

MCUXpresso IDE Debug Features for Bare-Metal on EVKB-i.MXRT1050

1) Fundamentals: What “debug” means on a Cortex-M7, and what MCUXpresso provides

1.1 Definitions

- **Bare-metal:** Software running directly on the microcontroller with no operating system. You control reset, clocks, pin mux, interrupts, memory, and application logic.
- **CoreSight:** ARM® debug/trace architecture present in Cortex-M7. It exposes hardware resources (breakpoints, watchpoints, trace) over JTAG/SWD.
- **EVKB-i.MXRT1050:** NXP evaluation kit for the i.MX RT1050 crossover MCU (Cortex-M7 @ up to 600 MHz).
- **CMSIS-DAP (Cortex Microcontroller Software Interface Standard – Debug Access Port):** The on-board debug probe interface on EVKB used by MCUXpresso’s LinkServer.
- **ITM/SWO (Instrumentation Trace Macrocell / Single-Wire Output):** Low-intrusion, hardware-assisted message/printf and event streaming over a single pin.
- **ETM (Embedded Trace Macrocell):** Instruction trace (not always routed/available on all boards without external hardware).
- **DWT (Data Watchpoint and Trace):** Hardware comparators/counters for watchpoints and cycle counting.
- **FPB (Flash Patch and Breakpoint):** Hardware instruction breakpoints in flash.
- **LPUART (Low-Power Universal Asynchronous Receiver/Transmitter):** i.MX RT1050 UART peripheral used in our ARINC 429 lab.
- **DO-178C (Software Considerations in Airborne Systems and Equipment Certification):** Primary avionics software standard; requires determinism, evidence, and traceability.
- **ARINC 429 (Aeronautical Radio, Incorporated 429):** Unidirectional 2-wire data bus used in avionics. We will ingest it via a UART bridge (ADK-8582) for labs.

1.2 CoreSight resources we will use

- **PC breakpoints:** Halt when Program Counter reaches an address/line.
- **Watchpoints (DWT):** Halt on read/write of a specific memory location; ideal to catch unexpected variable changes without instrumentation.

- **Single-step controls:** Step-into/over/out; run-to-line.
- **Core & Peripheral register access:** Inspect CPU (xPSR, PRIMASK, BASEPRI) and SoC peripheral registers live.
- **ITM/SWO prints:** Real-time logging with minimal timing impact.
- **DWT cycle counter:** Cycle-accurate timing for latency/jitter bounds.

Why it matters for avionics: Use hardware-assisted observability (watchpoints, SWO, cycle counter) to gather objective evidence with minimal perturbation—preferable to intrusive printf over UART.

2) MCUXpresso IDE: Your debugging workbench

Key views/panels: - **Quickstart → Debug:** Launch session via **LinkServer (CMSIS-DAP)**. - **Breakpoints:** Add/enable/disable line breakpoints; create **Watchpoints** (DWT data breakpoints). - **Expressions:** Watch C variables and raw addresses; enable *continuous refresh* for live variables. - **Peripherals+ (NXP):** Decoded peripheral registers (GPIO, IOMUXC, LPUART, etc.). - **Memory:** Hex/ASCII view, selectable widths. - **Console(s):** MCUXpresso console for semihosting; **SWO Trace Console** for ITM/SWO. - **Disassembly:** Interleaved C + instructions; useful for instruction-level tracing. - **Registers:** CPU core regs, fault status (CFSR, HFSR), fault addresses (BFAR/MMFAR).

Reset/connect modes: - **Halt on reset:** Recommended for low-level bring-up. - **Attach to running:** Non-intrusive attach to an already running target. - **System vs Core reset:** System reset reinitializes peripherals; Core reset is faster but preserves peripheral state.

Semihosting vs SWO: - **Semihosting** uses BKPT traps → intrusive, can stall; OK for early bring-up only. - **SWO** provides low-intrusion, high-bandwidth telemetry; preferred for timing-sensitive work.

3) Hands-On Labs (SDK-based, compile-ready)

Each lab includes a **generic example** and an **avionics use case**. Paths are relative to the NXP SDK for EVKB-i.MXRT1050.

Lab 1 — Breakpoints, stepping, Peripherals+ (LED Blinky)

Project: boards/evkbimxrt1050/demo_apps/led_blinky/

Steps: 1. Import into MCUXpresso (File → Import → Existing Projects into Workspace). 2. Build and **Debug** with **LinkServer (CMSIS-DAP)**. 3. Set a line breakpoint inside led_blinky.c where GPIO_PinWrite() toggles the LED. 4. Step-over and observe logic. 5. Open **Peripherals+ → GPIO** for the LED port; watch the **DR (data) register** bit change as

you step. 6. Add the LED variable or register address to **Expressions** and enable live refresh. 7. Compare **System Reset** vs **Core Reset** and note peripheral state differences.

