Virtual Analysis of an i.MX6 Multicore SoC Design

Jon McDonald

Solutions Architect

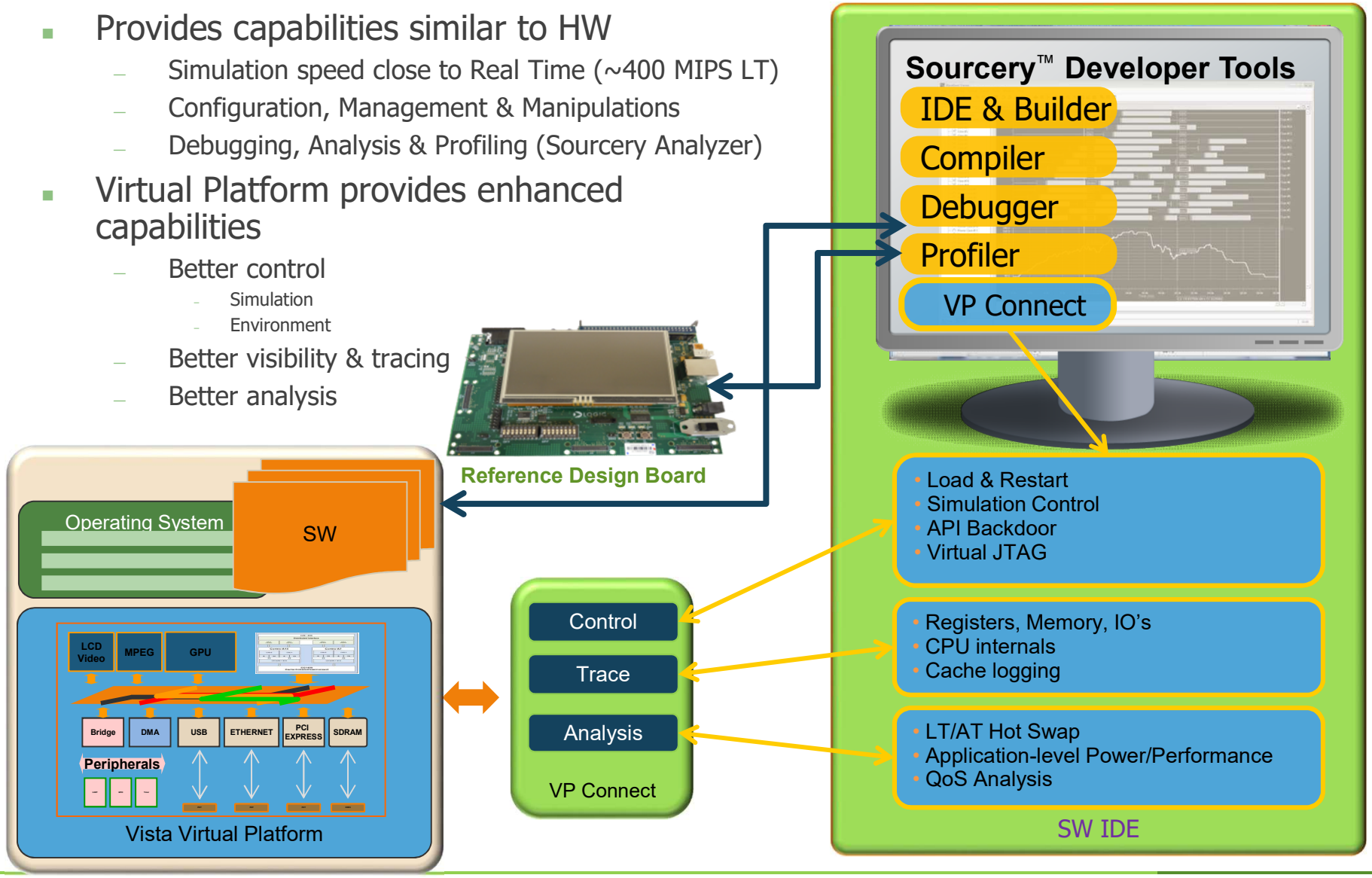
Mentor Consulting Division

mentor embedded

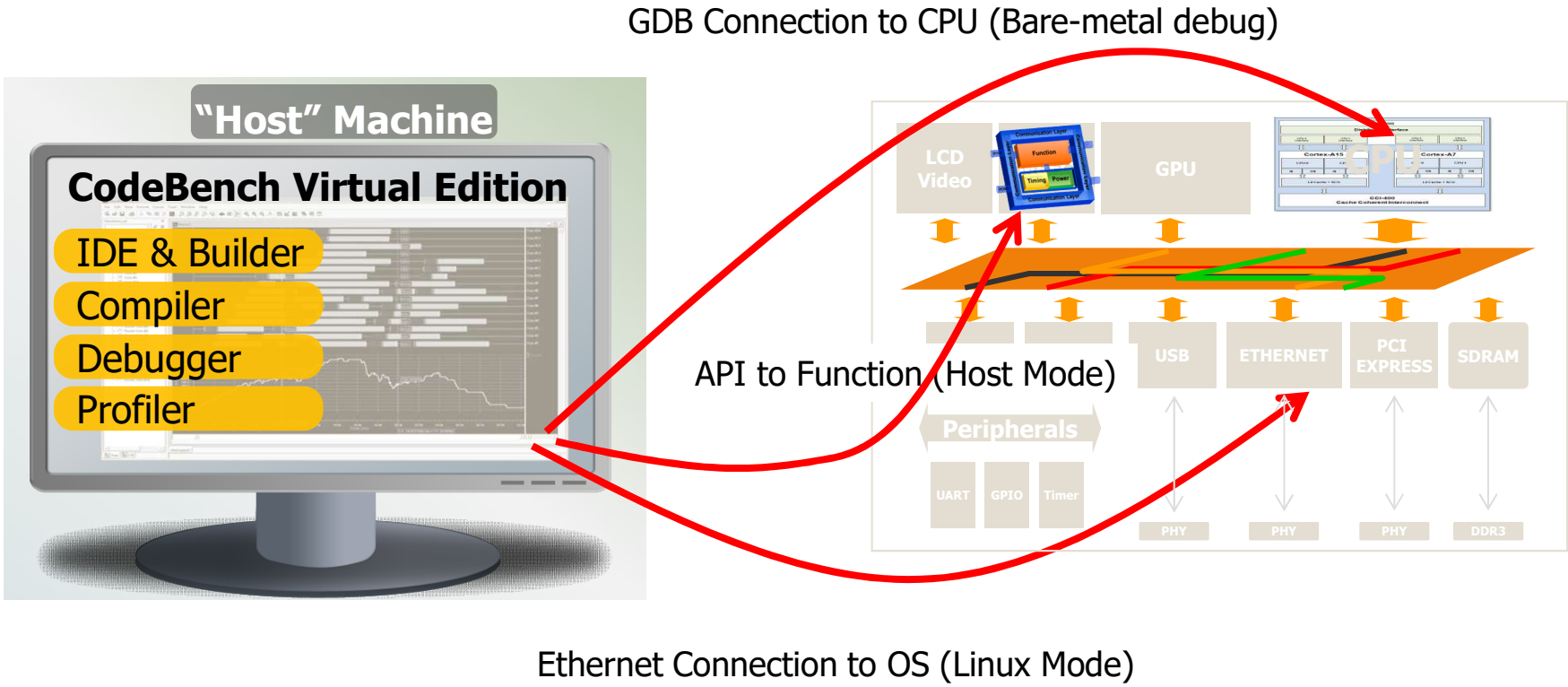
mentor.com/embedded

Virtual Platform Approach

- Provides capabilities similar to HW
  - Simulation speed close to Real Time (~400 MIPS LT)
  - Configuration, Management & Manipulations
  - Debugging, Analysis & Profiling (Sourcery Analyzer)
- Virtual Platform provides enhanced capabilities
  - Better control
    - Simulation
    - Environment
  - Better visibility & tracing
  - Better analysis



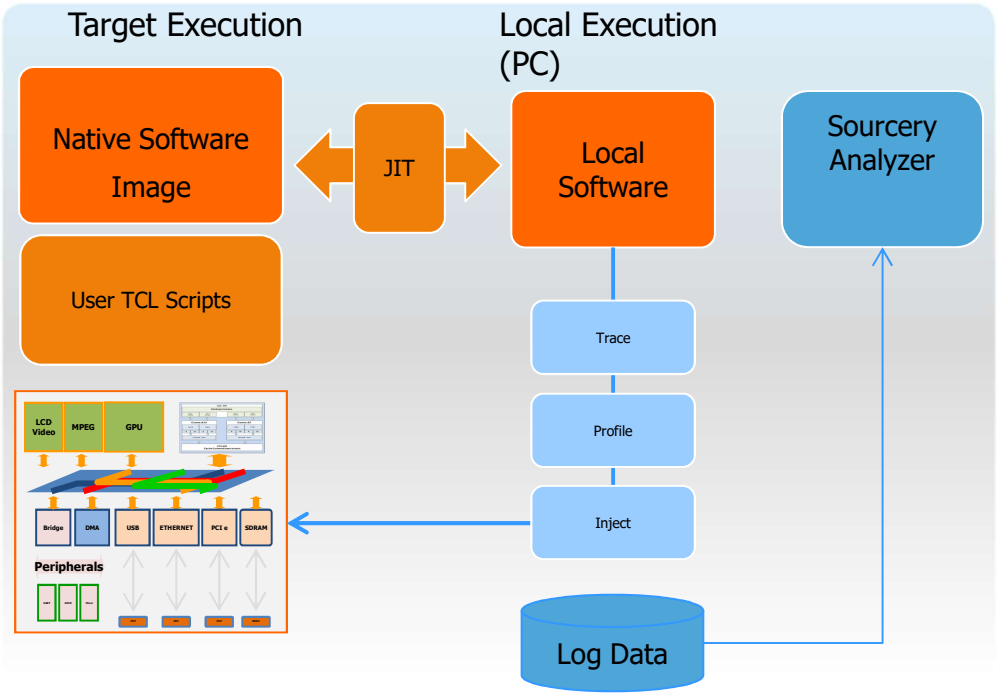
Connection to Virtual Target



The debugger can connect to the virtual target in Linux, Bare-metal or Host mode without any probe.

