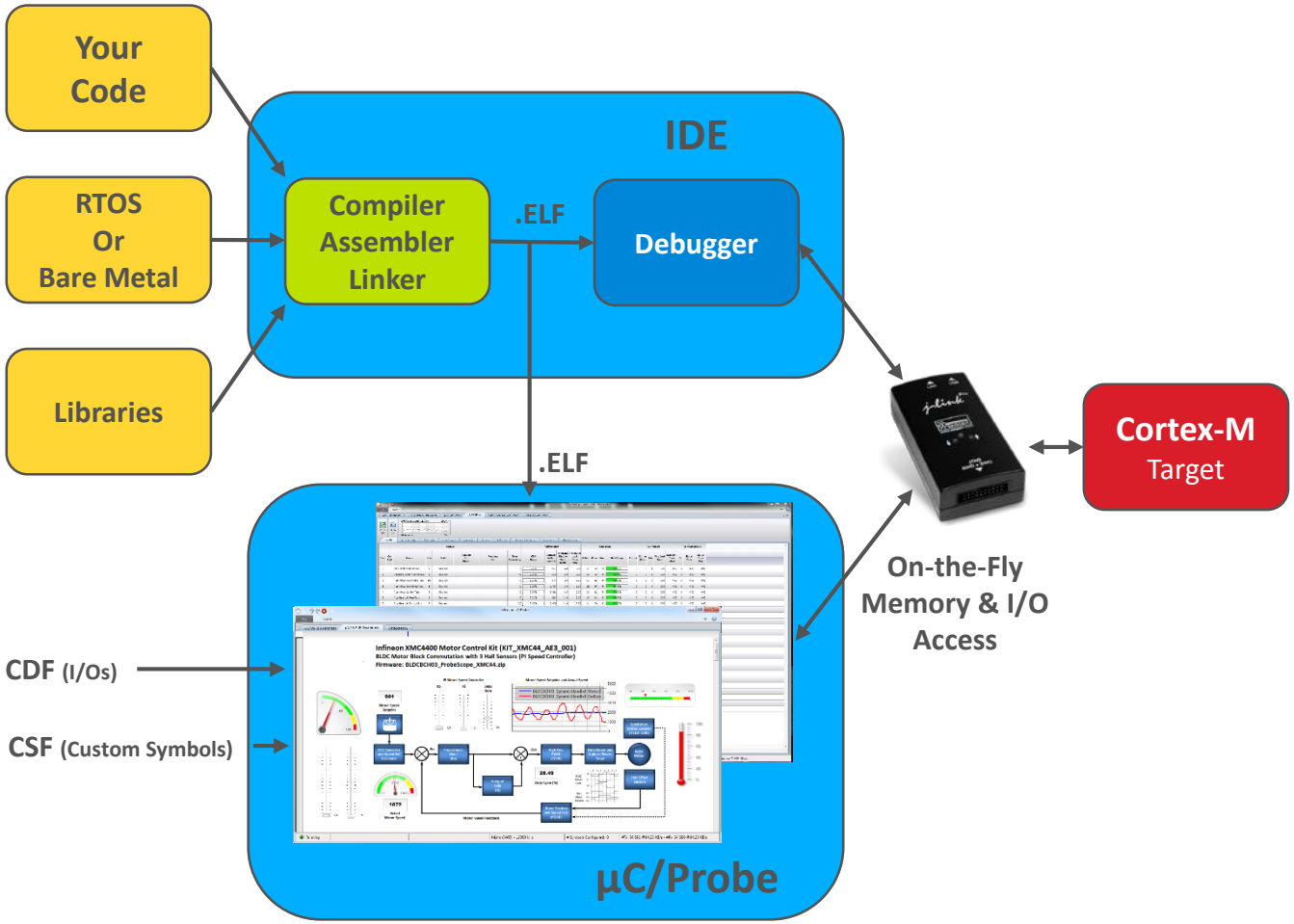
```
static  void  AppTempCtrl (void)
{
    AppTempErr      = AppTempActual - AppTempStp;
    AppTempHeatRate = ((CPU_FP32)AppTempHeaterWatts / (CPU_FP32)1000.0)
                    * ((CPU_FP32)1.0 / (CPU_FP32)AppTempRoomSize);
    AppTempCoolRate = ((CPU_FP32)1.0 / (CPU_FP32)AppTempRoomSize);
    if (AppTempActual > (AppTempStp + AppTempHyst)) {     /* Determine what state we are in           */
        AppTempState   = 3;                               /* Above Stp + Hyst                         */
    } else if (AppTempActual < (AppTempStp - AppTempHyst)) {
        AppTempState   = 1;                               /* Below Stp - Hyst                         */
    } else {
        AppTempState   = 2;                               /* Between Stp + Hyst and Stp - Hyst        */
    }

    if (AppTempCtrlEn == DEF_ENABLED) {                   /* See if controller is turned on           */
        BSP_LED_Toggle(2);
                                                          /* ------------ HEATING MODE --------------- */
        if (AppTempSelHeat == DEF_ON) {                   /* See if heater is selected                */
            AppTempAC_Ctrl     = DEF_OFF;
            switch (AppTempState) {
                case 1:
                     AppTempHeater_Ctrl = DEF_ON;
                     AppTempActual     += AppTempHeatRate;
                     BSP_LED_On(3);
                     BSP_LED_Off(1);
                     break;

                case 2:
                     if (AppTempHeater_Ctrl) {
                         AppTempActual += AppTempHeatRate;
                     } else {
                         AppTempActual -= (CPU_FP32)0.0005;    /* Cool the room at natural rate        */
                     }
                     break;

                case 3:
                     AppTempHeater_Ctrl = DEF_OFF;
                     AppTempActual     -= (CPU_FP32)0.0005;   /* Cool the room at natural rate        */
                     BSP_LED_Off(3);
                     BSP_LED_Off(1);
                     break;
            }
                                                          /* ---------- COOLING MODE ------------- */
        } else {                                          /* We want to get the room colder           */
            AppTempHeater_Ctrl = DEF_OFF;
            switch (AppTempState) {
```