Generic example: Verify a basic state machine that toggles the LED only when a condition is true.

Avionics use case: Treat the LED as a *discrete output line* driving a **Weight-On-Wheels (WOW)** signal. Place a breakpoint at the WOW decision gate and verify that debounced state transitions alone modify the discrete output. This mirrors verifying a safety-critical discrete output path before enabling downstream functions.

Evidence to save: Peripherals+ screenshot of GPIO bit toggling and a note on which reset mode you used and why.

Lab 2 — Low-intrusion logging with ITM/SWO

Project: boards/evkbimxrt1050/demo_apps/hello_world_swo/

Demo behavior: Board init enables trace clocks; SWO console prints messages from `PRINTF()` configured for SWO (see `hardware_init.c` and `app.h`).

Steps: 1. Import, build, and debug the project. 2. Open **SWO Trace Console**; configure to the demo's SWO baud (default 4 MHz in `app.h`). 3. Trigger button events and observe SWO messages; observe periodic SysTick prints. 4. Set a breakpoint in the button's GPIO interrupt handler; step and inspect **Registers** (PRIMASK/BASEPRI) to confirm interrupt masking state.

Generic example: Replace button events with a small state machine and log transitions via SWO.

Avionics use case: Use SWO as maintenance/logging telemetry in an **Air Data Computer** prototype—for example, log “pitot fault injected” events without disturbing UART bandwidth budgets needed elsewhere (e.g., an ARINC 429 bridge).

Why SWO: Lower intrusion than semihosting or UART prints; suitable for timing-sensitive paths.

Lab 3 — Watchpoints + UART RX with ADK-8582 (ARINC 429 over UART)

Goal: Use a **DWT watchpoint** to halt exactly when a specific ARINC 429 label arrives from an external converter (ADK-8582 → UART → EVKB). Step ISR and inspect a ring buffer.

Project: boards/evkbimxrt1050/driver_examples/lpuart/interrupt_transfer/

Physical setup: Configure **ADK-8582** to output ARINC 429 32-bit words as ASCII hex lines: XXXXXXXX\r\n, **115200-8-N-1** into EVKB LPUART. Wire **GND↔GND**, **ADK TX → EVKB RX**.

Parsing fields (SDK-style C):

```
/* ARINC 429 field extraction (Labels commonly documented in OCTAL).
   For this lab treat 'label' as the low 8 bits of the 32-bit word. */
typedef struct {
    uint8_t label;      // bits 1..8 (LSB)
    uint8_t sdi;        // bits 9..10
    uint32_t data;      // bits 11..29
    uint8_t ssm;        // bits 30..31
    bool parity;        // bit 32 (MSB)
} arinc429_t;

static inline arinc429_t parse_arinc429(uint32_t w)
{
    arinc429_t f;
    f.label = (uint8_t)(w & 0xFFu);
    f.sdi = (uint8_t)((w >> 8) & 0x3u);
    f.data = (uint32_t)((w >> 10) & 0x1FFFFFFu);
    f.ssm = (uint8_t)((w >> 29) & 0x3u);
    f.parity = ((w >> 31) & 0x1u) ? true : false;
    return f;
}
```

Probe variables for watchpoints:

```
volatile arinc429_t g_lastRx;
volatile uint8_t g_lastLabel; // convenience copy for watchpoints
```

SDK-pattern LPUART init (based on fs1_lpuart.h example):

```
lpuart_config_t config;
LPUART_GetDefaultConfig(&config);
config.baudRate_Bps = 115200U;
config.enableRx = true;
config.enableTx = true;
LPUART_Init(DEMO_LPUART, &config, CLOCK_GetFreq(kCLOCK_UartClk));

static lpuart_handle_t g_lpuartHandle;
LPUART_TransferCreateHandle(DEMO_LPUART, &g_lpuartHandle,
LPUART_UserCallback, NULL);
// Start first receive (non-blocking)
static uint8_t ch;
lpuart_transfer_t rx = { .data = &ch, .dataSize = 1 };
LPUART_TransferReceiveNonBlocking(DEMO_LPUART, &g_lpuartHandle, &rx, NULL);
```

In the RX path, assemble a line → parse → update probe vars:

```

#define RX_LINE_MAX 16
static uint8_t rxLine[RX_LINE_MAX];
static size_t rxLen = 0;

void LPUART_UserCallback(LPUART_Type *base, lpuart_handle_t *handle,
                        status_t status, void *userData)
{
    if (kStatus_LPUART_RxIdle == status) {
        // When IDLE, check if we have a full line like "XXXXXXXX\n"
        if (rxLen >= 8) {
            rxLine[8] = '\0';
            uint32_t raw = (uint32_t)strtoul((const char*)rxLine, NULL, 16);
            g_lastRx = parse_arinc429(raw);
            g_lastLabel = g_lastRx.label;
        }
        rxLen = 0; // prepare for next word
    }
}

```

Create a DWT Watchpoint in MCUXpresso: 1. Debug the app. 2. **Breakpoints** view → **New** → **Watchpoint**. 3. Address: symbol `g_lastLabel`. Size: 1 byte. Condition: **Break on write**. 4. Resume. The core halts exactly when a new label is parsed.

