Task Registration and Scheduling (Bare-Metal): Scheduler Integration and Priorities

---

1) Why a scheduler on bare metal?

On a bare-metal system, a **task** is simply a function that performs a bounded unit of work, usually periodically or upon an event. A **scheduler** is the policy and mechanism that decides which task runs *now*. Without an RTOS (Real-Time Operating System), you still need determinism:

- **Periodic activities** (e.g., health monitoring, sensor sampling),
- **Sporadic/aperiodic activities** (e.g., bytes arriving on UART),
- **Background activities** (e.g., maintenance logging).

Two common patterns without an RTOS:

1. **Cyclic executive (time-triggered)**: a timer tick sets readiness; the main loop runs ready tasks in **fixed priority** order. Cooperative (a running task is not preempted by another task), simple, analyzable, and cert-friendly.
2. **Interrupt-driven with deferred work**: interrupts do the minimum (buffer, set flags), while the main loop performs heavier processing. Priorities come from **NVIC (Nested Vectored Interrupt Controller)** for interrupts and from your task-order policy in the main loop.

For avionics consistent with **DO-178C (Software Considerations in Airborne Systems and Equipment Certification)**, we prefer: **static registration**, **no dynamic allocation after init**, **bounded execution per activation**, and **measurable timing**.

---

2) On-chip building blocks you will use

- **PIT (Periodic Interrupt Timer)** or **GPT (General Purpose Timer)** for a base tick. See SDK examples:
  - `boards/evkbimxrt1050/driver_examples/pit/pit.c`
  - `boards/evkbimxrt1050/driver_examples/gpt/timer/gpt_timer.c`
- **NVIC**: prioritizes interrupts (lower numeric priority value ⇒ higher urgency on Cortex-M). CMSIS provides `__NVIC_PRIO_BITS` and `NVIC_SetPriority`.
- **LPUART**: low-power UART used to receive ARINC 429 words from ADK-8582. See SDK examples under `driver_examples/lpuart/*`.
- **DWT (Data Watchpoint and Trace)** cycle counter for microsecond timing on Cortex-M7; excellent for jitter/latency measurement during labs.

3) Fixed-priority scheduling without an RTOS

We implement a **static task registry**: a table where each entry defines name, function pointer, period (ms), **fixed priority** (0..255, larger means higher in our policy), and an optional WCET (Worst-Case Execution Time) hint. The scheduler provides:

- A **1 kHz** (configurable) tick via **PIT** that marks periodic tasks ready at release times.
- A **main dispatcher** that continuously scans priorities from high to low, running ready tasks **to completion** (cooperative). When nothing is ready, the CPU sleeps with **WFI (Wait-For-Interrupt)**.

This pattern is deterministic and analyzable provided every task does **bounded** work per activation.

---

4) Priority assignment you can justify

For periodic tasks with deadlines equal to periods, **RMS (Rate-Monotonic Scheduling)** says: shorter period ⇒ higher priority. In avionics, combine RMS with **criticality**: some safety monitors may sit above parsers even if their period is longer when the safety case demands it.

Important: Task priorities (handled by the dispatcher) are entirely separate from **NVIC** priorities (hardware interrupt preemption). ISRs (Interrupt Service Routines) always preempt tasks. Keep ISRs short; defer heavy work to tasks.

---

5) Implementation: Bare-metal scheduler (drop-in for EVKB-i.MXRT1050)

Below are production-style files compatible with the EVKB SDK. They avoid dynamic allocation and rely on `fsl_pit.h`, `fsl_gpt.h`, and CMSIS.

