01 ESL – Engineering Software Lab

Embedded software consultancy · optimizing and migrating embedded projects

ESL
Engineering Software Lab

- Embedded software consultancy specializing in optimizing and migrating embedded projects.
- Parasoft and Perforce partner in Israel for over 20 years.
- This proof was designed and built by ESL to show a migration path with evidence, not just a plan.



Daniel Liezrowice
CTO, ESL

Has used ThreadX for over 25 years.

LinkedIn profile

02 ThreadX 5.6 to 6.5, Cortex-M4 to M85

One application, three migration stages, executable evidence

The same Clang-built application runs on the real ThreadX 5.6 kernel, then on current MIT ThreadX 6.5, then is built for Cortex-M85.

Stage	Kernel / core	Evidence achieved
1 TX5-M4	ThreadX G5.6.5.2 prebuilt archive / Cortex-M4	Built, executed: 8/8 runs
2 TX6-M4	ThreadX 6.5.2.202603 / Cortex-M4	Built, executed: 8/8 runs
3 TX6-M85	ThreadX 6.5.2.202603 / Cortex-M85	Compiled and linked only
+ TX6-M55	ThreadX 6.5.2.202603 / Cortex-M55	Supplement, executed: 8/8 runs

Use left / to, touch arrows, or swipe. Home returns to the start.

Navigate in the interactive edition with arrows, swipe and Home. This PDF contains every slide in sequence.

03 Change one thing per stage

So every difference has exactly one cause

Step	Kernel	Core / ABI	Application source
Stage 1 to 2	TX5.6 to TX6.5	Cortex-M4 unchanged; soft-float to hard-float	Unchanged
Stage 2 to 3	TX6.5 unchanged	Cortex-M4 to Cortex-M85	Unchanged
Supplement	TX6.5 unchanged	Cortex-M55: newest core this QEMU runs	Unchanged

- Stage 1 to 2 isolates the RTOS migration on the core the TX5 kernel was built for.
- Stage 2 to 3 isolates the CPU port change on an unchanged kernel release.
- M55 shares the Armv8.1-M port family with M85, so it is the closest runnable proxy, not a substitute for M85 execution.

The float ABI change in stage 1 to 2 is forced by the TX5 archive, not chosen. It is called out as its own diff later.

04 The challenges of this project

What we did not have, and how each gap was closed

Challenge	Why it matters	How ESL handled it
No ThreadX 5 source code: only a prebuilt binary archive	The port contract, ABI and configuration cannot be read from source or rebuilt	Recovered them from build attributes, symbols and disassembly of the 192 archive objects
No real target hardware: no M4 board, no M85 silicon	No physical GPIO, clocks, interrupts or cycle-accurate timing	QEMU MPS2 AN386 (M4) and MPS3 AN547 (M55), virtual LEDs and buttons, deterministic icount time
No Cortex-M85 emulator available	Stage 3 cannot execute anywhere in this setup	Ran the closest Armv8.1-M core (M55); M85 verified by build attributes and a full link
TX5 binary licence restrictions and a fixed soft-float ABI	The kernel cannot be redistributed, and every object must match its ABI	Linked from an external checkout only; Stage 1 built soft-float with packed enums; TCB size asserted
Compiler change from GCC to Clang	The TX5 archive was built by GCC; Clang has different defaults and assembler behaviour	Matched the archive ABI explicitly; fixed a flag-dependent startup loop found by QEMU trace

Each gap that could not be closed without hardware is listed again on the acceptance-gate slide, so nothing is silently assumed.

05 ThreadX 5 and 6: what actually changes

Same service model; different delivery, port contract and licence

Concern	ThreadX 5.6 (WICED archive)	ThreadX 6.5 (Eclipse)
Delivery	Headers + prebuilt GCC archive, no kernel source	Full C source + CPU/toolchain ports
Licence	Commercial; SDK copy restricted to WICED use	MIT
Ports in this delivery	One Cortex-M3/M4 port	M4, M55, M85 and many more
Float ABI	Soft-float; r4-r11 context only	Hard-float; FPU context in the port
Thread start / handlers	SVC; __tx_PendSVHandler, __tx_SVCallHandler	PendSV; PendSV_Handler
Configuration	Fixed by archive: errno extension, no error checking	Rebuilt from tx_user.h

Rows are findings from this specific SDK archive and its headers. Other licensed TX5 distributions may have been built with different options.

06 Where the ThreadX 5 kernel comes from

TX5 was never open source; the WICED SDK ships it as binaries

- The public WICED SDK 3.1.0 contains ThreadX 5.6 headers and ThreadX.ARM_CM4.release.a.
- Its README restricts those objects to WICED use, so nothing from the SDK is in this repository.
- The build links the archive from TX5_ROOT, a separate checkout; configuration fails if TX5_ROOT is inside the repository.
- CI clones the same SDK commit into the runner's temp directory for every run.

Reproducible external dependency, identified by commit and hash

```
git clone --filter=blob:none --no-checkout \
https://github.com/wwwlllllll/bcm-wiced-sdk.git $TX5_ROOT
git -C $TX5_ROOT sparse-checkout set --no-cone '/WICED/RTOS/ThreadX/'
git -C $TX5_ROOT checkout 0886ec932a063e12db768cd09428e8a8ab555bce
# ThreadX.ARM_CM4.release.a sha256:
# 71288b46e58b2c398118674b53feb708994e18740cf7e5d68bc8d070f72fac1e
```

07 Reading a binary-only kernel

No source, so inspect build attributes, symbols and machine code

Actual inspection of the WICED ThreadX.ARM_CM4.release.a members

```
$ llvm-readelf -A tx_thread_create.o
CPU_name: Cortex-M4        CPU_arch: ARM v7E-M
ABI_enum_size: Packed      (no ABI_VFP_args, no FP_arch)
$ llvm-nm tx_thread_schedule.o
T __tx_PendSVHandler   T __tx_SVCallHandler
U tx_low_power_enter   U _tx_thread_system_stack_ptr
$ llvm-strings tx_thread_initialize.o
ThreadX Cortex-M3/GNU Version G5.6.5.2
```

- No VFP tag: the archive is soft-float, so every object in the image must be soft-float.
- Packed enums: GCC -fshort-enums; Clang must match it.
- The CM4 archive is the Cortex-M3 port: it never saves FPU registers.
- Unresolved WICED hooks show exactly what the board layer must provide.

08 The exact versions, not moving targets

Every kernel input is pinned and identifiable

Input	Identity
TX5 kernel	ThreadX G5.6.5.2, WICED SDK 3.1.0 commit 0886ec932a06
TX5 archive sha256	71288b46e58b2c398118674b53feb708994e18740cf7e5d68bc8d070f72fac1e
TX6 kernel	Eclipse ThreadX v6.5.2.202603_rel, latest observed on 2026-10-11
TX6 commit	93387b0a60382a29579bbad5045c68328e3751df
Compiler	clang version 22.1.3 (https://github.com/msys2/MINGW-packages b2ba9d193810f115e9af26819b666d1fa929a940)
Emulator	QEMU emulator version 11.1.0 (v11.1.0-12130-ge470268ff4)

The originally linked v6.1.12_rel tag carries Azure RTOS hardware-linked licence text in its own tree; it is not substituted for the MIT release.