Avionics use case: Track **baro-corrected altitude** using **label 203 (octal)** = decimal 131 = 0x83. Stop when `g_lastLabel == 0x83` (or add a conditional BKPT). This proves your **Input Data Concentrator** correctly detects the required label without UART printf jitter. Use **Peripherals+ → LPUART** to confirm FIFO/status bits.

Evidence to save: Screenshot of watchpoint hit (`g_lastLabel == 0x83`) plus LPUART status view.

Lab 4 — Timing analysis with DWT Cycle Counter + SWO

Goal: Measure ISR latency and code section execution time at cycle precision.

Helpers:

```

#include "fsl_device_registers.h" // brings in DWT/DEMCR definitions

static inline void DWT_InitCycleCounter(void)
{
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // enable DWT/ITM
    DWT->CYCCNT = 0;
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;
}

static inline uint32_t DWT_GetCycles(void) { return DWT->CYCCNT; }

```

Use in code:

```
DWT_InitCycleCounter();
uint32_t t0 = DWT_GetCycles();
// section under test
uint32_t dt = DWT_GetCycles() - t0;
PRINTF("section cycles=%u us=%u\r\n", dt,
        (unsigned)((uint64_t)dt * 1000000ULL / SystemCoreClock));
```

Avionics use case: Bound latency from **UART word arrival** (RX complete) to **integrity check completion** (parity/SSM verification). Record min/mean/max over $\geq 10,000$ events via SWO. Convert cycles to microseconds using SystemCoreClock and attach results as WCET evidence for a DAL-B/C prototype.

4) Deep-Dive Techniques

4.1 Breakpoints (software vs hardware)

- Software breakpoints patch BKPT into flash (managed by IDE); hardware breakpoints use FPB comparators. If you run out, remove some or use *Run to Line*.

4.2 Watchpoints (DWT)

- Watchpoints trigger on read/write/modify of aligned addresses (1/2/4 bytes). Mark variables `volatile` to prevent optimization from eliding them.

4.3 Fault analysis (SCB registers)

- Inspect **CFSR**, **HFSR**, **BFAR**, **MMFAR** when stopping in HardFault to identify the faulting access and instruction. Combine with **Disassembly view**.

4.4 SWO Trace Console tips

- Match SWO baud to project (app.h in sample uses 4 MHz). Ensure trace clock enabled in board init. Prefer SWO over UART inside ISRs.

4.5 Reset strategy

- Use **System Reset + Halt** when changing clock trees/pin mux for a clean start. Use **Attach to running** for endurance tests that must not be reset.

4.6 Linker Map & memory insight

- Open the **.map** file to verify code/data placement. Track BSS/stack headroom during stress tests.
-

5) Strong Avionics Scenarios

1. **IOMUXC mis-mux / drive strength:** With Peripherals+ open, verify pad mux & drive for an **ARINC 429 TX-enable** pin before enabling the line driver. Watchpoint on the GPIO direction register guarantees output mode is set exactly once.
 2. **UART framing errors in noise:** In **Peripherals+ → LPUART**, monitor FE/NE/PE flags while ADK-8582 streams data. Watchpoint a shadowed status variable to halt on FE and dump a short diagnostic.
 3. **Timing regression after refactor:** Wrap the ARINC label parser with DWT cycle timing. If cycles exceed a threshold, issue a SWO alert and log a timestamp.
 4. **Induced HardFault boundary test:** Intentionally overrun a test buffer; on HardFault, capture PC/LR/CFSR/BFAR in a minimal fault handler. This becomes part of your laboratory safety case.
-

6) Practical SDK-aligned Snippets

6.1 SWO init/prints (mirrors hello_world_swo)

```
#include "fsl_debug_console.h"
#include "board.h"
#include "app.h"

int main(void)
{
    BOARD_InitHardware();
    BOARD_InitDebugConsoleSwo(DEMO_DEBUG_CONSOLE_SWO_PORT,
    DEMO_DEBUG_CONSOLE_SWO_BAUDRATE);
    while (1) {
        PRINTF("Swo alive\r\n");
        for (volatile uint32_t i=0; i<1000000; ++i) {}
    }
}
```