5.1 `bm_sched.h`

```c
/* bm_sched.h — Bare-metal fixed-priority scheduler (EVKB-IMXRT1050) */
#pragma once
#include <stdint.h>
#include <stddef.h>

#ifdef __cplusplus
extern "C" {
#endif
```

```c
typedef void (*bm_task_fn_t)(void);

/* Priority: larger number = higher priority (independent of NVIC) */
typedef struct {
    const char  *name;
    bm_task_fn_t run;
    uint32_t     period_ms;    /* 0 => aperiodic (manually released) */
    uint8_t      prio;         /* 0..255, we'll scan high→low */
    uint32_t     wcet_us;      /* instrumentation hint (0 if unknown) */
    /* Internal (do not touch after registration) */
    volatile uint8_t ready;
    uint32_t     next_release_tick;
    uint32_t     last_start_us;
    uint32_t     last_finish_us;
    uint32_t     overruns;
} bm_task_t;

void BM_Sched_Init(uint32_t tick_hz);
void BM_Sched_Register(bm_task_t *tasks, size_t count);
void BM_Sched_Run(void);

/* Release an aperiodic task from ISR or main. */
void BM_Sched_Release(bm_task_t *t);

/* Must be called by the 1 kHz timer ISR (PIT). */
void BM_Sched_Tick_ISR(void);

/* Time utils (implemented with GPT/PIT/ARM DWT if enabled) */
uint32_t BM_Ticks(void);          /* ms ticks since init */
uint32_t BM_TimeUs(void);         /* microseconds since boot (monotonic) */

#ifdef __cplusplus
}
#endif
```

5.2 bm_sched.c

```c
/* bm_sched.c — uses PIT for 1kHz tick; uses ARM DWT for microsecond timing i
f available */
#include "bm_sched.h"
#include "fsl_device_registers.h"
#include "fsl_common.h"
#include "fsl_pit.h"
#include "fsl_gpt.h"
#include "board.h"

#ifndef BM_MAX_TASKS
#define BM_MAX_TASKS 16
#endif
```

```c
static struct {
    bm_task_t  *tasks[BM_MAX_TASKS];
    size_t      count;
    volatile uint32_t tick_ms;
    uint32_t    tick_hz;
} g_bm;

static inline void dwt_init(void)
{
#if (__CORTEX_M == 7U)
    if ((CoreDebug->DEMCR & CoreDebug_DEMCR_TRCENA_Msk) == 0U)
        CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
    DWT->CYCCNT = 0;
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;
#endif
}

uint32_t BM_Ticks(void) { return g_bm.tick_ms; }

uint32_t BM_TimeUs(void)
{
#if (__CORTEX_M == 7U)
    uint32_t cpu_hz = CLOCK_GetFreq(kCLOCK_CpuClk);
    return (uint32_t)((uint64_t)DWT->CYCCNT * 1000000ULL / cpu_hz);
#else
    /* Fallback: approximate using ms tick */
    return g_bm.tick_ms * 1000U;
#endif
}

void BM_Sched_Init(uint32_t tick_hz)
{
    g_bm.count = 0;
    g_bm.tick_ms = 0;
    g_bm.tick_hz = tick_hz;

    BOARD_InitBootPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    dwt_init();

    /* --- PIT: 1 ms tick (see: boards/evkbimxrt1050/driver_examples/pit/pit.
c) --- */
    pit_config_t pitCfg;
    PIT_GetDefaultConfig(&pitCfg);
    PIT_Init(PIT, &pitCfg);

    uint32_t busHz = CLOCK_GetFreq(kCLOCK_BusClk);
```

```
    uint32_t period_us = 1000000U / tick_hz; /* tick_hz=1000 → 1000 us */
    PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, USEC_TO_COUNT(period_us, busHz));
    PIT_EnableInterrupts(PIT, kPIT_Chnl_0, kPIT_TimerInterruptEnable);

    /* NVIC: give PIT a moderate urgency (leave room for UART RX) */
    NVIC_SetPriority(PIT_IRQn, 7); /* 0=highest urgency on Cortex-M; tune per
system */
    EnableIRQ(PIT_IRQn);

    PIT_StartTimer(PIT, kPIT_Chnl_0);

    /* --- Optional: GPT as free-running microsecond timebase (not strictly n
eeded) --- */
    gpt_config_t gptCfg;
    GPT_GetDefaultConfig(&gptCfg);
    gptCfg.enableFreeRun = true;
    GPT_Init(GPT1, &gptCfg);
    GPT_SetClockDivider(GPT1, 1);
    GPT_StartTimer(GPT1);
}

void BM_Sched_Register(bm_task_t *tasks, size_t count)
{
    for (size_t i = 0; i < count && g_bm.count < BM_MAX_TASKS; ++i) {
        tasks[i].ready = 0;
        tasks[i].next_release_tick = (tasks[i].period_ms ? (BM_Ticks() + task
s[i].period_ms) : 0);
        g_bm.tasks[g_bm.count++] = &tasks[i];
    }
}

/* Called from PIT ISR */
void BM_Sched_Tick_ISR(void)
{
    g_bm.tick_ms++;
    /* Mark periodic tasks ready on their release time */
    for (size_t i = 0; i < g_bm.count; ++i) {
        bm_task_t *t = g_bm.tasks[i];
        if (t->period_ms) {
            if ((int32_t)(g_bm.tick_ms - t->next_release_tick) >= 0) {
                t->ready = 1;
                t->next_release_tick += t->period_ms;
            }
        }
    }
}

void BM_Sched_Release(bm_task_t *t) { t->ready = 1; }
```

```
/* Cooperative fixed-priority dispatcher */
void BM_Sched_Run(void)
{
    for (;;) {
        uint8_t ran = 0;
        /* scan priorities high→low each pass */
        for (int pr = 255; pr >= 0; --pr) {
            for (size_t i = 0; i < g_bm.count; ++i) {
                bm_task_t *t = g_bm.tasks[i];
                if (t->ready && t->prio == (uint8_t)pr) {
                    t->ready = 0;
                    t->last_start_us = BM_TimeUs();
                    t->run();
                    t->last_finish_us = BM_TimeUs();
                    if (t->wcet_us && (t->last_finish_us - t->last_start_us)
> t->wcet_us) {
                        t->overruns++;
                    }
                    ran = 1;
                }
            }
        }
        if (!ran) { __WFI(); } /* idle until next interrupt */
    }
}

/* PIT IRQ — wire to vector table */
void PIT_IRQHandler(void)
{
    if (PIT_GetStatusFlags(PIT, kPIT_Chnl_0) & kPIT_TimerFlag) {
        PIT_ClearStatusFlags(PIT, kPIT_Chnl_0, kPIT_TimerFlag);
        BM_Sched_Tick_ISR();
    }
    __DSB(); __ISB();
}
```