09 Cortex-M4, M55 and M85

Keep CPU architecture distinct from a particular microcontroller

Property	Cortex-M4	Cortex-M55	Cortex-M85
Architecture	Armv7E-M	Armv8.1-M	Armv8.1-M
DSP / vectors	Scalar DSP extensions	Helium / MVE	Helium / MVE
Floating point	Optional single precision	Configuration-dependent FP/MVE	Configuration-dependent FP/MVE
Security model	No Armv8-M TrustZone	Armv8-M TrustZone	TrustZone; PACBTI capability
This proof	TX5 and TX6 run on MPS2 AN386	TX6 run on MPS3 AN547	TX6 compile/link only

Core IP features and SoC implementations vary. A new core does not bring the old board's GPIO, clocks, memory addresses, DMA or wiring with it.

10 The most modern runnable QEMU Cortex-M here

Probe the installed executable instead of relying on development docs

Actual local inventory, saved in evidence/qemu-*.txt

```
qemu-system-arm --version
qemu-system-arm -machine help
qemu-system-arm -cpu help
# installed: 11.1.0
# present: mps2-an386 / cortex-m4, mps3-an547 / cortex-m55
# absent:  mps3-an555 / cortex-m85
```

M55 is the newest Cortex-M CPU exposed by this installed QEMU build. QEMU development documentation mentions AN555, but it is not in the tested binary.

An M85 image is never run under an M55 CPU and labelled M85. The M85 ELF remains a separate, explicitly unexecuted build artifact.

11 Licensing is an engineering input

Public on GitHub does not mean free to redistribute

- The WICED SDK README limits its ThreadX / NetX objects to use with WICED.
- The TX5 archive and headers are linked from an external checkout and never committed or published.
- The application, board layer, adapter and tests are independently written; original work is MIT.
- ThreadX 6.5 is pinned as an MIT submodule with its upstream notices.
- PDF typography embeds Roboto under OFL 1.1, with the notice shipped beside the PDF.

This is an engineering demonstration, not a legal opinion on WICED licence scope. Product use of a TX5 kernel needs its own licence.

ESL
Engineering Software Lab

12 The source trees

The TX5 kernel stays outside; the repository holds only original work

Source structure: external TX5 kernel vs public repository

TX5_ROOT: external WICED SDK checkout

```
bcm-wiced-sdk/  (sparse, commit 0886ec9)
  WICED/RTOS/ThreadX/
    ver5.6/tx_api.h, tx_user.h ...
    ver5.6/Cortex_M3_M4/GCC/tx_port.h
    ThreadX.ARM_CM4.release.a
      192 objects, soft-float
  (no other SDK component is used)

never copied into the repository
```

TX5-upgrade (MIT original work)

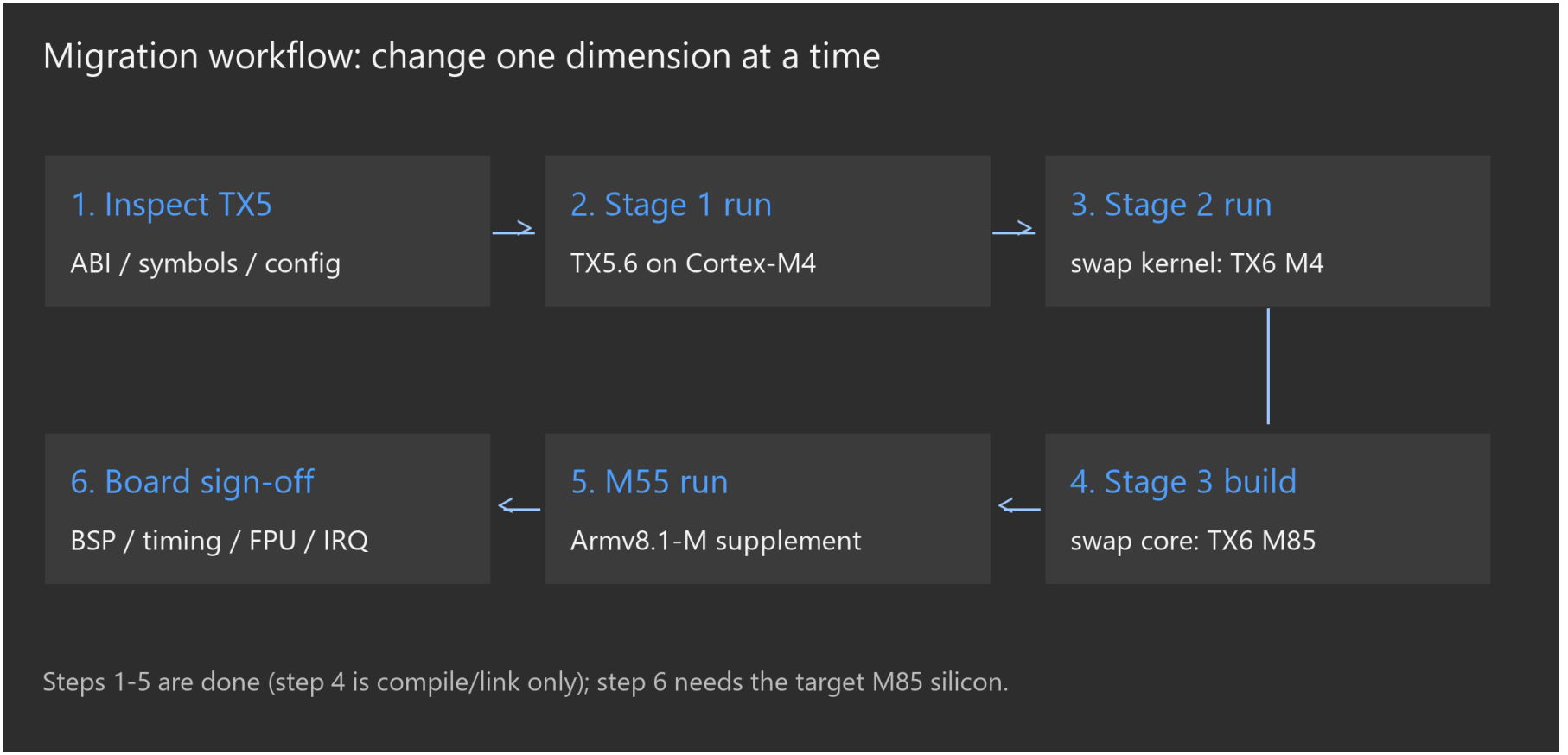
```
TX5-upgrade/
  src/led_demo.c + rtos_selftest.c
  src/startup.S + qemu_board.c
  src/tx5_wiced_port.c  [TX5 only]
  cmake/tx5-wiced-adapter.cmake
  config/tx6/tx_user.h  [TX6 only]
  third_party/threadx/ [pinned MIT 6.5]
  boards/qemu.ld + tests/ + evidence/
  scripts/ + presentation/ + LICENSE
```

Left: what TX5_ROOT must contain. Right: the public repository.

TX5-only and TX6-only files are small and named for their stage: one adapter, one board hook, one tx_user.h.