Non-intrusive technology

- Debug and Profile unmodified software images
  - Code Instrumentation affects behavior and performance
- Debug third party Software when sources are not available
- Manipulate software dynamically at runtime



Non-intrusive Analysis

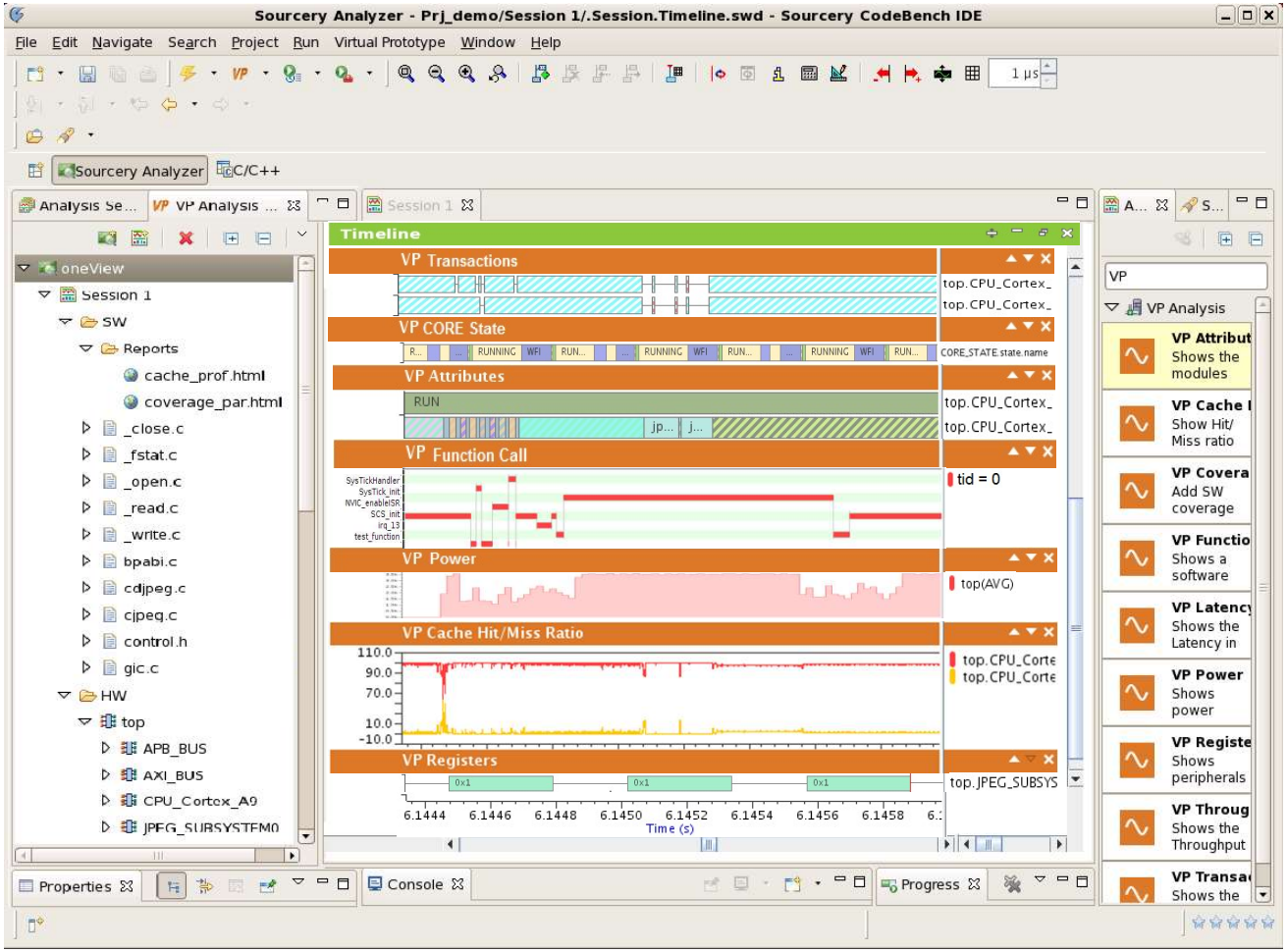
- Concurrent HW & SW Analysis under CodeBench VE – Sourcery Analyzer

Hardware:

- ✓ Transactions
- ✓ Throughput,
- ✓ Latency
- ✓ Power
- ✓ Registers
- ✓ Cache Hit/Miss

Software:

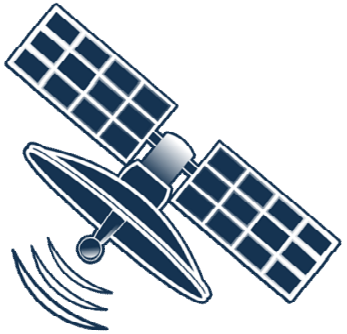
- ✓ Function-calls
- ✓ Profiling
- ✓ Code-Coverage
- ✓ CPU States
- ✓ CPU Current-function
- ✓ OS-Aware:
  - Tasks
  - Interrupts
  - Trap
  - …
- ✓ Custom HW/SW Analysis



What Are We Designing?

- A yacht water depth alarm system with GPS information panel
- Using Freescale i.MX6 technology with I2C peripherals for Sonar, GPS, and the Alarm
- The alarm is raised when the water depth becomes too low
- The water depth and GPS details along with speed are displayed on the LCD panel