- Debuggers have offered ***Live Watch*** for years
  - Uses the on-the-fly-feature of the Cortex-M
- Typically only displays numerical values
  - Difficult to see trends and orders of magnitudes
  - Choice of Decimal, Hex, Float, etc.
- Update rate is typically 1 Hz

Live Watch

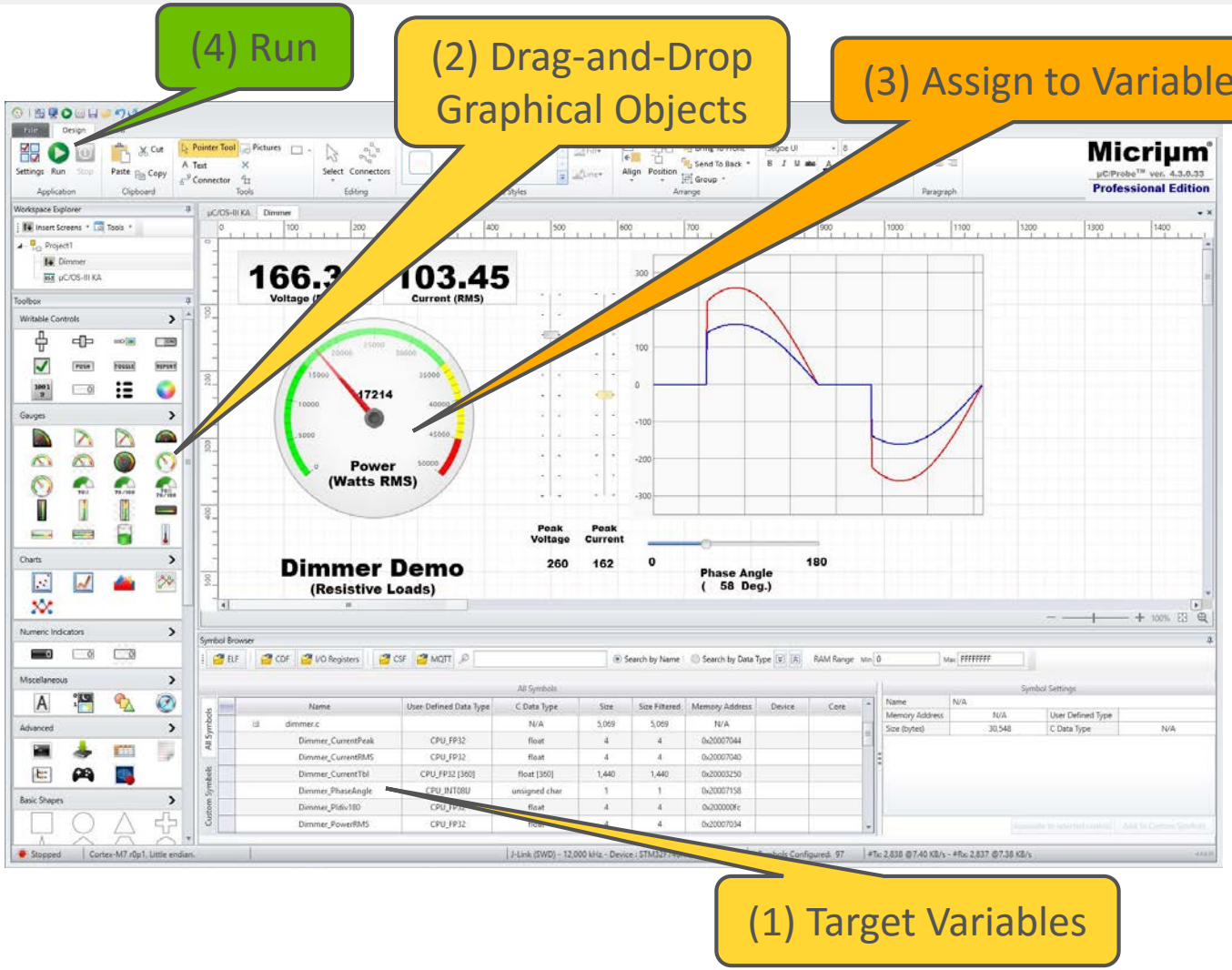
Expression	Value	Location	Type
TempCtrl_Actual	7.301209...	0x2000D8D8	float
TempCtrl_Stp	75.0	0x2000D8DC	float
TempCtrl_Err	-1.98690795	0x2000D8E0	float
TempCtrl_Hyst	1.0	0x2000D8E4	float
TempCtrl_ACCoolRate	5.0	0x2000D8E8	float
TempCtrl_HeaterWarmRate	5.0	0x2000D8EC	float
TempCtrl_RoomWarmRate	5.0	0x2000D8F0	float
TempCtrl_RoomCoolRate	5.0	0x2000D8F4	float
TempCtrl_State	'.' (0x01)	0x2000D978	CPU_INT08U
TempCtrl_HeaterCtrl	'\0' (0x00)	0x2000D979	CPU_BOOLEAN
TempCtrl_ACCtrl	'\0' (0x00)	0x2000D97A	CPU_BOOLEAN
TempCtrl_HeatColdSel	'\0' (0x00)	0x2000D97B	CPU_BOOLEAN
TempCtrl_OnOff	'\0' (0x00)	0x2000D97C	CPU_BOOLEAN