13 Migration workflow

Change one dimension at a time

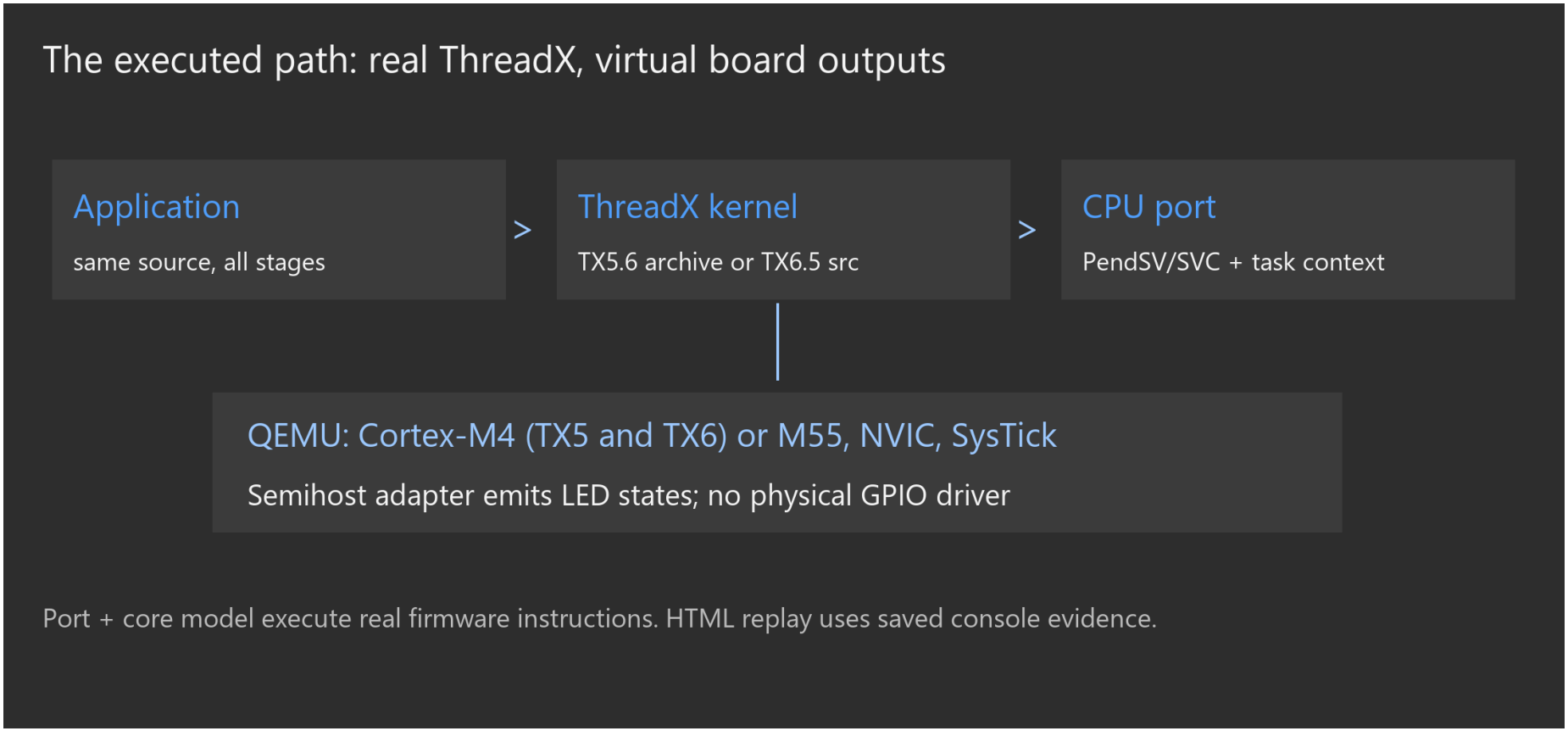


Kernel first on the original core, then the core on the new kernel.

Stages 1 and 2 give a like-for-like Cortex-M4 run on each kernel. Stage 3 then needs only the CPU port change.

14 What really executes

No mock RTOS, no fake emulator success



Clang-built firmware runs the real TX5.6 or TX6.5 scheduler under QEMU.

The board adapter virtualizes two LEDs and deterministic active-low buttons. It does not emulate or access a physical GPIO controller.

15 Diff 1: how each kernel enters the build

Imported binary archive vs rebuilt source

Actual build integration; the application target is identical

```
# Stage 1: cmake/tx5-wiced-adapter.cmake
- add_library(threadx STATIC IMPORTED GLOBAL)
- IMPORTED_LOCATION  ${TX5_ROOT}/.../ThreadX.ARM_CM4.release.a
- INTERFACE_COMPILE_DEFINITIONS  TX_INCLUDE_USER_DEFINE_FILE
- include dirs: ver5.6/ and ver5.6/Cortex_M3_M4/GCC/
# Stages 2-3: CMakeLists.txt
+ set(THREADX_ARCH ${TX_CORE})       # cortex_m4 / m55 / m85
+ set(TX_USER_FILE config/tx6/tx_user.h)
+ add_subdirectory(third_party/threadx)
# all stages
target_link_libraries(threadx_gpio_demo PRIVATE threadx)
```

With source, the kernel is recompiled for the application's ABI and configuration. With a binary, the application must adopt the archive's ABI and configuration instead.

16 Diff 2: the float ABI is dictated by the archive

Hard-float objects cannot link against this TX5 kernel

Toolchain change plus disassembly of __tx_ts_handler in the archive

```
# cmake/clang-arm-none-eabi.cmake
- TX5: -mcpu=cortex-m4 -mthumb -mfloat-abi=soft -fshort-enums
+ TX6: -mcpu=cortex-m4 -mthumb -mfloat-abi=hard
# TX5.6 PendSV: integer context only
mrs   r12, psp
stmdb r12!, {r4-r11}      @ no s16-s31, no lazy FP state
...
mvn   lr, #2              @ EXC_RETURN 0xFFFFFFFD
```

- A hard-float task on this TX5 port would break: it saves no s16-s31 and always returns with a non-FP EXC_RETURN.
- Stage 2 is therefore also the first point where FPU code becomes safe on M4.
- The image attributes confirm it: TX5 has no FP_arch tag; TX6 M4 has VFPv4-D16 and AAPCS VFP.

17 Diff 3: vectors and the first context switch

TX5.6 starts threads through SVC, TX6 does not

Handler names are per-stage CMake defaults, validated as plain symbols

```
# TX5.6 _tx_thread_schedule (from the archive)
str   r1, [r0, #0x10]      @ SysTick CTRL |= ENABLE
cpsie i
svc   #0                   @ first thread starts in SVCall
# src/startup.S vector table
- .word DEMO_SVC_HANDLER     @ TX5: __tx_SVCallHandler
- .word DEMO_PENDSV_HANDLER  @ TX5: __tx_PendSVHandler
+ .word DEMO_SVC_HANDLER     @ TX6: fault_handler (unused)
+ .word DEMO_PENDSV_HANDLER  @ TX6: PendSV_Handler
```

Leaving SVC at the TX6 default fault handler makes TX5 hang before the first thread runs. The verifier treats that as a timeout failure.

18 Diff 4: low-level init and the idle hook

Contracts recovered from disassembly, implemented in MIT code

src/tx_initialize_low_level.c and src/tx5_wiced_port.c

```
void _tx_initialize_low_level(void) {
_tx_initialize_unused_memory = &__first_free_ram;
#ifdef DEMO_TX5
-   _tx_thread_system_stack_ptr = (void *)demo_vectors[0];
#endif
/* VTOR, SHPR2/3 priorities, 100 Hz SysTick: shared */
}
- unsigned long platform_power_down_hook(unsigned long ticks) {
-     __asm volatile("wfi");
-     return 0UL;       /* ticks slept; SysTick kept running */
- }
```

- The TX5 port objects reference _tx_thread_system_stack_ptr; the TX6 Cortex-M4 port sources do not use it.
- The WICED idle loop calls tx_low_power_enter, which calls platform_power_down_hook(ticks to next timer).
- A non-zero return is added to the kernel clock (tickless mode). Returning 0 keeps normal ticking.