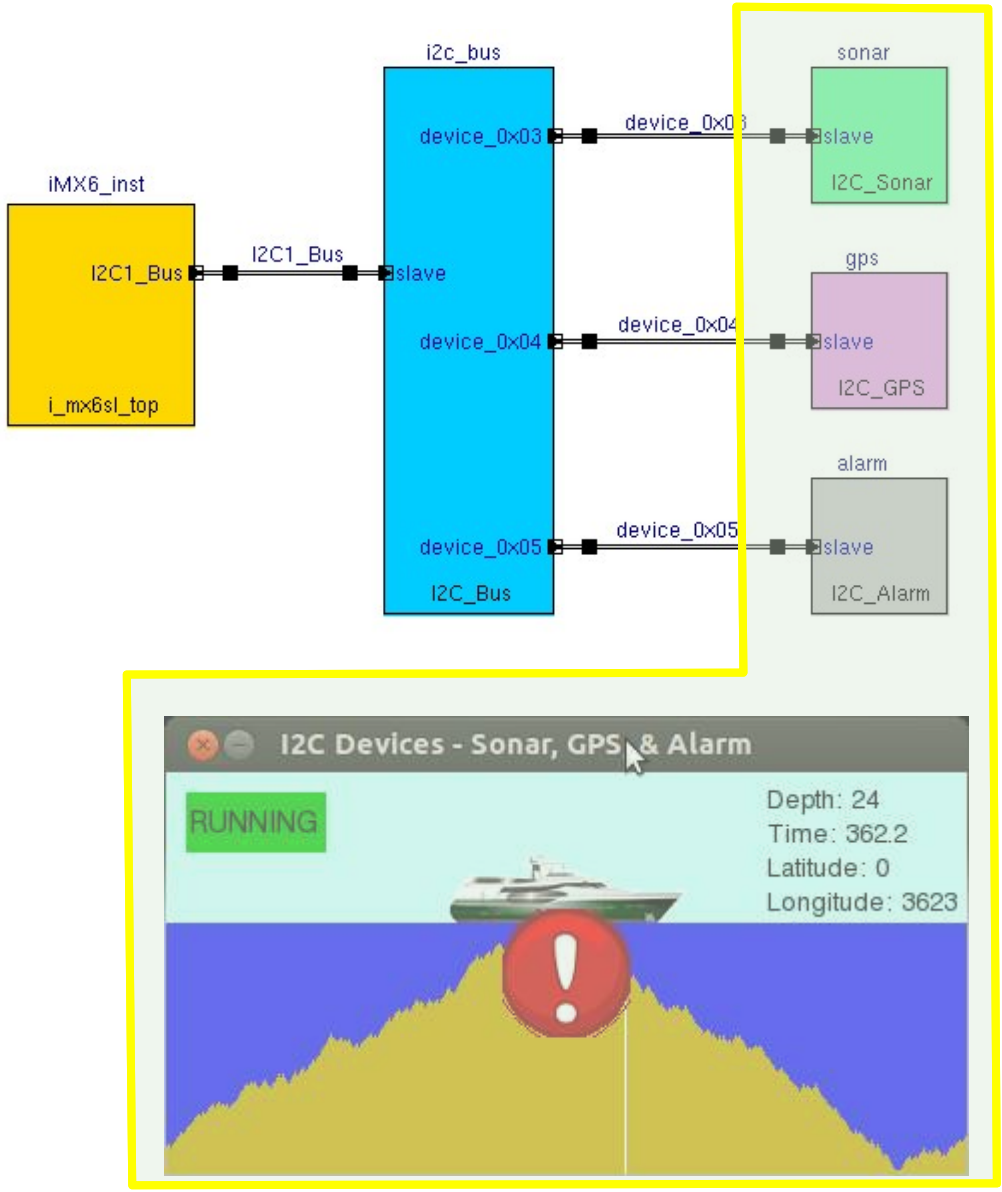
Hardware Components



- Freescale i.MX6
  - LCD
- I2C Devices
  - GPS
  - Sonar
  - Alarm

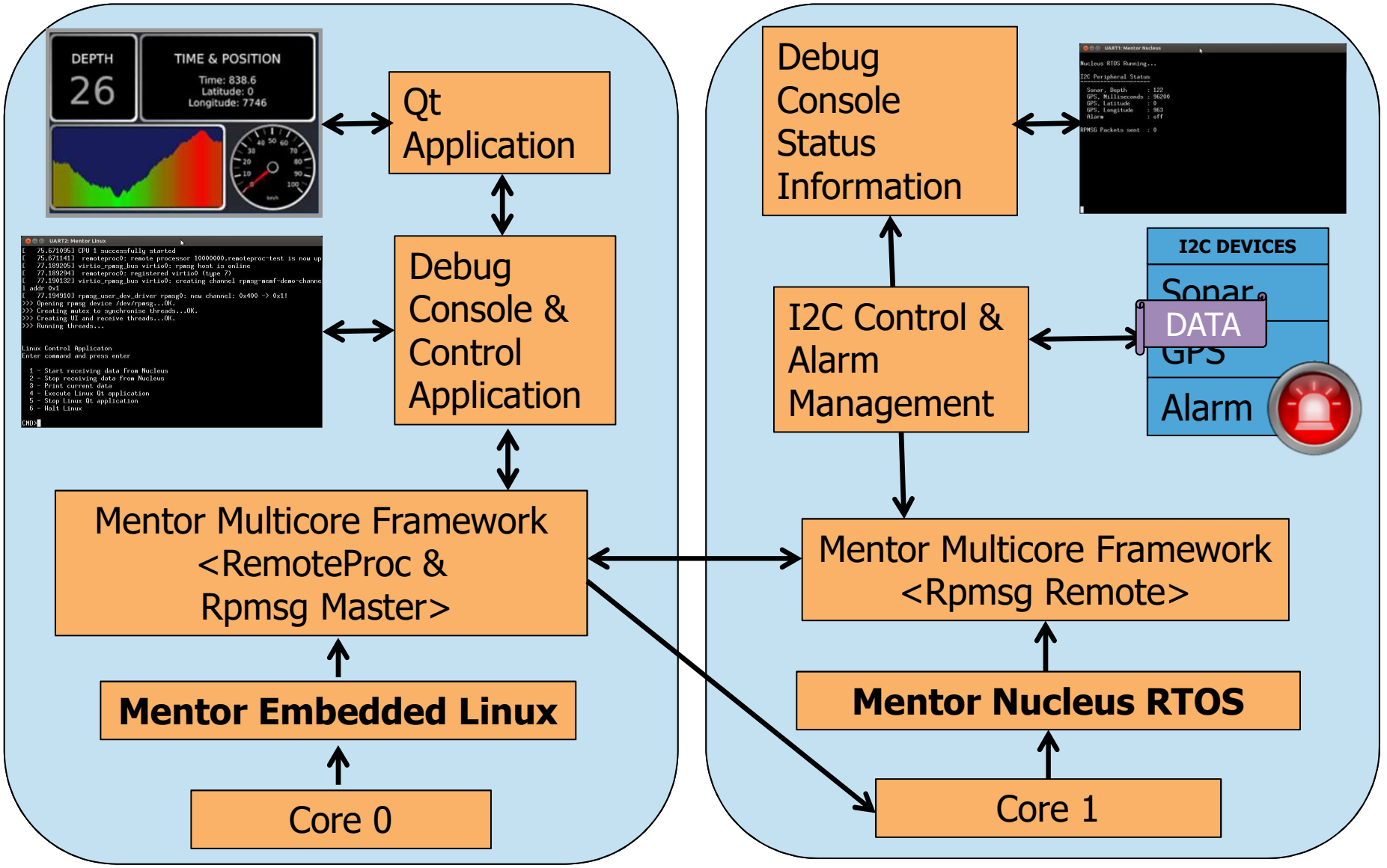
Simulation Infrastructure

- Virtual i.MX6, including:
  - LCD
  - I2C, UARTs, SDCard
- Virtual I2C Devices
  - GPS
  - Sonar
  - Alarm
- Simulation Stimulus
  - Stop/Start yacht
  - Generate random water depth
  - Generate GPS data
  - Display Alarm state