---

6) UART RX integration (ADK-8582) and NVIC priorities

We give **LPUART RX** a **higher urgency** (numerically smaller NVIC priority value) than the PIT tick to avoid RX overrun. The ISR only copies bytes into a **static ring buffer** and returns. A high-priority task in the main loop (ARINC_RX_SVC) drains and reassembles `arinc_word_t` structures.

> Pick an LPUART instance **not used by the debug console** on your EVKB build. If your board init uses LPUART1 for the console, select LPUART2/3 for ADK-8582 and set IOMUXC (Input/Output Multiplexer Controller) accordingly.

6.1 arinc_uart.h

```
#pragma once
#include <stdint.h>
#include <stddef.h>

#ifdef __cplusplus
extern "C" {
#endif

typedef struct {
    uint8_t  label;     /* ARINC 8-bit label (LSB first in bus order) */
    uint8_t  sdi;       /* Source/Destination Identifier (2 bits)     */
    uint32_t data;      /* 19 bits meaningful for many labels         */
    uint8_t  ssm;       /* Sign/Status Matrix (2 bits)                */
    uint8_t  parity_ok; /* odd parity check result                    */
} arinc_word_t;

/* ring buffer interface for RX task */
size_t ARINC_Rx_Drain(uint8_t *dst, size_t maxlen); /* copy raw UART bytes */
void   ARINC_UART_Init(uint32_t baud);              /* initialize LPUART + N
VIC */

#ifdef __cplusplus
}
#endif
```