19 Diff 5: a 4-byte configuration bug

Found by the metrics, then made impossible to build

Actual first Stage 1 run, root cause and fix

```
[metric] sizeof_TX_THREAD=176       <- application's view
_tx_thread_create:  movs r2, #0xb4   <- archive clears 180 B
bl   memset
[metric] led_stack_high_water_bytes=1024   <- impossible
# cause: -Iinclude found the TX6 tx_user.h before WICED's,
#       dropping "#define TX_THREAD_USER_EXTENSION int errno;"
+ move TX6 config to config/tx6/tx_user.h
+ _Static_assert(sizeof(TX_THREAD) == 180U, ...);
[metric] sizeof_TX_THREAD=180   led_stack_high_water_bytes=196
```

The kernel cleared 4 bytes past each 176-byte thread object. led_thread_stack follows led_thread in RAM, so its first fill word was zeroed and the stack scan reported a full stack. The functional test still passed: only the measurement exposed the corruption.

With a binary kernel, the header configuration is part of the ABI. It must be the configuration the archive was built with, and it should be asserted.

20 Diff 6: API and configuration details

Small differences that break builds or change behaviour

Item	TX5.6 archive	TX6.5 build here
_tx_version_id	extern const CHAR[]	extern CHAR[]
Thread extension	int errno; (WICED tx_user.h)	None
Error checking	Disabled in release archive	Enabled (txe_ wrappers)
Preemption-threshold	Disabled	Enabled
Timer expiry	Processed in the SysTick ISR	System timer thread (default)
Stack checking	Off; stacks still filled with 0xEF	On

- The application dropped its own _tx_version_id declaration and uses the port header's.
- Stack high-water is measured by scanning the 0xEF fill, which works on both kernels.
- Thread resumption is checked with an application flag, not scheduler internals.

21 Diff 7: milliseconds versus RTOS ticks

Keep units visible; avoid a silent tenfold delay error

Timing integration, plus a measured tick invariant

```
#define TX_TIMER_TICKS_PER_SECOND 100   // tick rate used by both stages
#define LED_UPDATE_TICKS 25U
tx_thread_sleep(LED_UPDATE_TICKS);      // 250 ms virtual time
SysTick reload = DEMO_CLOCK_HZ / 100 - 1
SysTick_Handler -> _tx_timer_interrupt  // same symbol in TX5 and TX6
```

The tested board clock constants are 25 MHz for MPS2-M4 and 32 MHz for MPS3-M55. The M85 build reuses the demonstration map and clock; a real M85 SoC must replace them.

QEMU icount is synthetic: 128 ns per emulated instruction. The tick invariant is a scheduler test, not a physical time or jitter guarantee.

22 Diff 8: preserve the held-button latch

A subtle behavioural detail, checked at release as well as press

Application behaviour, identical on every stage

```
- toggle the same boolean even while the button is held
+ if (!button0) toggle0 = !toggle0;
+ output0 = button0 || toggle0;
+ qemu_led_set(0, output0);
# cycle 1: latch OFF / output OFF
# cycles 2-3: latch unchanged / held button forces ON
# cycle 4: button released / next toggle yields ON
```

The verifier deliberately rejects a cycle-4 OFF result.

23 Diff 9: startup assembly under Clang

A real encountered failure, with a trace-supported fix

Compared with the upstream TX6 example startup, not Broadcom source

```
- sub r2, r2, r1   // Clang emitted sub.w, without updating flags
- beq copy_done    // branches on stale flags
+ copy_loop:
+     cmp r1, r2   // explicit end-pointer comparison
+     bhs copy_done
+     ldrb r3, [r0], #1
+     strb r3, [r1], #1
+     b copy_loop
```

QEMU traced a bus fault at 0x00080000 during the endless startup copy. The replacement uses explicit comparisons, copies .data, clears .bss and enables CP10/CP11.

24 Diff 10: TX6 security mode and BSP scope

Scope down the first proof; make omitted features explicit

Demonstrator configuration, not a production M85 BSP

```
# config/tx6/tx_user.h
+ #define TX_SINGLE_MODE_SECURE
+ #define TX_TIMER_TICKS_PER_SECOND 100
+ #define TX_ENABLE_STACK_CHECKING
# boards/qemu.ld (all stages)
CODE 0x00000000 512 KiB, RAM 0x20000000 32 KiB
system stack 2048 B, 8-byte aligned; VTOR = demo_vectors
PendSV/SVC priority 0xff, SysTick 0x40
```

- Single-secure avoids exercising secure-stack paths without a non-secure image.
- Do not infer a TrustZone security policy from a successful boot.
- A real M85 SoC has its own memory map, clocks, IRQ routing, caches and MPU.

25 Diff 11: success is a checked contract

Fail on missing, wrong or partial output

tests/run_qemu.py + 15 verifier unit tests

```
+ require exactly one banner version, equal to the stage's
+     TX5: G5.6.5.2      TX6: 6.5.2.202603
+ verify the CPUID part number for the requested core
+ require queue / event / semaphore / mutex / pool self-test PASS
+ check all 10 button + LED rows and 9 intervals of 25 ticks
+ require explicit completion and QEMU exit status 0
+ reject timeout, wrong identity, wrong state or partial log
```

A TX6 image cannot pass as a TX5 run: the expected version is per stage and must match exactly. CMake also refuses a TX5 build on any core but M4, or with TX5_ROOT inside the repository.

26 The RTOS services really switch tasks

A higher-priority worker must preempt, suspend, resume and signal

Condensed from the independently written service self-test

```
// LED task priority 5; worker priority 4
worker: worker_progress = 1;
tx_queue_receive(&queue, &value, TX_WAIT_FOREVER);
worker_progress = 2;
tx_event_flags_set(&flags, 1, TX_OR);
LED:    require(worker_progress == 1);   // preempted, then suspended
tx_queue_send(&queue, &message, TX_NO_WAIT);
require(worker_progress == 2);   // resumed by the send
// observed on all three runtime images:
//   worker_thread_runs = 2; led_thread_runs = 12
```

- The progress flag is application state, so the check does not depend on TCB internals.
- The timer wakes the sleeping LED task nine times through each kernel's real timer path.
- Mutex and pool checks are basic operations, not contention or fragmentation stress.

27 Replay the verified LED run

Recorded Stage 1 QEMU evidence, not a browser-side emulator

Cycle	Tick	Button 0	Button 1	LED 0	LED 1
0	0	released	released	ON	ON
1	25	released	released	OFF	OFF
2	50	held	released	ON	ON
3	75	held	released	ON	OFF
4	100	released	released	ON	ON
5	125	released	held	OFF	ON
6	150	released	released	ON	OFF
7	175	released	released	OFF	ON
8	200	released	released	ON	OFF
9	225	released	released	OFF	ON

Two buttons are scripted active-low inputs. A held button forces its LED on; the paused toggle resumes after release. Step through cycles 0–9.

Embedded from evidence/tx5-m4-run.json. The generator checks that the TX6 M4 and M55 runs produced identical rows.