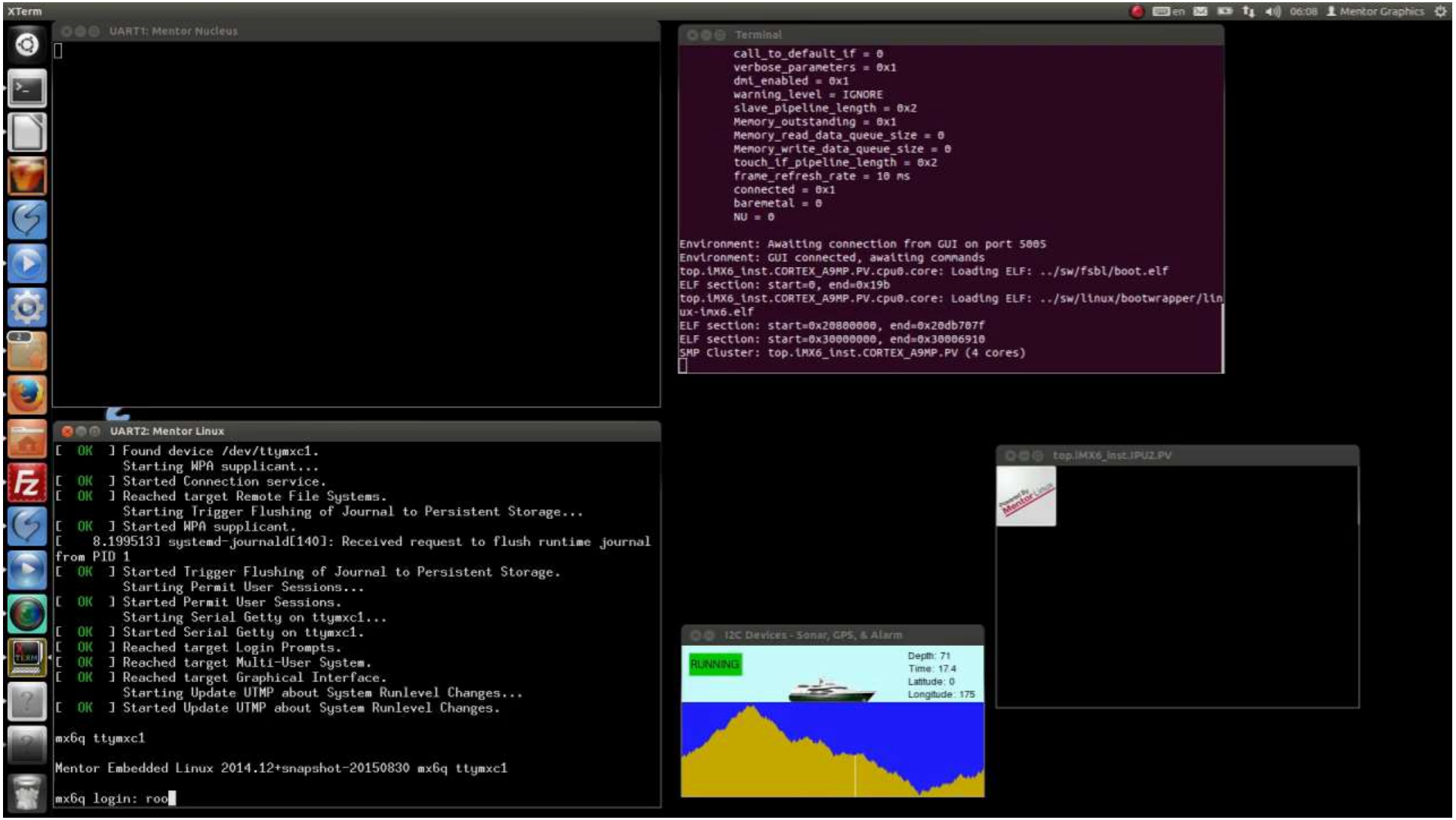
mentor embedded

Embedded Software Components



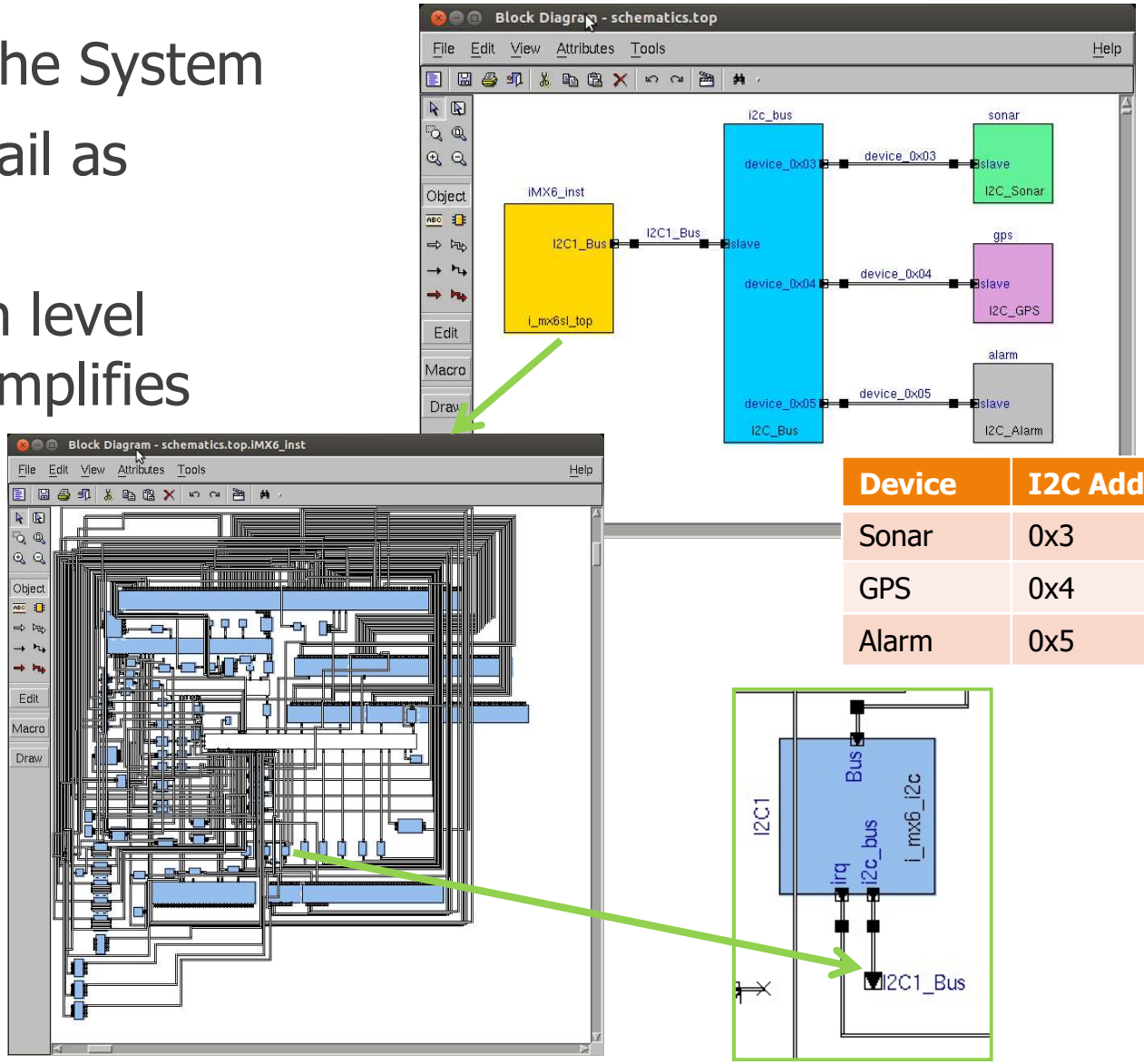
mentor embedded

Running the Virtual Platform



Structure

- Configure the System
- Expose detail as required
- Transaction level interface simplifies modeling



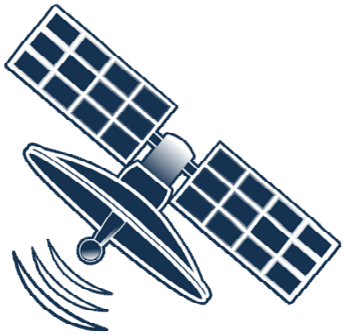
Device	I2C Address
Sonar	0x3
GPS	0x4
Alarm	0x5

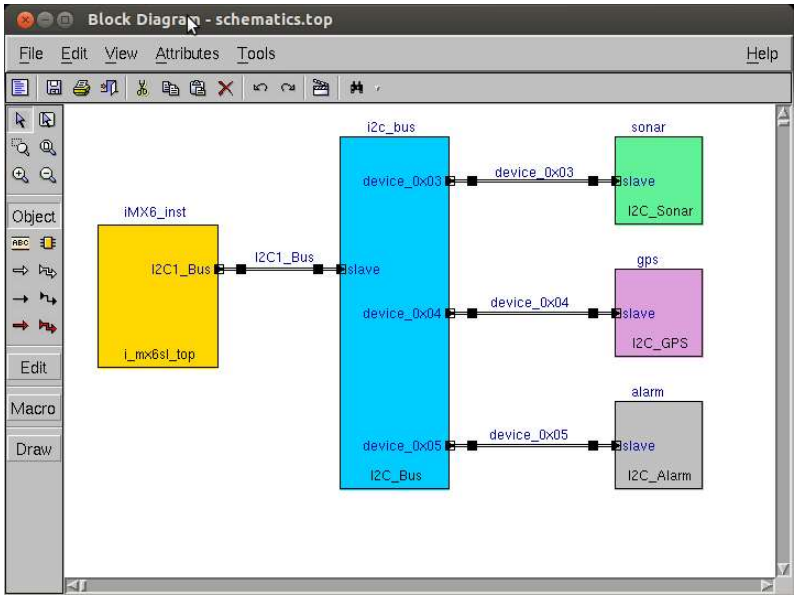
System Startup

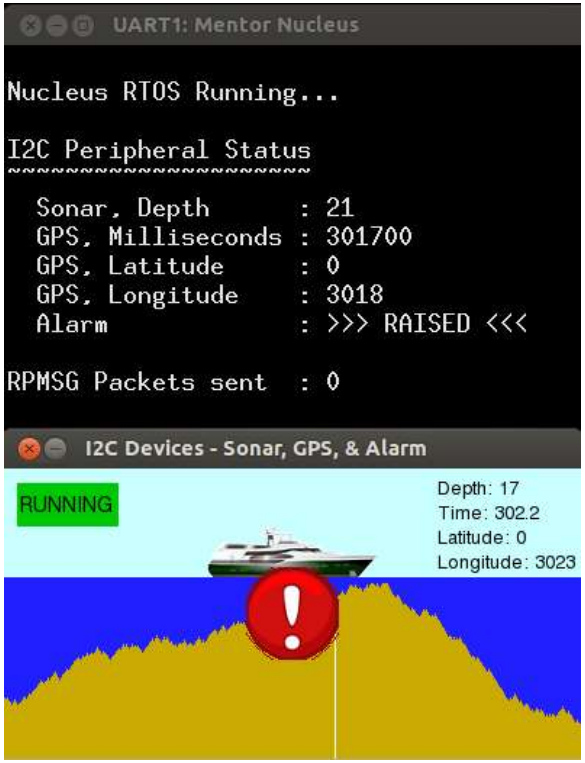
- Image identical to physical device
- Virtual models for environment
- Boot and execution sequence matches physical device usage

Environment Interaction

- Environment models provide realistic input
- UI models provide output
- Enable dynamic user interaction
- Console interface for dev flow