6.2 arinc_uart.c

```
#include "arinc_uart.h"
#include "fsl_lpuart.h"
#include "fsl_iomuxc.h"
#include "fsl_device_registers.h"
#include "board.h"

/* --- Static ring buffer for UART bytes (ISR <-> task) --- */
#define RX_RING_SIZE 2048
static volatile uint8_t s_rx_ring[RX_RING_SIZE];
static volatile uint32_t s_rx_head = 0, s_rx_tail = 0;

/* Select an LPUART instance dedicated for ADK-8582 */
#define ARINC_LPUART       LPUART1      /* change if LPUART1 is used by conso
le */
#define ARINC_LPUART_IRQn  LPUART1_IRQn

static void arinc_uart_pins_init(void)
{
    /* If board.c already configures your chosen LPUART pins, you may leave t
his empty.
       Otherwise, set IOMUXC here to route the pins for ARINC UART. */
```

```
}

void ARINC_UART_Init(uint32_t baud)
{
    arinc_uart_pins_init();

    lpuart_config_t cfg;
    LPUART_GetDefaultConfig(&cfg);
    cfg.baudRate_Bps = baud;
    cfg.enableRx = true;
    cfg.enableTx = false; /* RX-only path for the bridge */

    LPUART_Init(ARINC_LPUART, &cfg, CLOCK_GetFreq(kCLOCK_UartClk));
    LPUART_EnableInterrupts(ARINC_LPUART, kLPUART_RxDataRegFullInterruptEnabl
e | kLPUART_RxOverrunInterruptEnable);

    /* Give UART higher urgency than PIT (so RX cannot be delayed by the tick
) */
    NVIC_SetPriority(ARINC_LPUART_IRQn, 5); /* numerically lower ⇒ higher ur
gency than PIT's 7 */
    EnableIRQ(ARINC_LPUART_IRQn);
}

void LPUART1_IRQHandler(void)
{
    uint32_t stat = LPUART_GetStatusFlags(ARINC_LPUART);
    /* Read out all available bytes */
    while (stat & kLPUART_RxDataRegFullFlag) {
        uint8_t b = LPUART_ReadByte(ARINC_LPUART);
        uint32_t next = (s_rx_head + 1U) % RX_RING_SIZE;
        if (next != s_rx_tail) { s_rx_ring[s_rx_head] = b; s_rx_head = next;
} /* drop if full */
        stat = LPUART_GetStatusFlags(ARINC_LPUART);
    }
    if (stat & kLPUART_RxOverrunFlag) {
        LPUART_ClearStatusFlags(ARINC_LPUART, kLPUART_RxOverrunFlag);
    }
    __DSB(); __ISB();
}

size_t ARINC_Rx_Drain(uint8_t *dst, size_t maxlen)
{
    size_t n = 0;
    while (n < maxlen && s_rx_tail != s_rx_head) {
        dst[n++] = s_rx_ring[s_rx_tail];
        s_rx_tail = (s_rx_tail + 1U) % RX_RING_SIZE;
    }
    return n;
}
```

7) Application: avionics ARINC 429 use case with realistic priorities

Scenario. The ADK-8582 converts line-side ARINC 429 to a UART stream (e.g., 4 bytes per ARINC 32-bit word, possibly with framing). The EVKB receives bytes via **LPUART RX** ISR → ring buffer. The **ARINC_RX_SVC** task (highest priority in dispatcher) drains, reassembles, checks parity, and buffers completed `arinc_word_t` for the **ARINC_PARSE** task. A **HEALTH_MON** task audits deadlines, queue depths, and feeds the watchdog. **MAINT_LOG** prints occasional status.

NVIC priorities (hardware): LPUART RX IRQ = 5, PIT tick = 7. **Task priorities** (software): `ARINC_RX_SVC` > `HEALTH_MON` > `ARINC_PARSE` > `MAINT_LOG` (example set below).