28 Console evidence: Stage 1, ThreadX 5.6

Cortex-M4, prebuilt WICED kernel

Actual QEMU console, not generated example output

```
[boot] ThreadX GPIO behavior demo; LED output is semihosted
[core] cortex-m4
[metric] cpuid=1091551808
Copyright (c) 1996-2013 Express Logic Inc. * ThreadX Cortex-M3/GNU Version G5.6.5.2 SN: 23451-108-0501 *
[metric] sizeof_TX_THREAD=180
[metric] led_stack_reserved=1024
[metric] worker_thread_runs=2
[selftest] queue/preemption/events/semaphore/mutex/byte-pool PASS
cycle=0 BTN0=released BTN1=released LED0=ON LED1=ON tick=0
cycle=1 BTN0=released BTN1=released LED0=OFF LED1=OFF tick=25
cycle=2 BTN0=PRESSED BTN1=released LED0=ON LED1=ON tick=50
cycle=3 BTN0=PRESSED BTN1=released LED0=ON LED1=OFF tick=75
cycle=4 BTN0=released BTN1=released LED0=ON LED1=ON tick=100
cycle=5 BTN0=released BTN1=PRESSED LED0=OFF LED1=ON tick=125
cycle=6 BTN0=released BTN1=released LED0=ON LED1=OFF tick=150
cycle=7 BTN0=released BTN1=released LED0=OFF LED1=ON tick=175
cycle=8 BTN0=released BTN1=released LED0=ON LED1=OFF tick=200
cycle=9 BTN0=released BTN1=released LED0=OFF LED1=ON tick=225
[metric] led_thread_runs=12
[metric] led_stack_high_water_bytes=196
[pass] bounded LED sequence completed
```

29 Console evidence: Stage 2, ThreadX 6.5

Same Cortex-M4, same application, new kernel

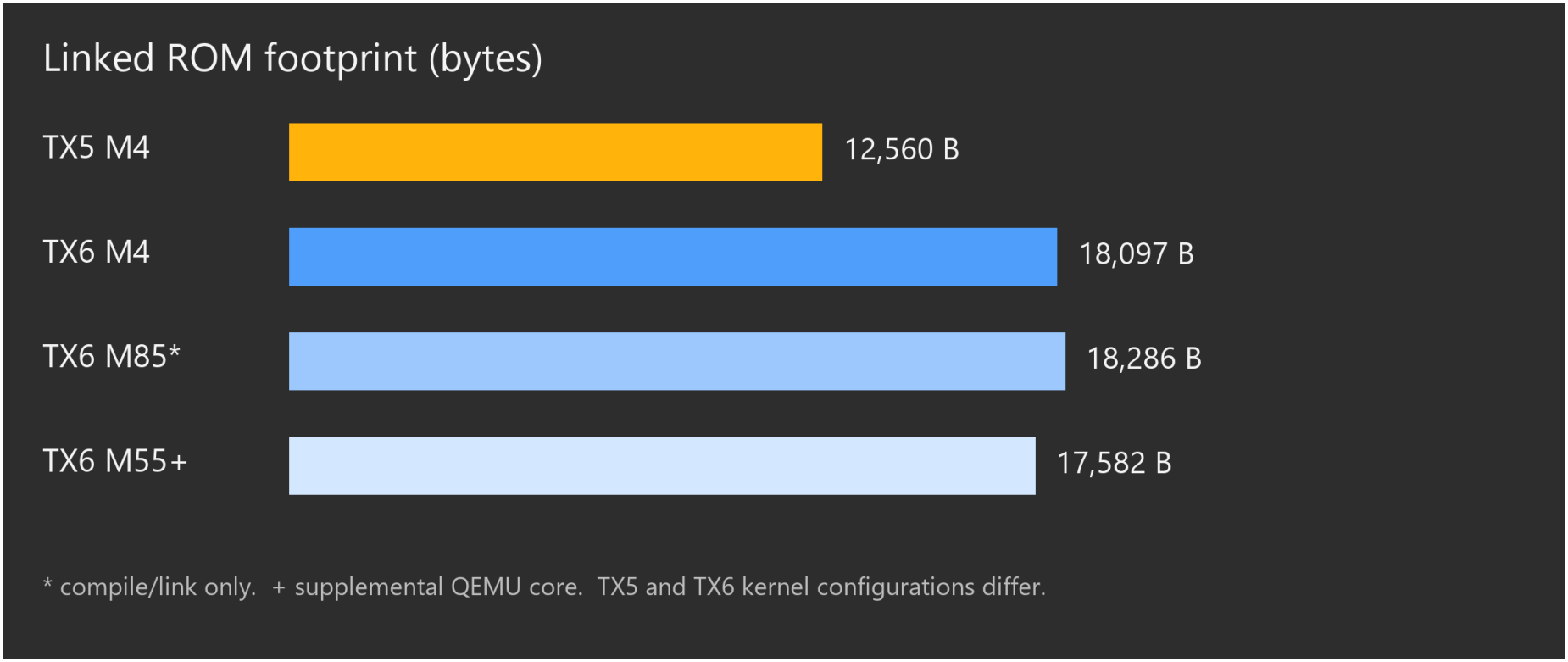
Actual QEMU console, not generated example output

```
[boot] ThreadX GPIO behavior demo; LED output is semihosted
[core] cortex-m4
[metric] cpuid=1091551808
(c) 2024 Microsoft Corp. (c) 2026-present Eclipse ThreadX contributors.  *  ThreadX Cortex-M4/GNU Version 6.5.2.202603 *
[metric] sizeof_TX_THREAD=180
[metric] led_stack_reserved=1024
[metric] worker_thread_runs=2
[selftest] queue/preemption/events/semaphore/mutex/byte-pool PASS
cycle=0 BTN0=released BTN1=released LED0=ON LED1=ON tick=0
cycle=1 BTN0=released BTN1=released LED0=OFF LED1=OFF tick=25
cycle=2 BTN0=PRESSED BTN1=released LED0=ON LED1=ON tick=50
cycle=3 BTN0=PRESSED BTN1=released LED0=ON LED1=OFF tick=75
cycle=4 BTN0=released BTN1=released LED0=ON LED1=ON tick=100
cycle=5 BTN0=released BTN1=PRESSED LED0=OFF LED1=ON tick=125
cycle=6 BTN0=released BTN1=released LED0=ON LED1=OFF tick=150
cycle=7 BTN0=released BTN1=released LED0=OFF LED1=ON tick=175
cycle=8 BTN0=released BTN1=released LED0=ON LED1=OFF tick=200
cycle=9 BTN0=released BTN1=released LED0=OFF LED1=ON tick=225
[metric] led_thread_runs=12
[metric] led_stack_high_water_bytes=224
[pass] bounded LED sequence completed
```

Only the banner and stack high-water differ from Stage 1. Every LED row and tick is identical.