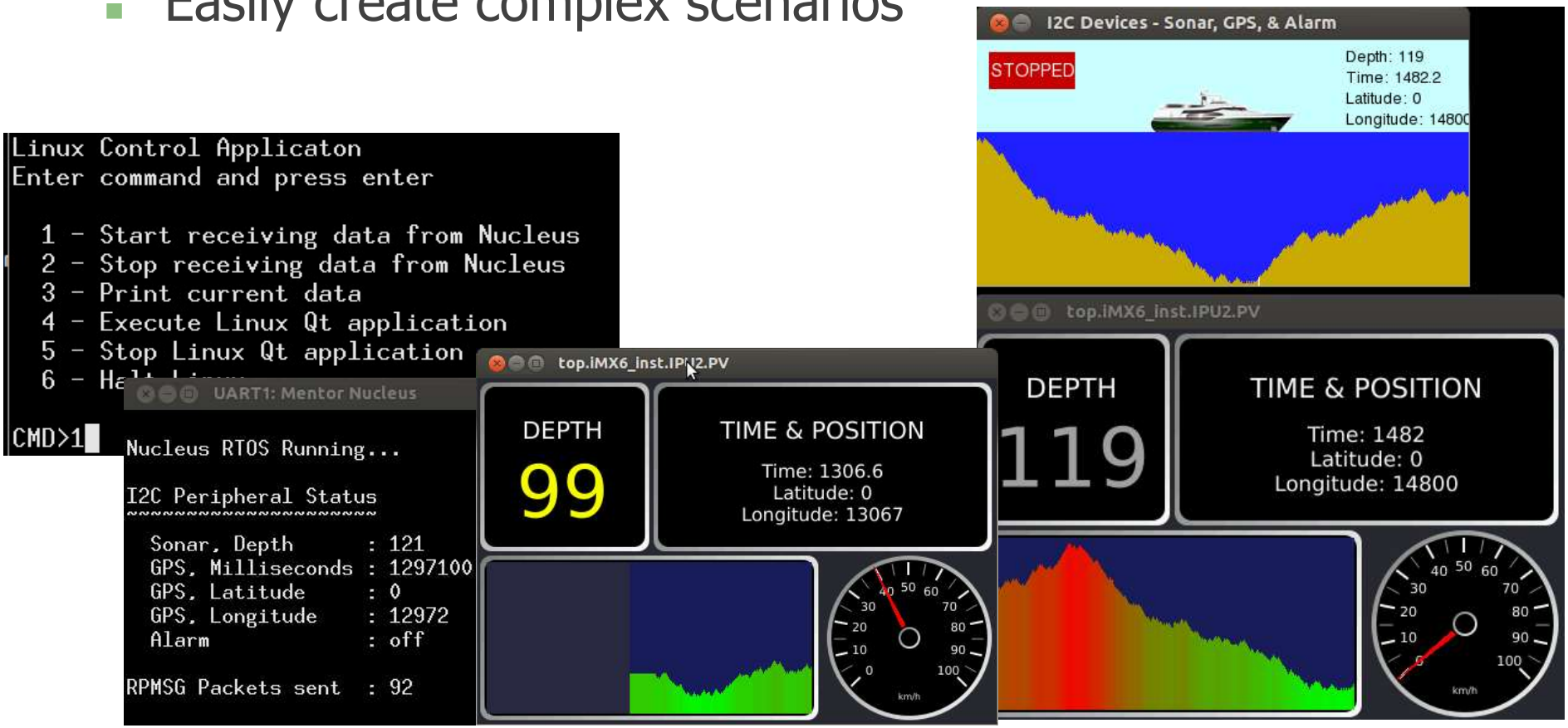




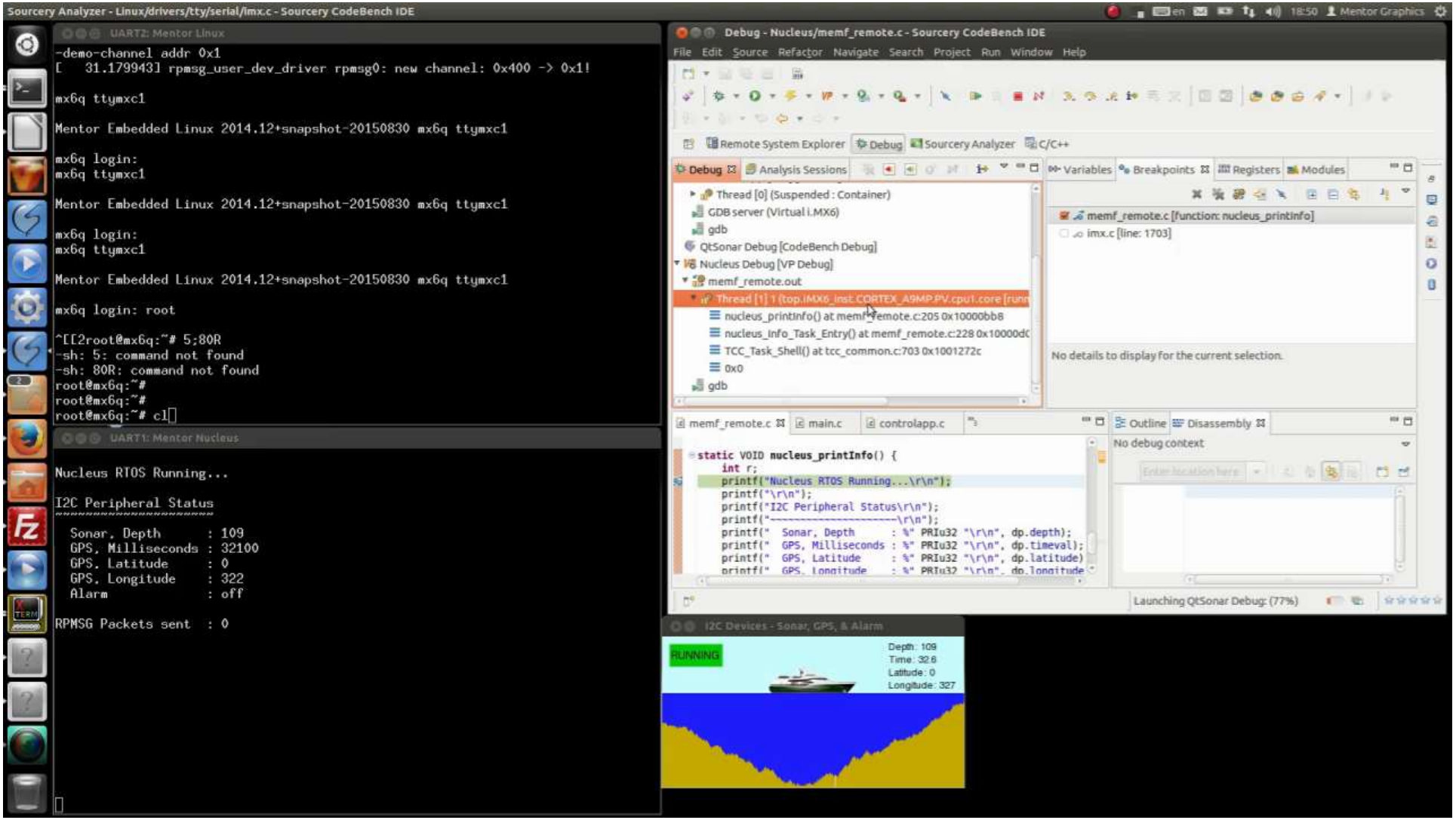


Running

- Command interface identical to physical device
- Data interaction corresponds to real world results
- Easily create complex scenarios