7.1 `main.c` (integration)

```c
#include "bm_sched.h"
#include "arinc_uart.h"
#include "fsl_gpio.h"
#include "fsl_debug_console.h"
#include "board.h"
#include <string.h>

/* -------------------- RX word FIFO between RX_SVC and PARSE ---------------
----- */
#define RX_WORD_FIFO_LEN 64
static arinc_word_t s_rx_words[RX_WORD_FIFO_LEN];
static volatile uint16_t s_rx_w_head = 0, s_rx_w_tail = 0;

static int fifo_push_word(const arinc_word_t *w)
{
    uint16_t n = (s_rx_w_head + 1U) % RX_WORD_FIFO_LEN;
    if (n == s_rx_w_tail) return -1; /* full */
    s_rx_words[s_rx_w_head] = *w;
    s_rx_w_head = n; return 0;
}
static int fifo_pop_word(arinc_word_t *w)
{
    if (s_rx_w_tail == s_rx_w_head) return -1;
    *w = s_rx_words[s_rx_w_tail];
    s_rx_w_tail = (s_rx_w_tail + 1U) % RX_WORD_FIFO_LEN;
    return 0;
}

/* -------------------- ARINC helpers -------------------- */
static uint8_t arinc_parity_odd(uint32_t word)
{
    /* bit 31 is parity; odd parity across bits [0..30] */
    uint32_t data = word & 0x7FFFFFFFUL;
```

```c
    uint32_t p = __builtin_popcount(data);
    uint32_t parity_bit = (word >> 31) & 1U;
    return ((p + parity_bit) & 1U) ? 1U : 0U; /* 1 => odd parity OK */
}

static void arinc_unpack(uint32_t raw, arinc_word_t *w)
{
    w->label     = (uint8_t)(raw & 0xFFU);
    w->sdi       = (uint8_t)((raw >> 8) & 0x3U);
    w->data      = (raw >> 9) & 0x7FFFFU;
    w->ssm       = (uint8_t)((raw >> 29) & 0x3U);
    w->parity_ok = arinc_parity_odd(raw);
}

/* -------------------- Tasks -------------------- */
static void TASK_ArincRxSvc(void)
{
    uint8_t buf[128];
    size_t n = ARINC_Rx_Drain(buf, sizeof(buf));
    /* Example framing: 4 bytes per ARINC word, little-endian raw32. Adjust i
f ADK-8582 adds headers. */
    for (size_t i = 0; i + 3 < n; i += 4) {
        uint32_t raw = (uint32_t)buf[i] |
                       ((uint32_t)buf[i+1] << 8) |
                       ((uint32_t)buf[i+2] << 16) |
                       ((uint32_t)buf[i+3] << 24);
        arinc_word_t w; arinc_unpack(raw, &w);
        (void)fifo_push_word(&w); /* on overflow, drop; HEALTH_MON reports */
    }
}

typedef struct {
    float indicated_airspeed_kt;
    float baro_altitude_ft;
    uint32_t last_update_us;
} avionics_state_t;
static avionics_state_t g_state;

static float arinc_bnr_to_float(uint32_t data_bits, float scale, float offset
)
{ return offset + scale * (int32_t)data_bits; }

static void TASK_ArincParse(void)
{
    arinc_word_t w; int processed = 0; uint32_t now = BM_TimeUs();
    while (fifo_pop_word(&w) == 0) {
        if (!w.parity_ok) continue;
        switch (w.label) {
            case 0xCB: g_state.indicated_airspeed_kt = arinc_bnr_to_float(w.d
```

```
ata, 0.01f, 0.0f); g_state.last_update_us = now; break; /* demo */
            case 0xCA: g_state.baro_altitude_ft      = arinc_bnr_to_float(w.d
ata, 1.0f,   0.0f); g_state.last_update_us = now; break; /* demo */
            default: break;
        }
        if (++processed > 16) break; /* bound per-activation work */
    }
}

static void TASK_HealthMon(void)
{
    static uint32_t last_ticks;
    uint32_t now_ticks = BM_Ticks();
    if ((now_ticks - last_ticks) > 120) {
        PRINTF("HEALTH: tick slip %lu ms\r\n", now_ticks - last_ticks);
    }
    last_ticks = now_ticks;

    uint16_t depth = (s_rx_w_head >= s_rx_w_tail)
                   ? (s_rx_w_head - s_rx_w_tail)
                   : (RX_WORD_FIFO_LEN - s_rx_w_tail + s_rx_w_head);
    if (depth > (RX_WORD_FIFO_LEN * 3 / 4)) {
        PRINTF("HEALTH: RX FIFO high watermark=%u\r\n", depth);
    }
    /* Feed watchdog here if configured; also toggle a heartbeat GPIO. */
}

static void TASK_MaintLog(void)
{
    PRINTF("LOG: IAS=%.1f kt ALT=%.0f ft @%lu us\r\n",
           (double)g_state.indicated_airspeed_kt,
           (double)g_state.baro_altitude_ft,
           (unsigned long)g_state.last_update_us);
}

/* -------------------- Task registry & boot -------------------- */
static bm_task_t g_tasks[] = {
    { "ARINC_RX_SVC",  TASK_ArincRxSvc,  10,  200,  500, 0,0,0,0,0 },  /* 10
ms, highest priority */
    { "HEALTH_MON",    TASK_HealthMon,  100,  180, 2000, 0,0,0,0,0 },  /* hea
lth above parser by policy */
    { "ARINC_PARSE",   TASK_ArincParse,  20,  150, 1000, 0,0,0,0,0 },  /* 20
ms, medium */
    { "MAINT_LOG",     TASK_MaintLog,   500,   50, 3000, 0,0,0,0,0 },  /* 500
ms, low */
};

int main(void)
{
```

```
    BM_Sched_Init(1000);          /* 1 kHz tick */
    ARINC_UART_Init(115200U);     /* set to actual ADK-8582 baud */
    BM_Sched_Register(g_tasks, sizeof(g_tasks)/sizeof(g_tasks[0]));
    BM_Sched_Run();               /* never returns */
    while (1) { }
}
```