30 Memory chart: linked ROM

Clang -O3; same application and service tests on every image



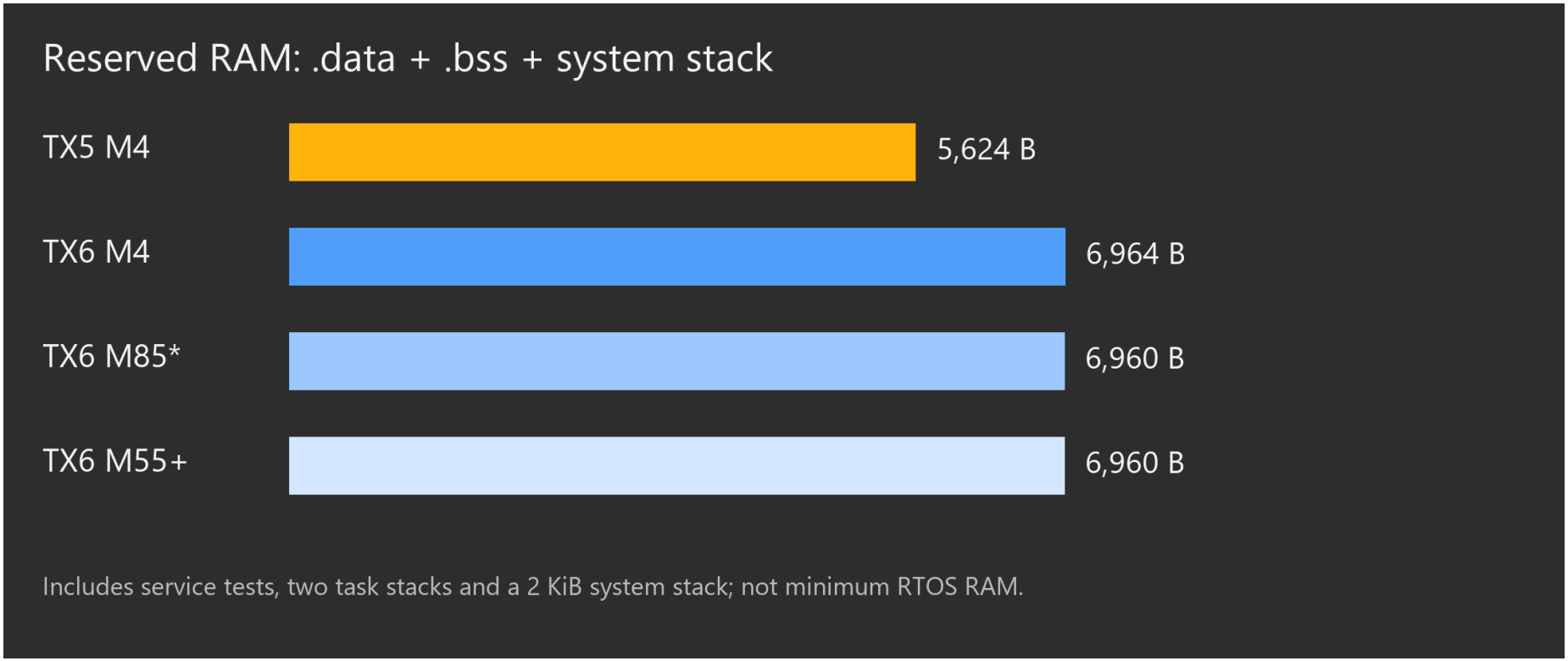
ROM = .vectors + .text + .ARM.exidx + .data load image.

TX5 M4: 12,560 B; TX6 M4: 18,097 B (+5,537 B); TX6 M85: 18,286 B; TX6 M55: 17,582 B.

The TX5toTX6 delta is not pure kernel growth: the TX6 build enables error checking, preemption-threshold and a system timer thread, which the WICED release archive leaves out.

31 Memory chart: reserved RAM

Include stacks and test infrastructure, not only global data

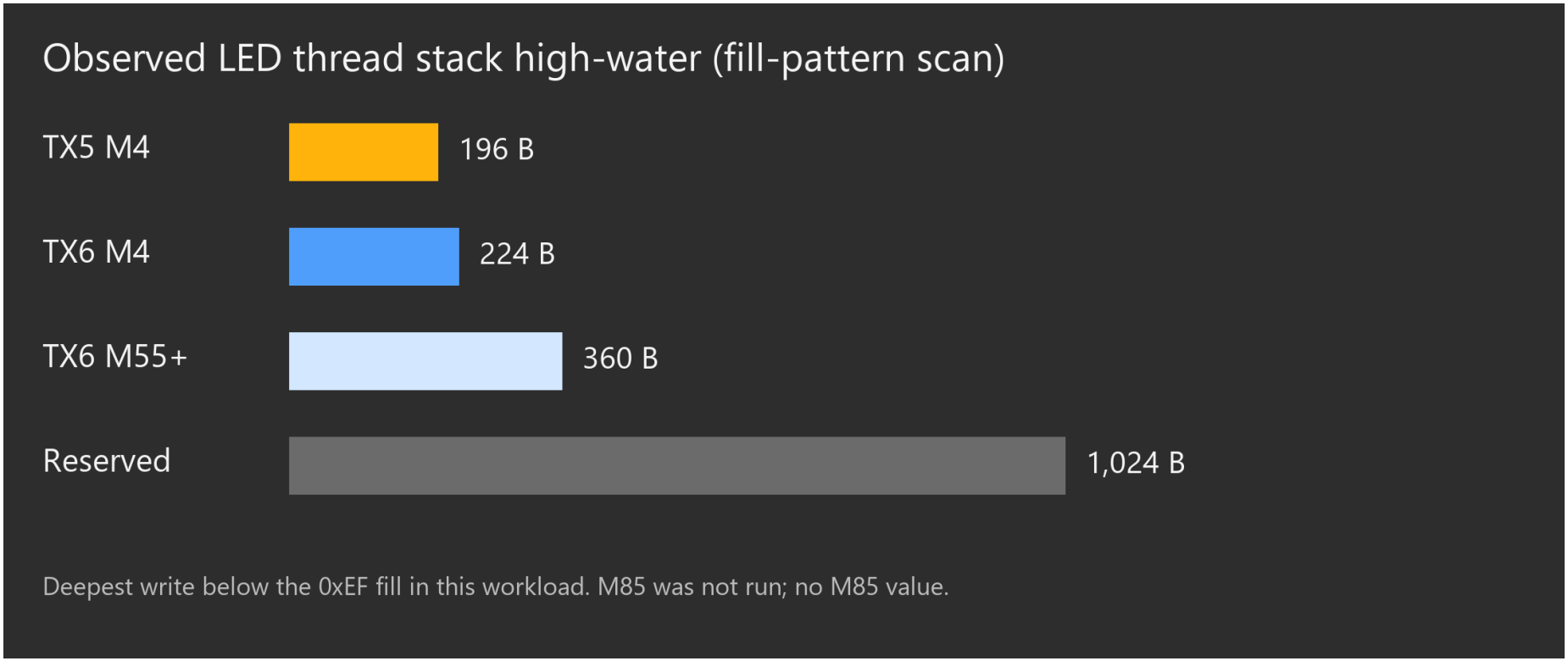


RAM = .data + .bss + 2 KiB system stack. Static task stacks are inside .bss.

TX5 M4: 5,624 B; TX6 M4: 6,964 B. TX6 adds a system timer thread and its stack; TX5 processes timers in the ISR. sizeof(TX_THREAD) happens to be 180 bytes on both kernels with these configurations.