µC/Probe, Graphical Live Watch®



- An MMI for embedded systems
  - Use the **.ELF** as the database (same as downloaded code)
  - Like a doctor's stethoscope (non-intrusive)
- Adding ***graphics*** capabilities to ***Live Watch***
  - Display or change values numerically or graphically
- A universal **tool** that interfaces to any target:
  - 8-, 16-, 32-, 64-bit and DSPs
  - No CPU intervention with Cortex-M
  - Requires target resident code if not using the debug port:
    - RS232C, TCP/IP or USB
- For **bare metal** or **RTOS**-based applications
  - Micrium's RTOS and TCP/IP awareness

Seeing inside Your Embedded Target

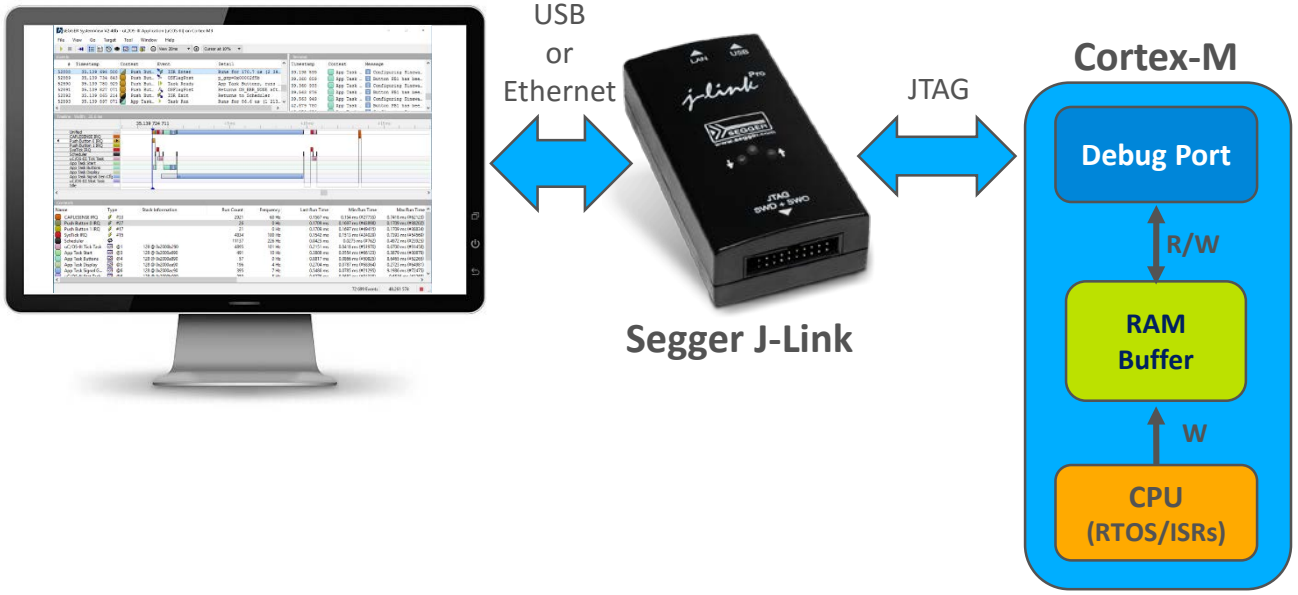


- (1) Load the **.ELF** from the build
  - You have access to ***all global variables*** by their name
- (2) Drag-and-drop graphical objects from the palette