---

8) Generic (non-avionics) example to teach priorities

Design three tasks:

- **SENSOR** (5 ms, high priority): sample ADC and compute a one-step filter.
- **CONTROL** (10 ms, medium priority): compute actuator command using last sensor value.
- **LOG** (100 ms, low priority): print status.

Inject a temporary 2 ms delay into CONTROL and observe that SENSOR still meets 5 ms releases, while LOG jitters (acceptable). Swap priorities to demonstrate why RMS-consistent assignments matter.

---

9) Hands-on exercises (SDK-based) with acceptance criteria

Exercise A — Build the cyclic executive and measure jitter

Objective. Compile `bm_sched.*` and toggle a GPIO at each task start/end to measure execution time and jitter with a logic analyzer.

Steps. 1. Create a new MCUXpresso project; add `bm_sched.c/h`, `arinc_uart.c/h`, and `main.c`. 2. Configure a GPIO as an output (see `driver_examples/gpio/`); in each task, set it high at entry and low at exit. 3. Capture traces; compute jitter as standard deviation of start offsets from ideal releases.

Acceptance. `ARINC_RX_SVC` jitter < 50 µs; other tasks within design limits.

Exercise B — Integrate UART RX ISR with ring buffer

Objective. Wire LPUART to ADK-8582 and confirm no overrun at nominal baud.

Steps. 1. Choose an LPUART instance not used by the console; adjust IOMUXC pin mux accordingly. 2. Use `ARINC_Rx_Drain` in `ARINC_RX_SVC` to drain and reassemble 4-byte words; print words per second in `MAINT_LOG`. 3. Increase baud or inject bursts; verify ISR remains short and ring buffer sizing is adequate.

Acceptance. No `kLPUART_RxOverrunFlag`; RX FIFO < 75% depth during worst-case bursts.

Exercise C — Deadline monitoring and BIT (Built-In Test)

Objective. Add per-task deadline checks and raise a BIT event on slips >10% of period.

Steps. 1. After each task run, compute: lateness = `(finish_us - start_us) - period_us`. 2. If lateness > 0.1×period_us, increment a `deadline_miss` counter and notify `HEALTH_MON`. 3. On repeated slips, temporarily disable `MAINT_LOG` to shed load.

Acceptance. Induced parser delays trigger controlled alarms and predictable degradation.

Exercise D — Priority inversion demo and mitigation (bare metal)

Objective. Demonstrate a low-priority task holding a shared peripheral delaying a high-priority task when a medium task is CPU-busy; fix via time partitioning.