32 Stack headroom is measured, not assumed

The same fill-pattern scan on both kernels



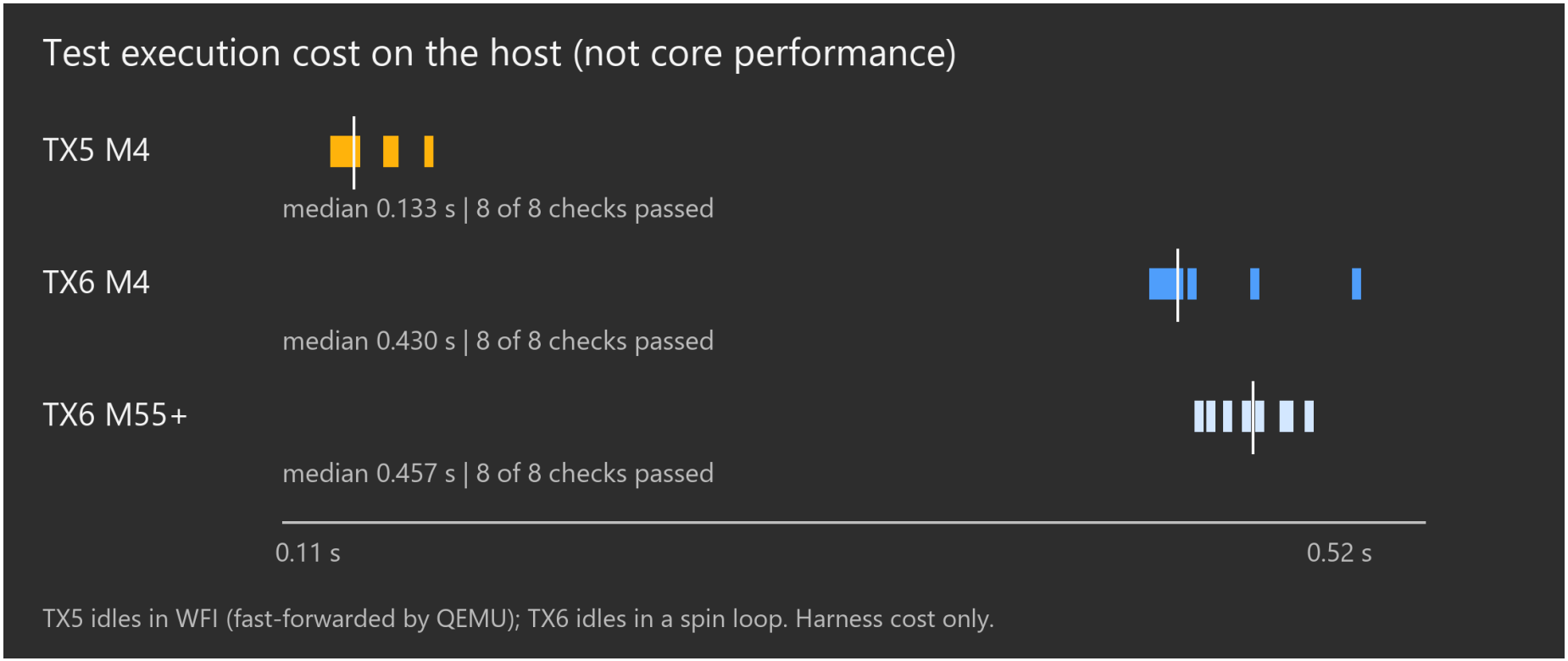
Deepest LED-thread stack write under the test workload; 1024 bytes reserved.

TX5 M4 196 B, TX6 M4 224 B, TX6 M55 360 B. Differences come from the ABI, compiler code generation and each port's stack frame; they are not attributed to one cause.

Production worst-case paths, nested interrupts, FPU/MVE context and M85 behaviour still require stress and hardware validation.

33 Performance chart: emulator test cost

Measure what was run, and say what the number means

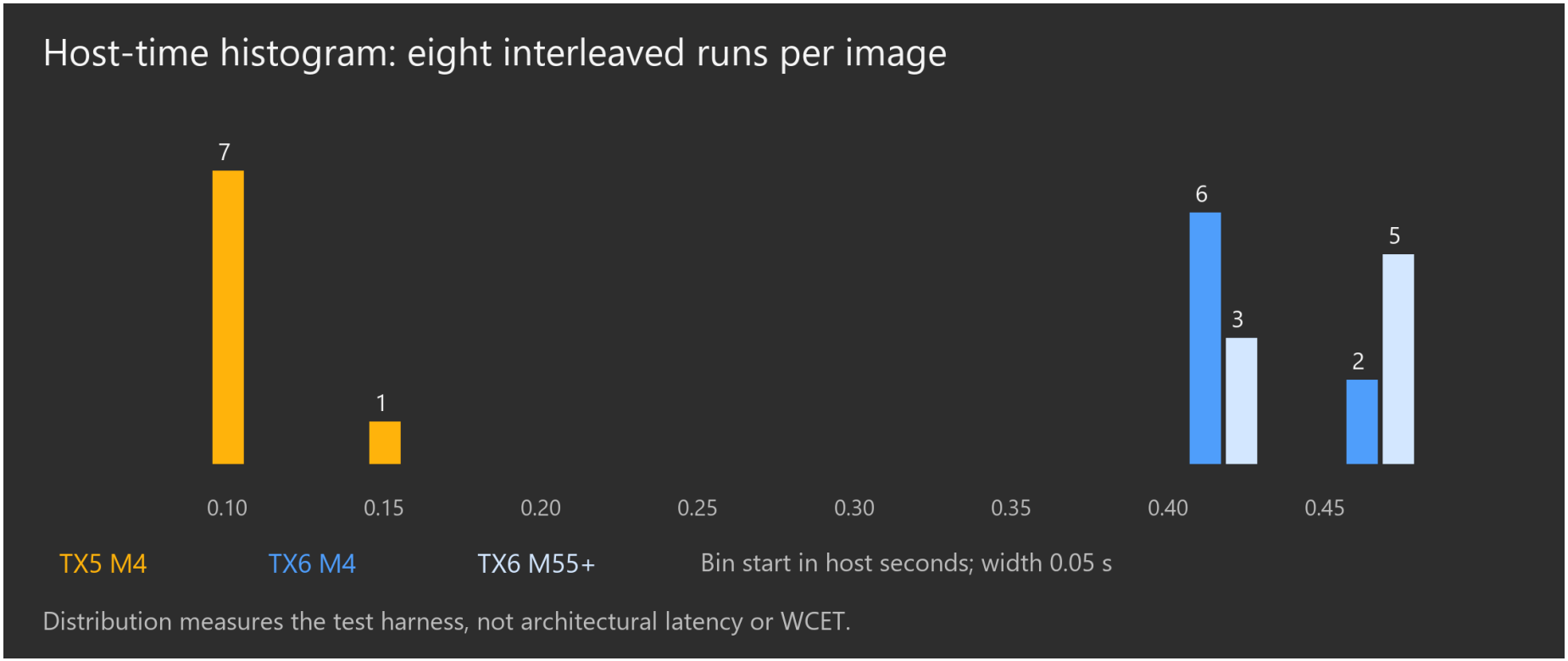


Eight interleaved QEMU runs per image on this Windows host, using deterministic icount.

TX5 looks faster because its idle hook executes WFI, which QEMU skips to the next timer, while TX6 idles in a spin loop that QEMU must emulate. This is idle strategy, not kernel speed.