- (3) Assign variables (by name) to:
  - Gauges, meters, bar graphs, cylinders, etc.
  - Numeric indicators, sliders, switches, etc.
  - Built-in oscilloscope (up to 8 channels)
  - Excel spreadsheet interface
  - Scripting
  - Terminal window
- (4) Run – starts collecting the current value of the selected variables.
  - Don't have to stop the target!

Advanced features

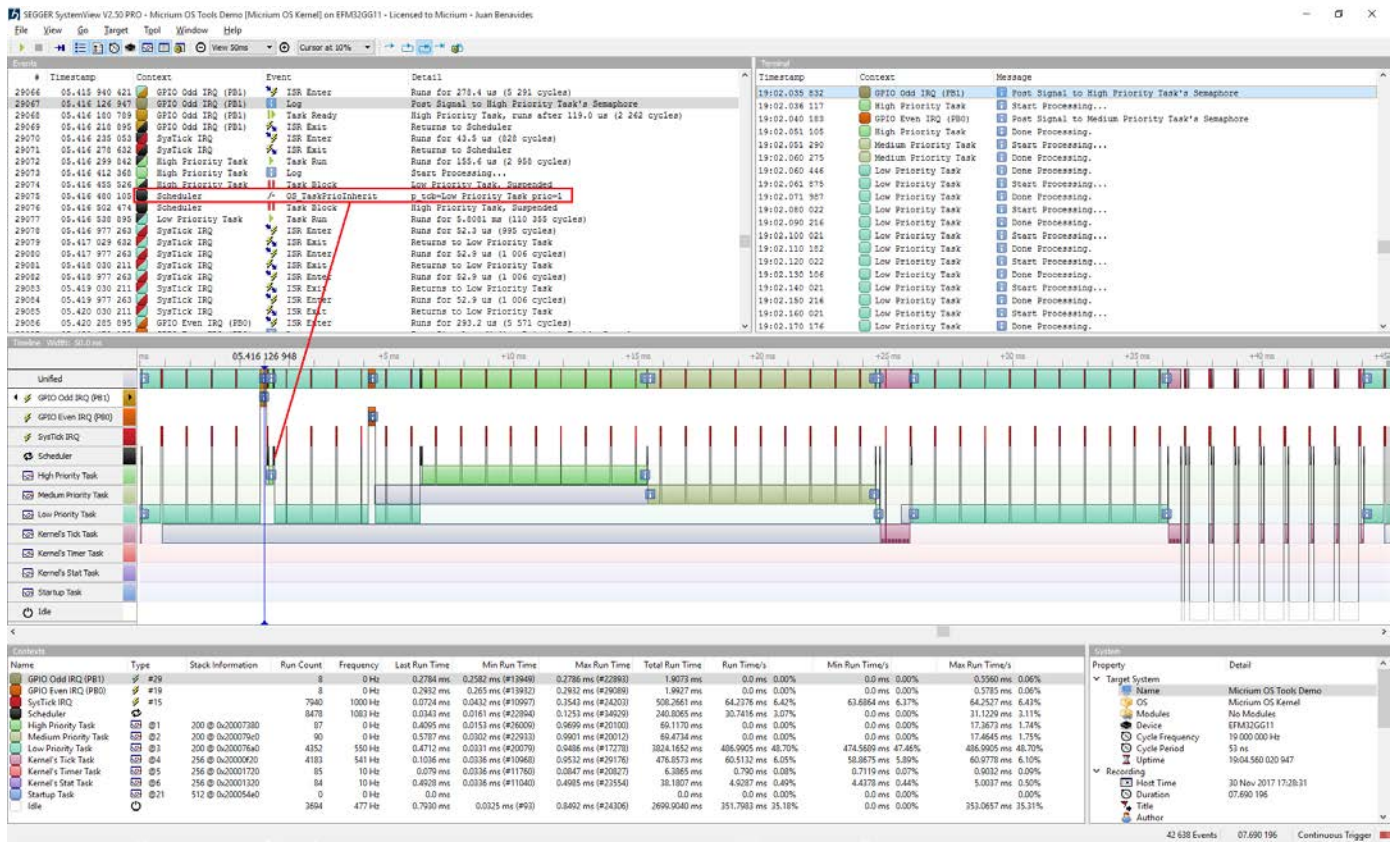
- 8-channel oscilloscope
  - No need to instrument your code and bring out signals
- Charts (trends)
- Excel spreadsheet interface
- Scripting
- Terminal window
- RTOS awareness
  - CPU usage of a per-task basis
  - ISR and task stack usage on a per-task basis
  - Status of all kernel objects
- TCP/IP Awareness
  - Buffer usage
  - Interface status (Ethernet or Wi-Fi)
  - Data transfer rates
  - ARP cache status
  - Etc.
- More