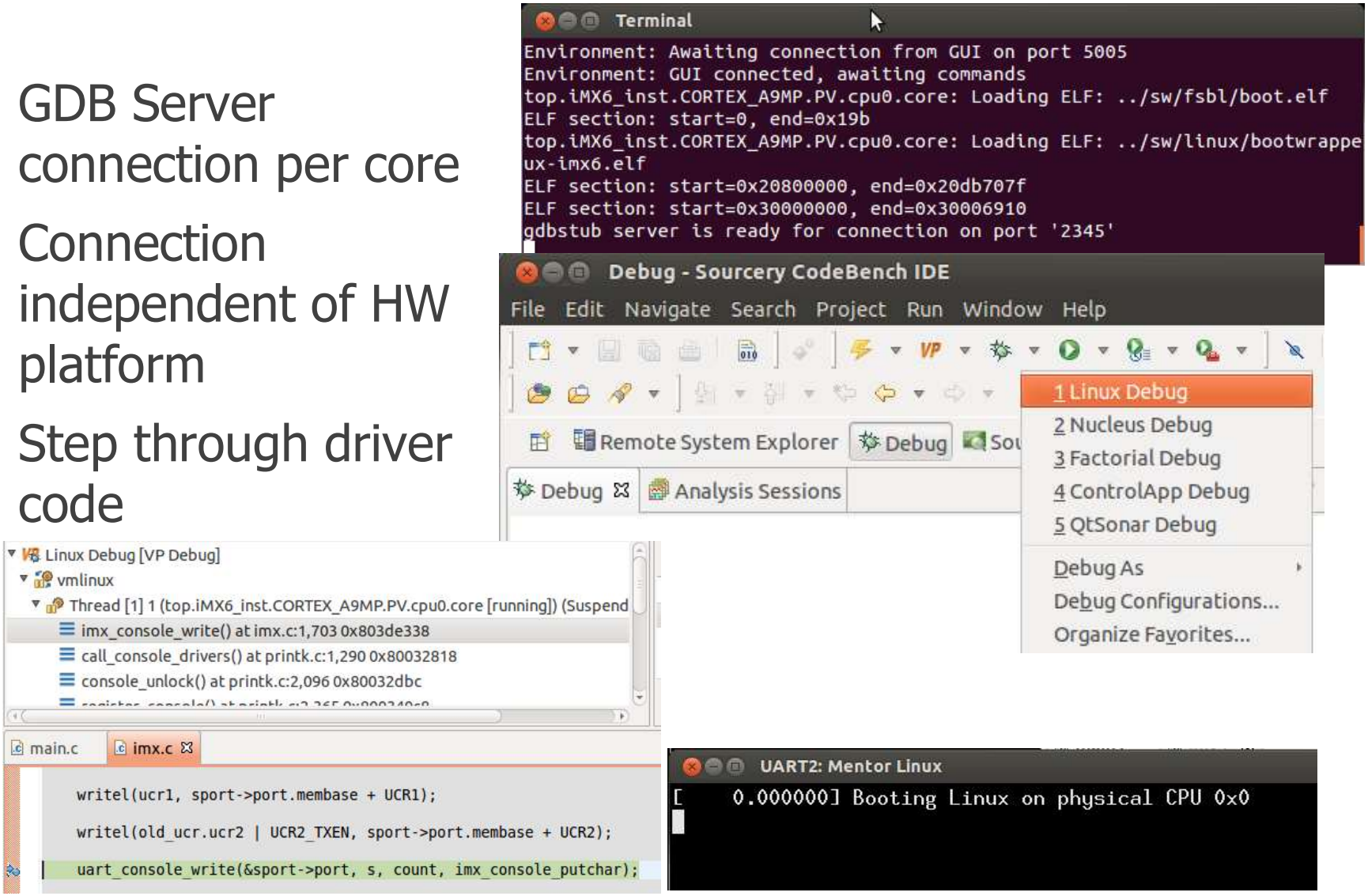


Debugging the Virtual Platform



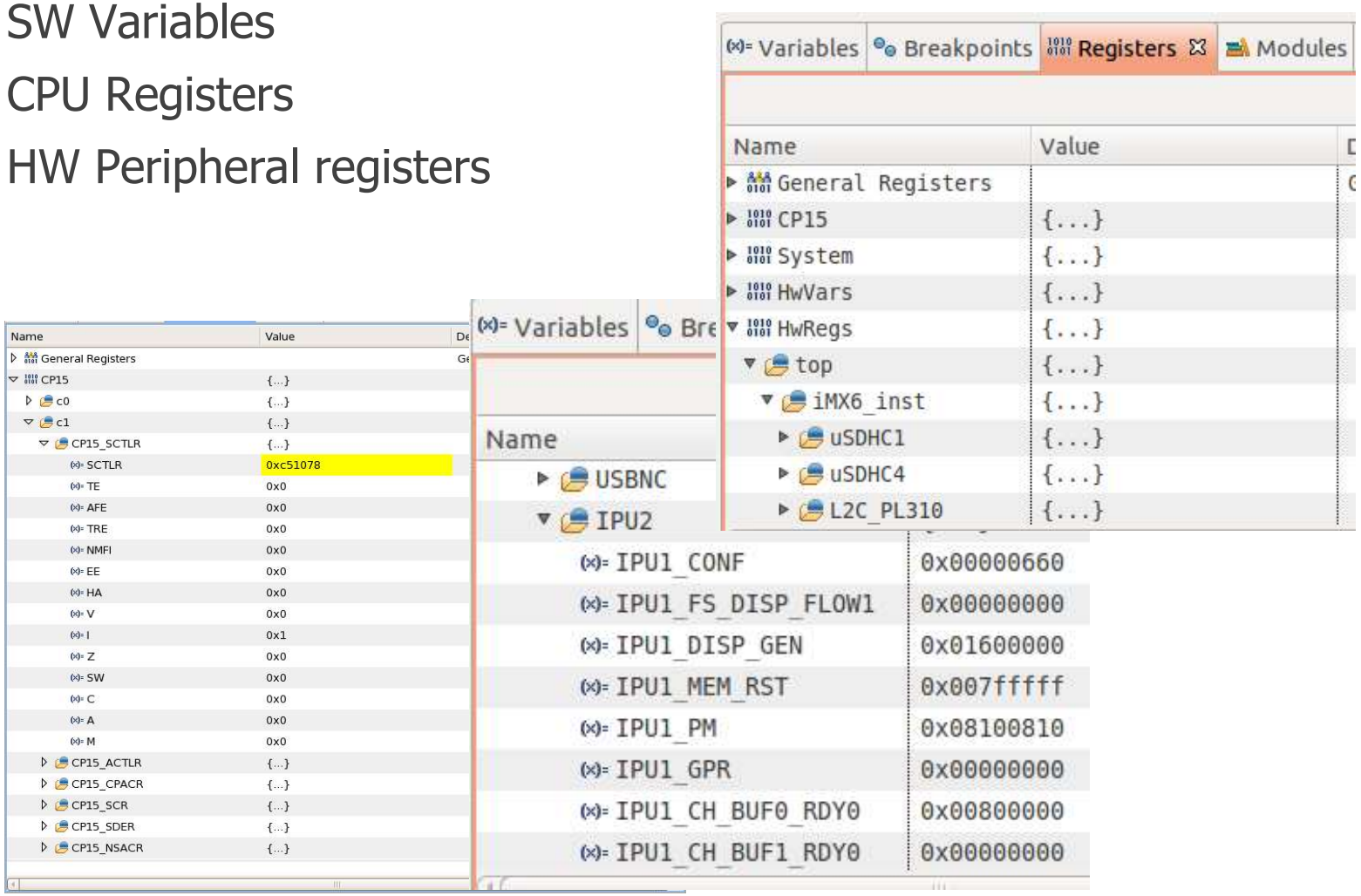
Linux Kernel Debug

- GDB Server connection per core
- Connection independent of HW platform
- Step through driver code



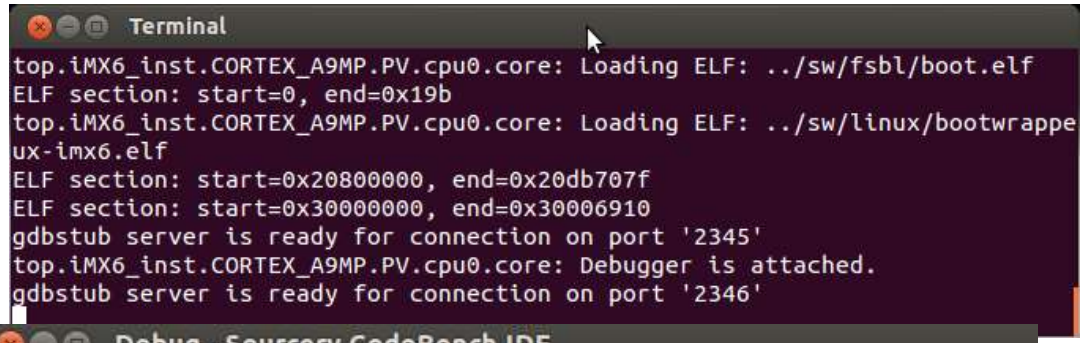
Visibility

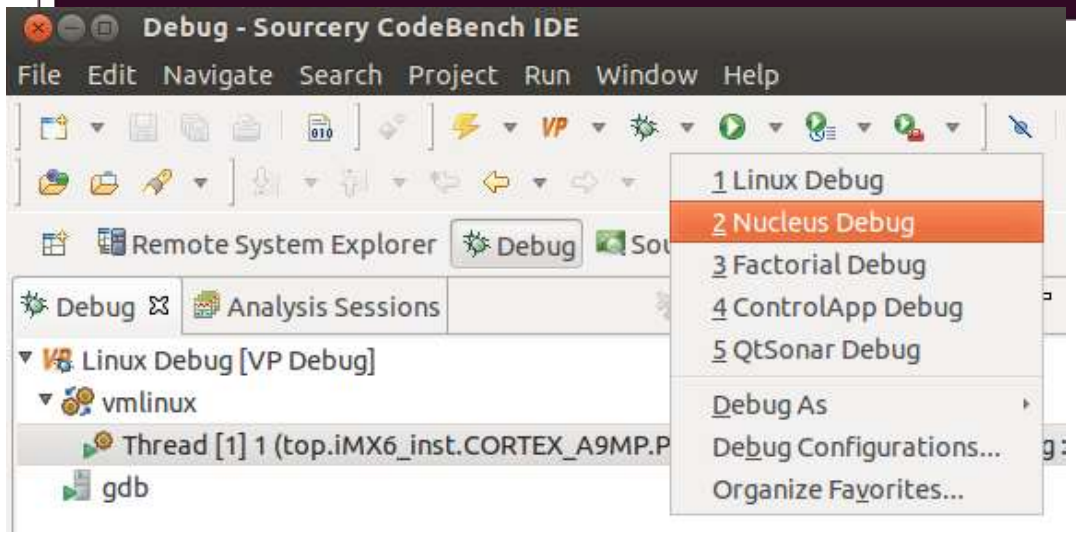
- View
  - SW Variables
  - CPU Registers
  - HW Peripheral registers



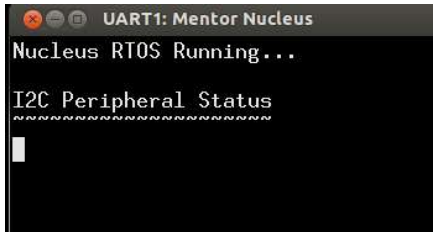
Nucleus Debug

- Core specific debug connection
- Debug and trace specific to RTOS
- Deterministic execution in debug or free running



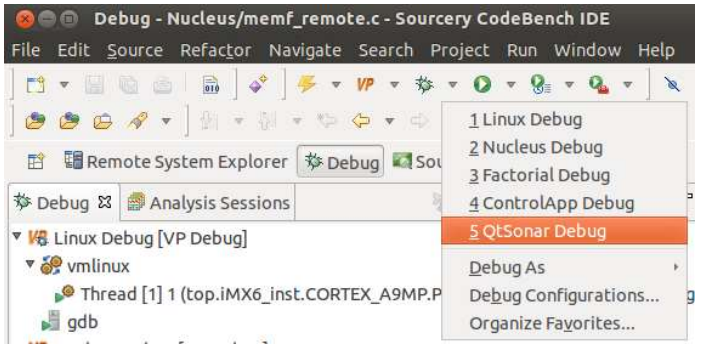


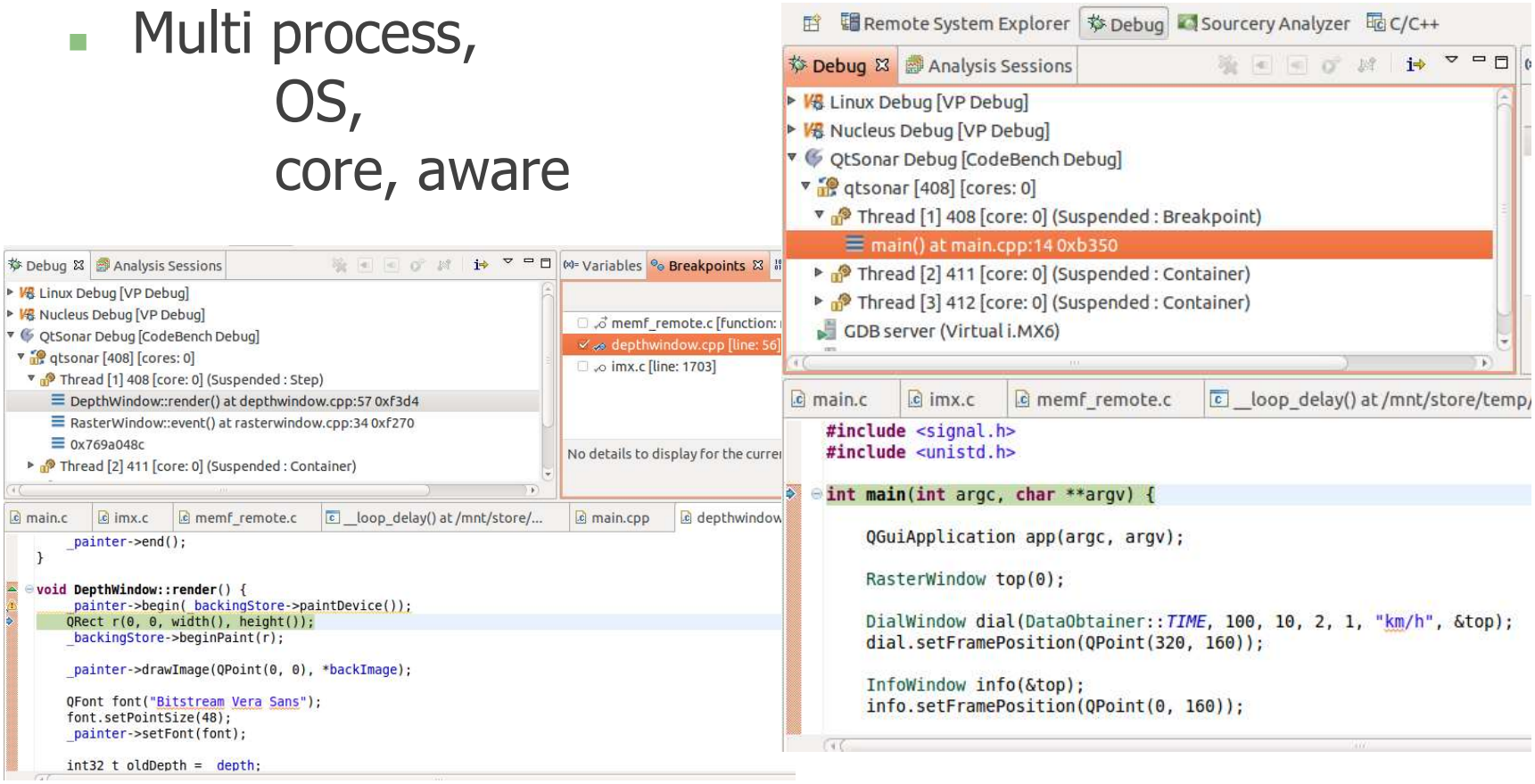

```
static VOID nucleus_printInfo() {
    int r;
    printf("Nucleus RTOS Running...\r\n");
    printf("\r\n");
    printf("I2C Peripheral Status\r\n");
    printf("~~~~~~~~~~~~~~~~~~~~~~\r\n");
    printf("  Sonar, Depth       : %" PRIu32 "\r\n", dp.depth);
    printf("  GPS, Milliseconds  : %" PRIu32 "\r\n", dp.timeval);
    printf("  GPS, Latitude      : %" PRIu32 "\r\n", dp.latitude);
    printf("  GPS, Longitude     : %" PRIu32 "\r\n", dp.longitude);
    if (lastAlarm == 0x1) {
        printf("  Alarm              : >>> RAISED <<<\r\n");
    } else {
```