Steps. 1. LOW task holds a pseudo-bus for 2 ms; HIGH wants it every 1 ms; MEDIUM is CPU heavy at 2 ms. 2. Observe HIGH misses when LOW collides with MEDIUM. 3. Fix: restrict bus usage to a reserved time window (e.g., the 0.5–1.0 ms slot of each 5 ms frame) or shorten the critical section.

Acceptance. HIGH meets its 1 ms deadline post-fix under the same load.

---

10) Best practices for Airbus-grade robustness

1. **No dynamic allocation after init.** All buffers and registries are `static`.
2. **ISR discipline.** ISRs copy bytes and set flags only; avoid parsing/printing inside ISRs.
3. **Bounded work per activation.** Cap processed items (e.g., ≤16 ARINC words per 20 ms).
4. **NVIC priorities.** Assign LPUART RX higher urgency than PIT; document all IRQ priorities in one place.
5. **Measure everything.** Use DWT cycle counter and GPIO toggles to quantify WCET and jitter.
6. **Time partition shared resources.** Prefer fixed windows over ad-hoc locks in cooperative executives.
7. **Input validation.** Check ARINC parity, SSM (Sign/Status Matrix), SDI (Source/Destination Identifier), label plausibility, and rate limits.
8. **Fail predictable.** On sustained overload, shed non-critical load (pause logging) before critical tasks suffer.

9. **Coding standard.** Follow MISRA-C; avoid recursion and unbounded loops.
10. **Configuration control.** Keep a single "Scheduling Table" with task, period, deadline, WCET, priority, and NVIC interactions.

---

11) Scheduling Table template (to maintain under configuration control)

```
Task Name     Period  Deadline  Priority  WCET(us)  Avg Exec(us)  Jitter(us)
Notes
-------------------------------------------------------------------------
--------
ARINC_RX_SVC  10 ms   10 ms     200       300       120           <50
Drains UART ring
HEALTH_MON    100 ms  100 ms    180       500       80            <100
WDT feed, audits
ARINC_PARSE   20 ms   20 ms     150       800       300           <150
Label routing
MAINT_LOG     500 ms  500 ms    50        2000      600           N/A
Background only
```

---

12) Build & integration notes (MCUXpresso SDK)

- Start from an SDK hello-world or PIT example and add these files.
- Ensure the chosen LPUART instance is **not** the debug console. If necessary, move the console to another LPUART or use SWO.
- Configure IOMUXC for the selected UART pins (TX from ADK-8582 to EVKB RX pin). Keep hardware flow control off unless required by the adapter.
- Keep PIT IRQ priority numerically **larger** (less urgent) than LPUART RX.
- For lab visibility, enable `PRINTF` via the SDK debug console and throttle prints (printing is slow).

---

13) Optional advanced: preemption via PendSV (for teaching only)

If you want true task-level preemption without an RTOS, use the Cortex-M **PendSV** exception as a software context switch. SysTick/PIT selects the next runnable task and sets **PendSV** to perform save/restore of registers and stacks. This is essentially building a micro-kernel—excellent as a teaching module, but in production prefer a well-tested RTOS if you need preemption between tasks.

---

14) Troubleshooting

- **UART overruns:** Lower NVIC priority number for LPUART (increase urgency), increase ring buffer, reduce ISR work, or reduce baud.
- **Missed periodic releases:** Check PIT clock source and interrupt enable; verify your main loop uses `__WFI()` and isn't spinning needlessly.
- **Jitter too high:** Reduce printing, bound per-activation work, pre-compute lookups, and consider moving heavy parsing to slower periods.
- **Pin mux issues:** Confirm IOMUXC settings and that pins aren't shared with the debug console.

---

15) What to measure and record for review

- **WCET** for each task (via DWT cycles → microseconds),
- **Jitter** of task start times relative to ideal release instants,
- **RX ring high-water mark** and **word FIFO depth** under maximum expected burst,
- **IRQ latencies** (time from byte arrival to ISR entry),
- **Deadline miss counters** and recovery behavior.

Maintaining these artifacts alongside the scheduling table supports your safety case and regression testing.