Segger's SystemView



- Typically used in an RTOS-based system
  - The RTOS needs to be 'instrumented'
  - Supports µC/OS-III, µC/OS-5, embOS and FreeRTOS
- Events are 'recorded' into a RAM buffer
  - ISR enter/exit
  - Semaphore pend/post
  - Mutex pend/post
  - Message queue pend/post
  - Etc.
- The RAM buffer is read when the RTOS is idling

Segger's SystemView



- Displays the execution profile of RTOS-based systems
  - Displayed live
  - Visualizing the execution profile of an application
  - Trigger on any task or ISR
  - Helps confirm the expected behavior of your system
- Measures CPU usage on a per-task basis
  - Min/Max/Avg task run time
  - Counts the number of task executions
- Display the occurrence of 'events' in your code
- Traces can be saved for post-analysis or record keeping

- www.Segger.com

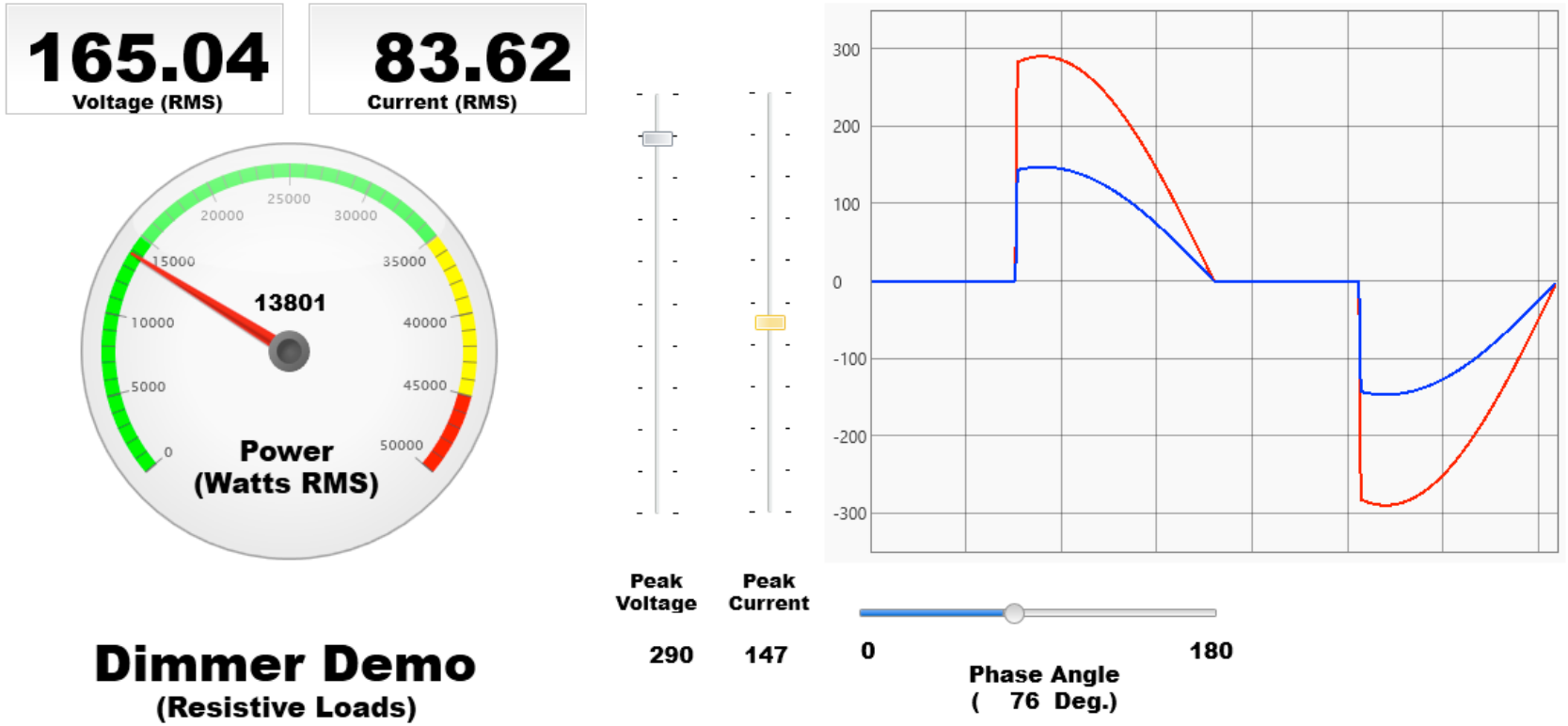
Percepio's Tracealyzer

- Records RTOS events on a timeline
  - ISRs
  - Tasks
- Traces are saved for post-analysis or record keeping
- More than 25 different views to look at data
  - Views are correlated
- Works with:
  - Segger's J-Link
  - TCP/IP
  - USB
- Works with many RTOSs:
  - µC/OS-III and µC/OS-5
  - FreeRTOS
  - RTX5
  - Etc.