34 Histogram: repeatability of the host harness

Samples are retained, not reduced to an unexplained mean

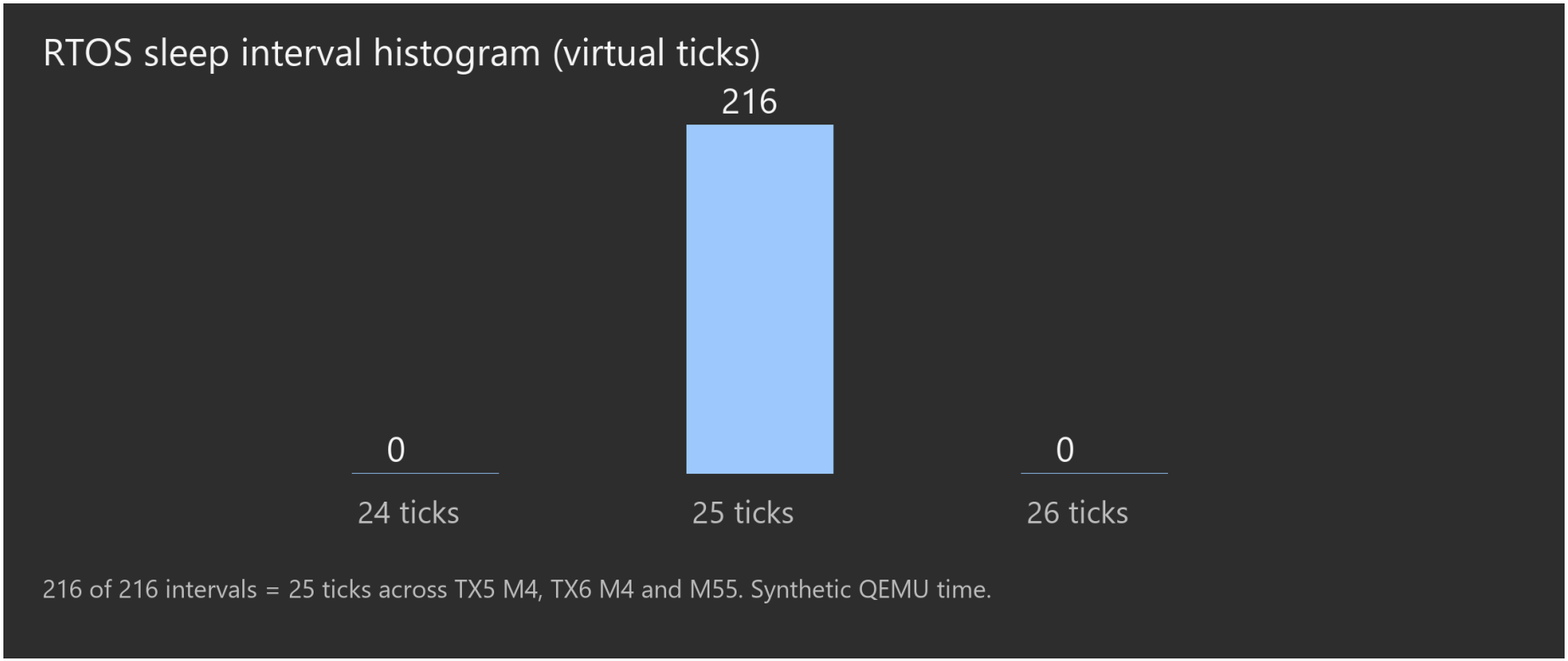


Every plotted sample is in evidence/summary.json.

Use this to budget an automated test job and spot host variability. Do not turn the median differences into CPU or kernel performance claims.

35 Histogram: scheduler timing contract

216 measured intervals, all at the requested 25 ticks



8 runs × 9 intervals × 3 images = 216 intervals.

The 100 Hz tick gives 250 ms in the configured virtual-time model on both kernels. Synthetic instruction time cannot establish physical sub-tick jitter.

36 What the M85 build proves

Correct target attributes and resolved symbols, not execution

Actual attribute inspection; build uses upstream ports/cortex_m85/gnu

```
llvm-readelf -A build/m85/threadx_gpio_demo.elf
CPU_name: cortex-m85
CPU_arch: ARM v8.1-M Mainline
FP_arch:  ARMv8-a FP-D16      ABI_VFP_args: AAPCS VFP
MVE_arch: MVE integer and float
llvm-nm --undefined-only build/m85/threadx_gpio_demo.elf
# no undefined symbols
```

Clang compiled and LLD linked the unchanged application with the M85 port. The demo linker map and clock are not a production M85 SoC definition.

There is no M85 runtime, performance, power, PACBTI or mixed FPU/MVE correctness result.

37 The hardware acceptance gate

Turn this proof into a product migration with explicit tests

Workstream	Required evidence before production
Board / SoC	Vendor startup, linker map, GPIO polarity, clock tree, IRQ map
Context / ABI	Preemption + nested IRQ + mixed scalar/FPU/MVE tasks; inspect saved context
Security	TrustZone policy, secure gateways/stacks, MPU and optional PACBTI
Performance	On-silicon cycles, IRQ latency, deadline/WCET, throughput and energy, TX5 vs TX6
Application	Peripheral, network, watchdog and low-power regression, incl. tickless idle
Governance	Licensing, SBOM, safety/change-impact assessment, traceable requirements

The proof reduces feasibility risk. It does not claim that Wi-Fi, TLS, OTA, safety certification or all application deadlines have migrated.

38 Reproduce the result

Public source, pinned dependencies, raw evidence and automated checks

Tools: Clang/LLVM, LLD, CMake, Ninja, Python and ARM QEMU

```
git clone --recurse-submodules https://github.com/zuwasi/TX5-upgrade.git
# fetch the external SDK into TX5_ROOT (see README), then:
cmake --preset tx5-m4  && cmake --build --preset tx5-m4
cmake --preset tx6-m4  && cmake --build --preset tx6-m4
cmake --preset tx6-m85 && cmake --build --preset tx6-m85
cmake --preset m55     && cmake --build --preset m55
ctest --test-dir build/tx5-m4 ; ctest --test-dir build/m4
ctest --test-dir build/m55
python -m unittest discover -s tests -p test_verifier.py -v
python scripts/collect_evidence.py --repeats 8
```

GitHub CI runs all four images, including Stage 1 against a fresh SDK clone. CI uses Linux packaged tools, so versions may differ from the recorded Windows run.

Only run firmware you trust with host semihosting enabled.

39 The technical conclusion

A bounded yes, backed by executable evidence

- YES: the application runs unchanged on the real ThreadX 5.6 Cortex-M4 kernel and on ThreadX 6.5.
- YES: behaviour, service checks and every tick match between TX5 and TX6 on the same core.
- YES: the same application compiles and links with the upstream Cortex-M85 port.
- LEARNED: with a binary kernel, its ABI and header configuration are part of the contract.
- NOT YET: M85 runtime and real-board BSP / timing / security qualification.

Next: choose the target M85 SoC and run the hardware acceptance suite, including a TX5-vs-TX6 cycle comparison on M4 silicon if that baseline matters.

Public source: github.com/zuwasi/TX5-upgrade
Presentation: zuwasi.github.io/Public-html-pages/tx5-upgrade/

40 Sources and evidence trail

Primary references and local measured artifacts

- WICED SDK 3.1.0 (external TX5 kernel source): https://github.com/wwwlllllll/bcm-wiced-sdk
- WICED GPIO behaviour reference: https://github.com/wwwlllllll/bcm-wiced-sdk/tree/master/apps/snip/gpio
- Originally linked ThreadX release: https://github.com/eclipse-threadx/threadx/tree/v6.1.12_rel
- Selected release: https://github.com/eclipse-threadx/threadx/tree/v6.5.2.202603_rel
- ThreadX release history: https://github.com/eclipse-threadx/threadx/releases
- Current MIT license: https://github.com/eclipse-threadx/threadx/blob/v6.5.2.202603_rel/LICENSE.txt
- Arm Cortex-M85: https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m85
- Arm M-profile architecture: https://www.arm.com/architecture/cpu/m-profile
- Arm Helium: https://www.arm.com/technologies/helium
- QEMU MPS2/MPS3 development docs: https://qemu-project.gitlab.io/qemu/system/arm/mps2.html
- Raw measurements: evidence/summary.json
- Stage 1 TX5 M4 console: evidence/tx5-m4-console.txt
- Stage 2 TX6 M4 console: evidence/m4-console.txt
- Supplement M55 console: evidence/m55-console.txt

Source and licence inventory checked on 2026-10-11. Runtime evidence records the exact compiler, emulator version, image and archive hashes and the virtual-time model.

41 Planning an embedded migration?

Talk to ESL about optimizing or migrating your embedded project



Embedded software consultancy specializing in optimizing and migrating embedded projects. Parasoft and Perforce partner in Israel for over 20 years.

- Contact ESL: https://eswlab.com/contact-us/
- Daniel Liezrowice on LinkedIn: https://www.linkedin.com/in/liezrowice/
- Source and evidence: https://github.com/zuwasi/TX5-upgrade