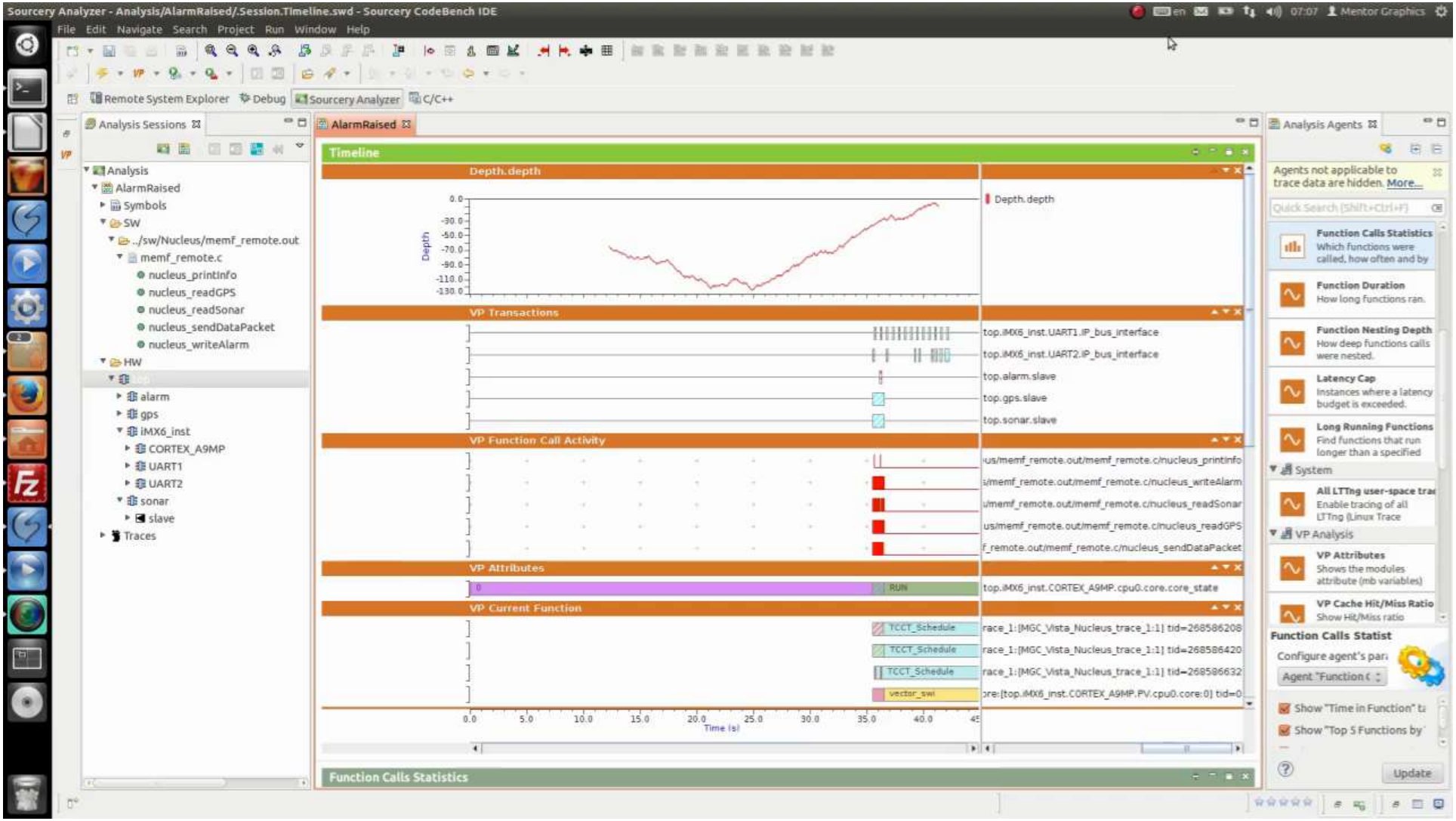
Application Debug

- Launch application via Ethernet
  - Full network interface
- Multi process,
  OS,
  core, aware