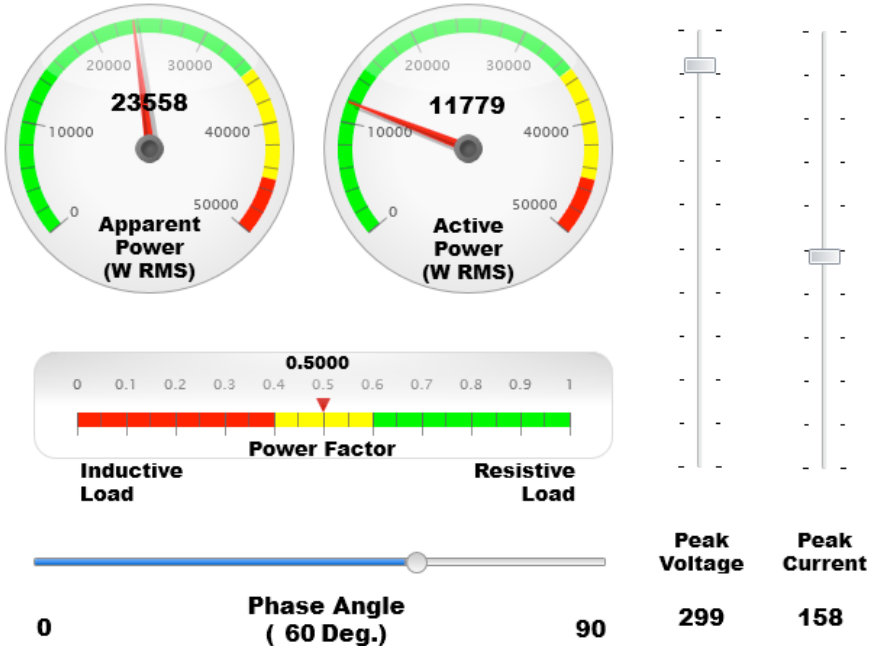
- www.Percepio.com

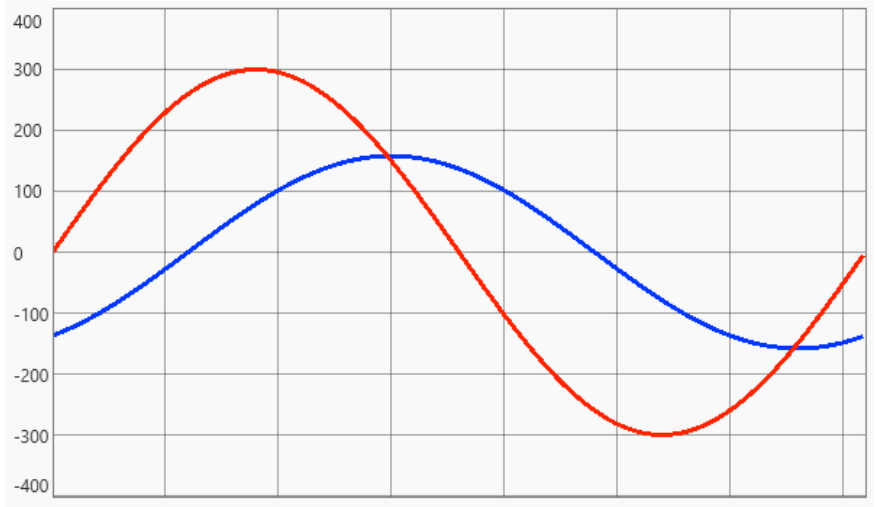
Using µC/Probe to Simulate Algorithms

Simulating a Dimmer

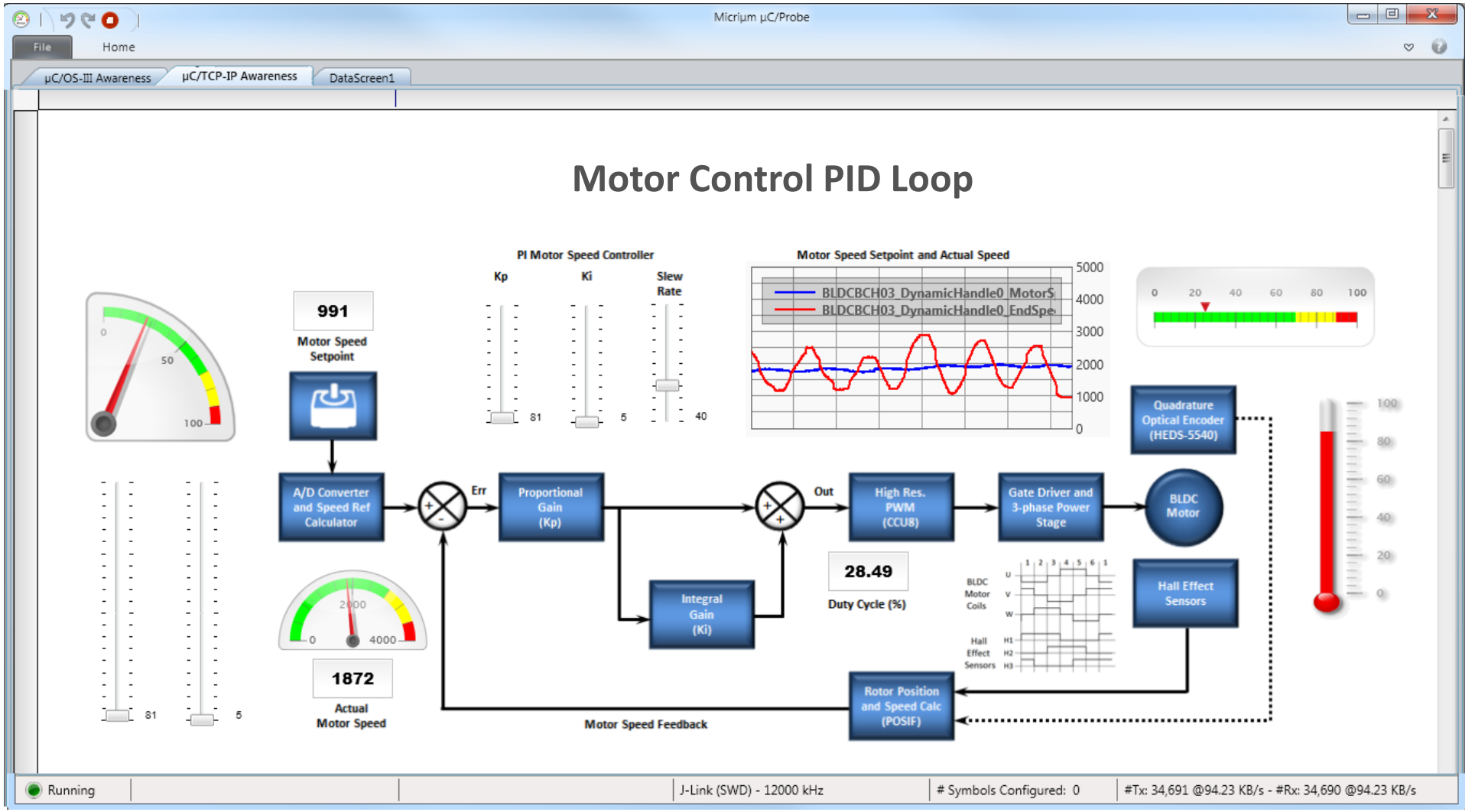


Simulating a Power Meter

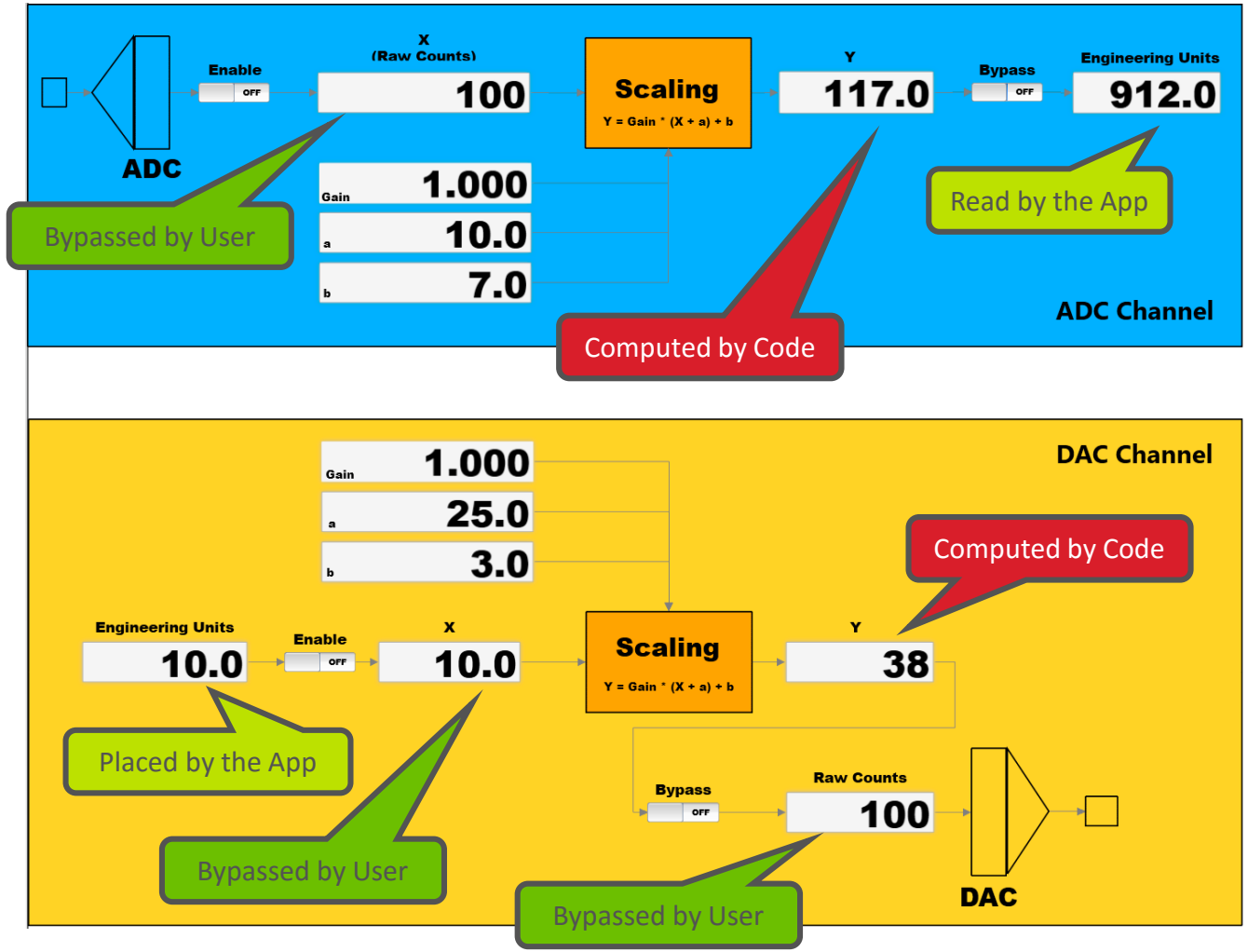




Motor Control Using a PID Loop



Verifying an ADC and DAC Driver




```
void  AppNP_AI_Update (void)
{
    if (AppNP_AI.En > 0u) {
        // Read analog input from actual ADC into AppNP_X
    }