6.2 LPUART non-blocking receive (SDK driver pattern)

```
#include "fsl_lpuart.h"
#include "board.h"
#include "app.h"

#define RX_LINE_MAX 16
static uint8_t rxLine[RX_LINE_MAX];
static size_t rxLen = 0;

volatile arinc429_t g_lastRx;
volatile uint8_t g_lastLabel;

static lpuart_handle_t g_lpuartHandle;
```

```

void LPUART_UserCallback(LPUART_Type *base, lpuart_handle_t *handle,
                        status_t status, void *userData)
{
    if (kStatus_LPUART_RxIdle == status) {
        if (rxLen >= 8) {
            rxLine[8] = '\0';
            uint32_t raw = (uint32_t)strtoul((const char*)rxLine, NULL, 16);
            g_lastRx = parse_arinc429(raw);
            g_lastLabel = g_lastRx.label;
        }
        rxLen = 0;
    }
}

static void uart_init_and_start(void)
{
    lpuart_config_t config;
    LPUART_GetDefaultConfig(&config);
    config.baudRate_Bps = 115200U;
    config.enableRx = true;
    config.enableTx = true;
    LPUART_Init(DEMO_LPUART, &config, CLOCK_GetFreq(kCLOCK_UartClk));

    LPUART_TransferCreateHandle(DEMO_LPUART, &g_lpuartHandle,
    LPUART_UserCallback, NULL);

    static uint8_t ch;
    lpuart_transfer_t rx = { .data = &ch, .dataSize = 1 };
    LPUART_TransferReceiveNonBlocking(DEMO_LPUART, &g_lpuartHandle, &rx,
    NULL);
}

```

6.3 DWT cycle counter helper

```

static inline void DWT_InitCycleCounter(void) {
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
    DWT->CYCCNT = 0;
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;
}

```

7) Best Practices (Airbus-grade)

1. **Prefer non-intrusive observability.** Use watchpoints and SWO prints; avoid semihosting and UART prints in critical paths.
2. **Freeze build settings per test.** Optimization/inlining impact code placement and evidence reproducibility. Record -O level, LTO, and linker options.

3. **Make probe state explicit.** Mark watched variables volatile; keep them in RAM and visible to the debugger.
 4. **Clocks & pins as single source of truth.** Step through `clock_config.c` and `pin_mux.c` once per board/BSP change while watching Peripherals+.
 5. **Reset discipline.** Use System Reset when changing clocks/mux; document reset method in your evidence set.
 6. **Bound worst-case, not average.** Use DWT cycles for WCET claims; SWO timestamps are supportive but not primary.
 7. **Faults as first-class artifacts.** Include a tiny HardFault report (PC, LR, CFSR, BFAR) for lab builds; disable for flight.
 8. **ARINC labels: be explicit with bases.** Document label numbers in **octal and hex** (e.g., 203(oct) = 0x83) to prevent confusion.
 9. **Probe bandwidth awareness.** Tune SWO baud (demo uses 4 MHz). If you outgrow SWO, move to external trace rather than more UART prints.
 10. **Configuration control.** Archive launch configs (.launch), map files, and SWO logs with test reports to support DO-178C reviews.
-

8) Session Checklist

- ☐ EVKB power and CMSIS-DAP (LinkServer) enumerated.
 - ☐ Launch configuration: **Halt on reset**, correct **SWO baud**, semihosting **disabled** (unless used intentionally).
 - ☐ Optimization level noted (-O0 for bring-up; -O2 for timing runs) and unchanged during evidence capture.
 - ☐ Variables for watchpoints marked volatile and in scope.
 - ☐ ARINC lab: ADK-8582 at **115200-8-N-1**, ASCII hex framing; GND common; TX→RX wired.
-

9) What to Submit (Recommended Artifacts)

1. **Lab 1:** Peripherals+ screenshot of GPIO bit toggling; note on reset mode and reasoning.
 2. **Lab 2:** SWO console capture of timed events; screenshot stepping inside GPIO ISR showing PRIMASK/BASEPRI state.
 3. **Lab 3:** Watchpoint hit on `g_lastLabel == 0x83` with LPUART status; brief throughput/overflow note.
 4. **Lab 4:** Min/mean/max cycles for ISR path, SystemCoreClock value, computed microseconds, and test conditions.
-

Appendix A — Troubleshooting SWO on EVKB-i.MXRT1050

- Ensure board init enables TRACE clock (the SWO demo's `hardware_init.c` does this).
- In **SWO Trace Console**, match the baud rate from `app.h` (default 4 MHz). If output is garbled, try lower rates (e.g., 2 MHz).
- Disable semihosting in the launch configuration if the target stalls unexpectedly.

Appendix B — Minimal HardFault Report (for lab builds)

```
void HardFault_Handler(void)
{
    volatile uint32_t hfsr = SCB->HFSR;
    volatile uint32_t cfsr = SCB->CFSR;
    volatile uint32_t bfar = SCB->BFAR;
    (void)hfsr; (void)cfsr; (void)bfar; // inspect in debugger or print via
SWO if safe
    __BKPT(0);
    while (1) { }
}
```