Analyzing the Virtual Platform

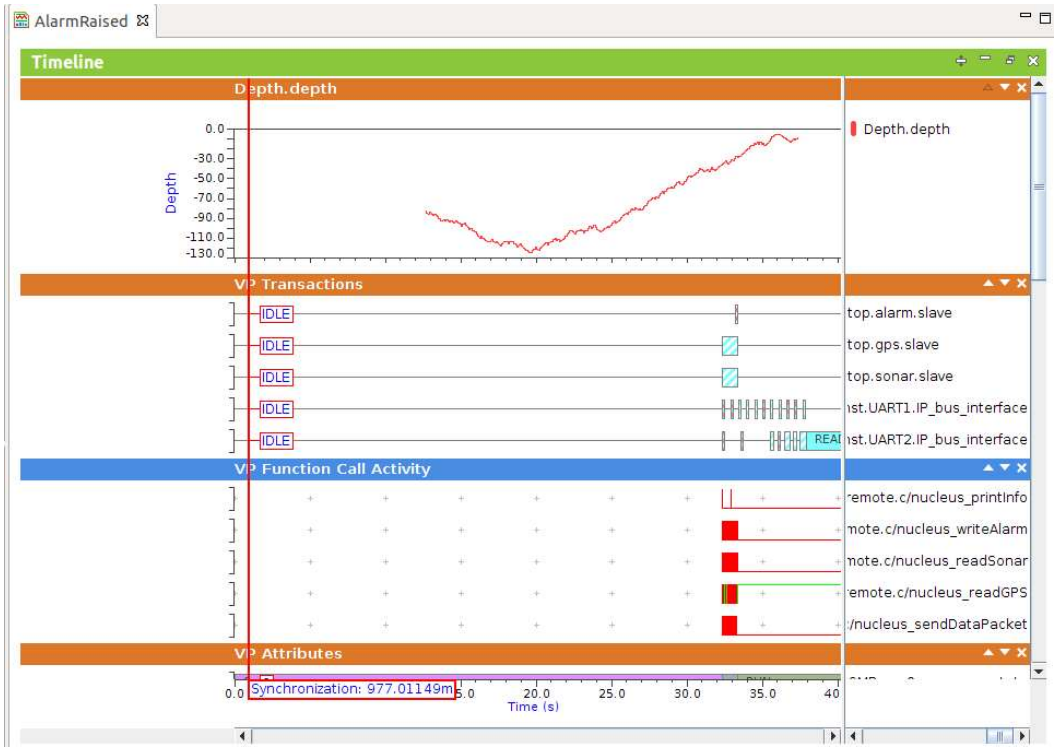


Record

- Dynamic trace control

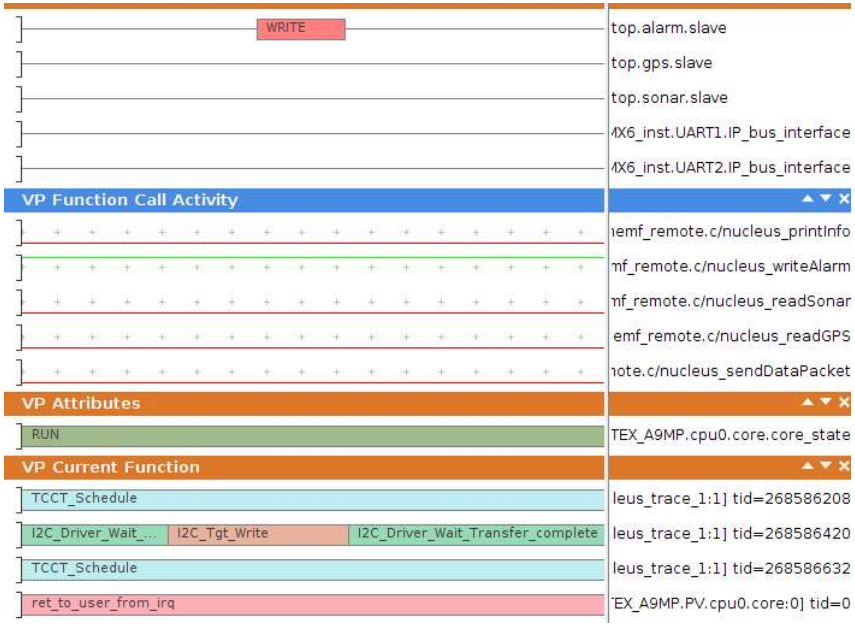
```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Tracing in Linux

select_core "top.iMX6_inst.CORTEX_A9MP.PV.cpu0.core"

trace_linux ../sw/linux/mel/vmlinux -libc ../sw/linux/mel/libc-2.20.so

trace_function_calls -kind eff -tag LINUX_FUNCTION_CALLS -disabled
trace_current_function -kind eff -tag LINUX_CURRENT_FUNCTION -disabled

trace_linux_process controlapp {
    add_symbol_file ../sw/ControlApp/controlapp

    trace_function_calls -kind eff -tag CONTROL_FUNCTION_CALLS

    insert_tracepoint controlapp_tp1 -at-function-entry main -do-tcl {
        puts "NIT >>> Control App, main function entered"
    }

    insert_tracepoint controlapp_tp1 -at-function-entry startReceiving -do-tcl {
        puts "NIT >>> Control App, startReceiving - ENABLING TRACING"
        enable_tag HW_LINUX_CORE
        enable_tag HW_I2C_SONAR
        enable_tag HW_I2C_GPS
        enable_tag HW_I2C_ALARM
        enable_tag HW_UART1
        enable_tag HW_UART2
        enable_tag NUCLEUS_FUNCTION_CALLS
        enable_tag NUCLEUS_FUNCTION_ACTIVITY
        enable_tag NUCLEUS_CURRENT_FUNCTION
        enable_tag LINUX_FUNCTION_CALLS
        enable_tag LINUX_CURRENT_FUNCTION
        enable_tag QT_FUNCTION_CALLS
    }

    insert_tracepoint controlapp_tp2 -at-function-entry startReceiving -do-raw {
        printf("NIT >>> Control App, DYNAMIC timing mode\n");
        set_parameter("top.iMX6_inst.I2C1.lt_policy_modeling", "DYNAMIC");
        set_parameter("top.sonar.lt_policy_modeling", "DYNAMIC");
        set_parameter("top.gps.lt_policy_modeling", "DYNAMIC");
        set_parameter("top.alarm.lt_policy_modeling", "DYNAMIC");
    }
```

- SW Variables
- HW Transactions
- Function calls



Interact

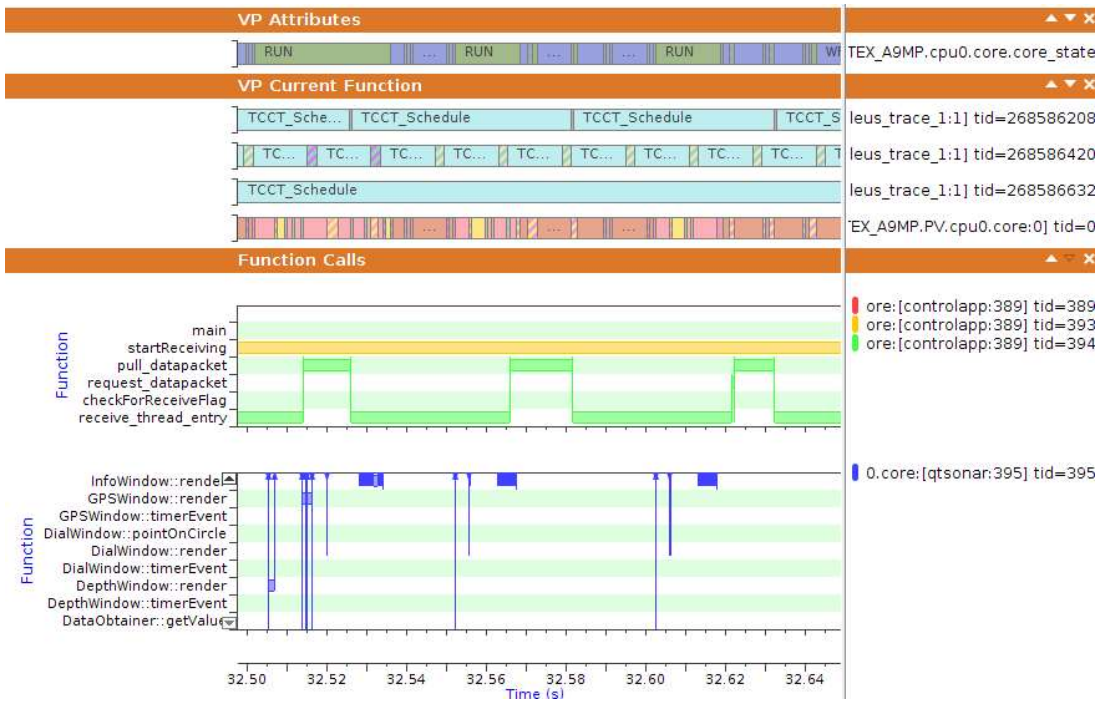
- Trace detailed register activity
- Correlate to SW



- Deterministic control of time

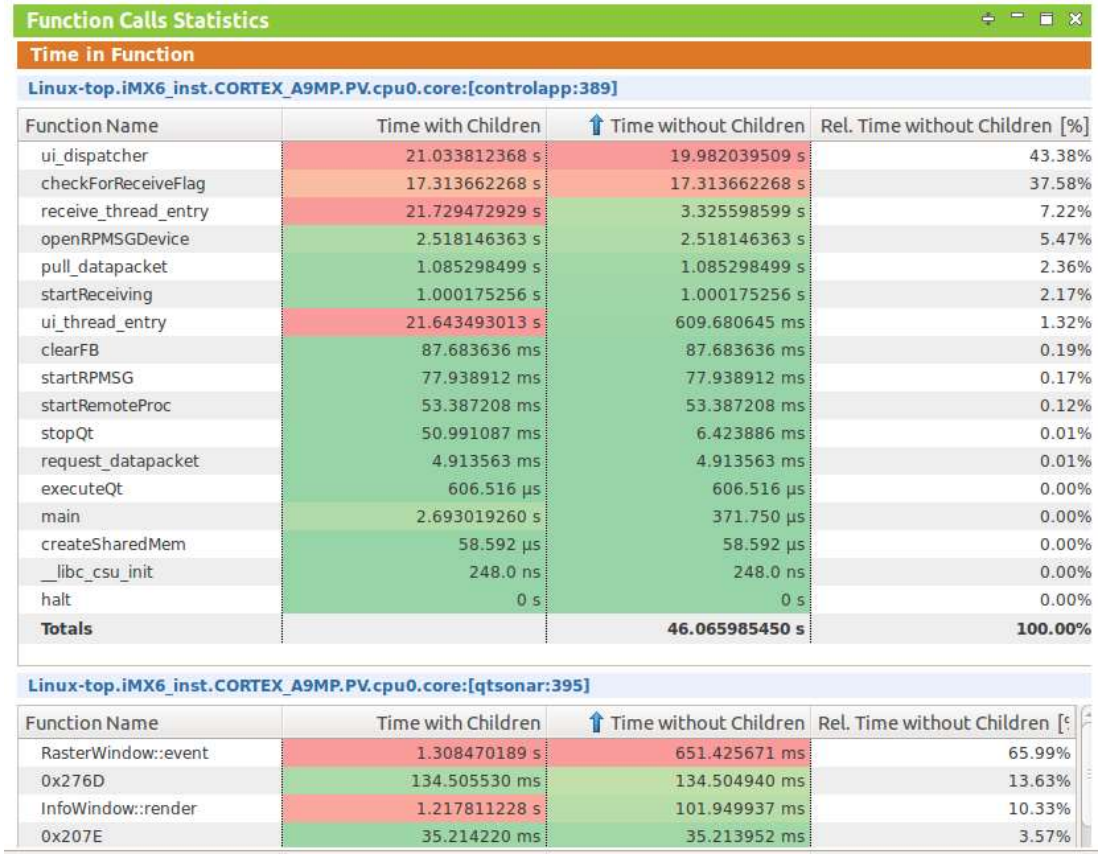
Visualize

- System Status
- RTOS Process activity
- Linux applications
  - Control App
  - Qt App



Statistics

- Bare Metal
- Core Activity
- Linux Process Activity



Function Calls Statistics

Time in Function

Linux-top.iMX6_inst.CORTEX_A9MP.PV.cpu0.core:[controlapp:389]

Function Name	Time with Children	Time without Children	Rel. Time without Children [%]
ui_dispatcher	21.033812368 s	19.982039509 s	43.38%
checkForReceiveFlag	17.313662268 s	17.313662268 s	37.58%
receive_thread_entry	21.729472929 s	3.325598599 s	7.22%
openRPMSGDevice	2.518146363 s	2.518146363 s	5.47%
pull_datapacket	1.085298499 s	1.085298499 s	2.36%
startReceiving	1.000175256 s	1.000175256 s	2.17%
ui_thread_entry	21.643493013 s	609.680645 ms	1.32%
clearFB	87.683636 ms	87.683636 ms	0.19%
startRPMSG	77.938912 ms	77.938912 ms	0.17%
startRemoteProc	53.387208 ms	53.387208 ms	0.12%
stopQt	50.991087 ms	6.423886 ms	0.01%
request_datapacket	4.913563 ms	4.913563 ms	0.01%
executeQt	606.516 µs	606.516 µs	0.00%
main	2.693019260 s	371.750 µs	0.00%
createSharedMem	58.592 µs	58.592 µs	0.00%
__libc_csu_init	248.0 ns	248.0 ns	0.00%
halt	0 s	0 s	0.00%
Totals		**46.065985450 s**	**100.00%**

Linux-top.iMX6_inst.CORTEX_A9MP.PV.cpu0.core:[qtsonar:395]

Function Name	Time with Children	Time without Children	Rel. Time without Children [
RasterWindow::event	1.308470189 s	651.425671 ms	65.99%
0x276D	134.505530 ms	134.504940 ms	13.63%
InfoWindow::render	1.217811228 s	101.949937 ms	10.33%
0x207E	35.214220 ms	35.213952 ms	3.57%

Questions:

Thank You!