    AppNP_AI.Y = (CPU_FP32)(AppNP_AI.Gain * (AppNP_AI.X + AppNP_AI.a) + AppNP_AI.b);
    if (AppNP_AI.Bypass > 0u) {
        AppNP_AI.EU = AppNP_AI.Y;
    }
}
```

```
void  AppNP_AO_Update (void)
{
    if (AppNP_AO.En > 0u) {
        AppNP_AO.X = AppNP_AO.EU;
    }
    AppNP_AO.Y = (CPU_INT32U)(AppNP_AO.Gain * (AppNP_AO.X + AppNP_AO.a) + AppNP_AO.b);
    if (AppNP_AO.Bypass > 0u) {
        AppNP_AO.Cnts = AppNP_AO.Y;
    }
    // Output AppNP_AO.Cnts to actual DAC
}
```

Conclusion

- Embedded systems ***can't always*** be single stepped when debugging
- Cortex-M's debug port allows ***on-the-fly*** memory read and writes
  - Great for debugging
  - Can be a security risk for deployed systems
- We need tools to visualize and debug ***live*** systems
  - Micrium's µC/Probe can display and change target values, at run-time without interfering with the CPU
  - Segger's SystemView and Percepio's Tracealyzer provide RTOS event traces of a running system

Thank you!

SILABS.COM