FPGA-Based Real-Time Task Profiler & Deadline Monitor

Hardware Event Logger with Single-Cycle Deadline Detection

Teresa Co (tc3499) Handong He (hh3152) Xiao Lu (xl3586)

Columbia University • CSEE 4840 Embedded System Design • Spring 2026

Implemented on Terasic DE1-SoC (Cyclone V 5CSEMA5F31C6 + ARM Cortex-A9 HPS)

Motivation: Limits of Software Timing

Why measure task execution in hardware?

Software profiling tools introduce overhead and noise into the very measurements they try to take. This is especially serious for microsecond-scale tasks on a non-RTOS Linux kernel.

Measurement overhead

clock_gettime() and similar POSIX calls take 200-500 ns each. For a 6 µs task, that pollutes the measurement by 3-8%.

Interrupt-induced jitter

Linux is not an RTOS. Timer interrupts, scheduler preemption, and other kernel activity preempt unpredictably between START and END markers.

Insufficient resolution

Linux software clocks resolve to ~1 µs. Sub-microsecond timing windows -- which exist in tight real-time loops -- are fundamentally invisible.

→ A hardware-side timestamping path eliminates all three problems at once.

Project Goals

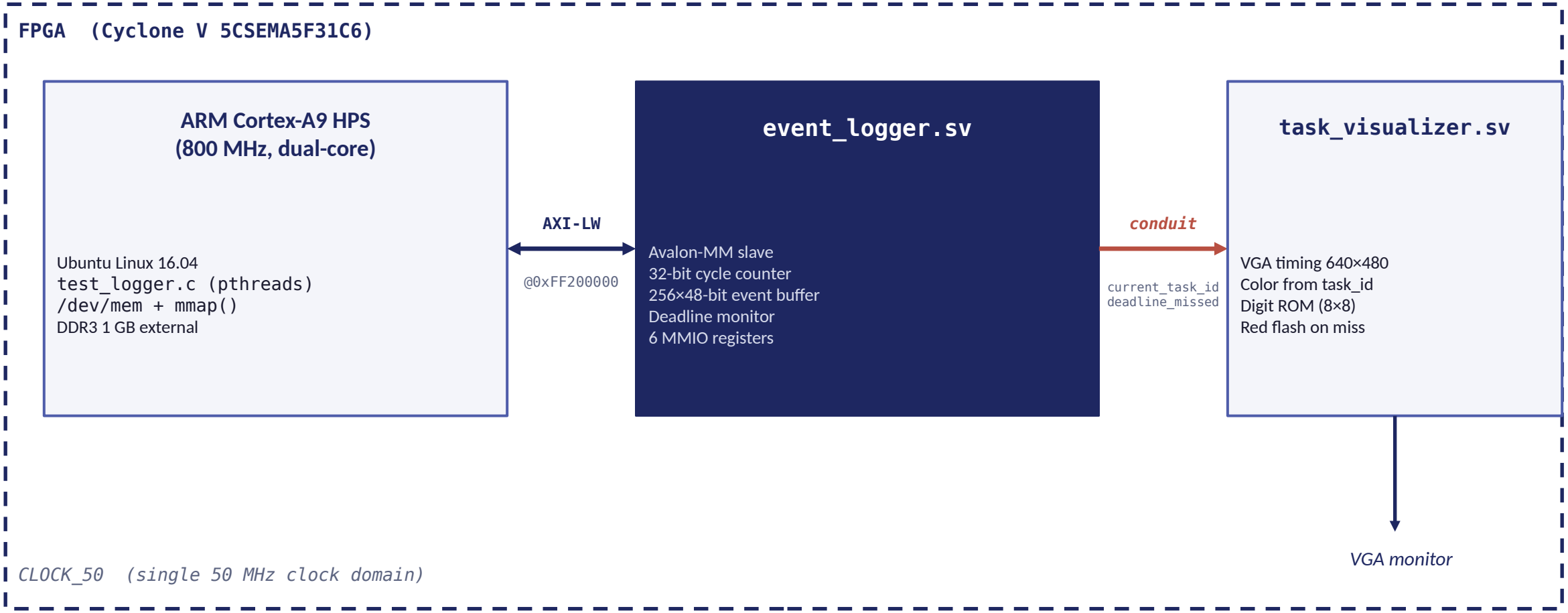
Hardware profiling, deadline detection, and live visualization

1.	**Hardware Event Logger**	Capture TASK_START/END events with FPGA-side cycle timestamps; zero software overhead beyond a single 32-bit MMIO write.
2.	**20 ns Timing Resolution**	Free-running 32-bit counter clocked by the 50 MHz CLOCK_50. Every event recorded with 1 cycle = 20 ns granularity.
3.	**Single-Cycle Deadline Check**	Configurable 32-bit deadline register. Each TASK_END is compared against the matching TASK_START in 1 clock cycle; sticky flag on violation.
4.	**VGA Visualization**	Frame-buffer-free hardware renderer drives 640×480@60Hz directly. Background color + centered digit + red flash on deadline miss.
5.	**Linux + FPGA Integration**	User-space Linux on ARM Cortex-A9 HPS accesses the FPGA peripheral via /dev/mem mmap on the lightweight HPS-to-FPGA AXI bridge.

Note: original design targeted DE0-Nano + NIOS-V + FreeRTOS. Hardware unavailable. Ported to DE1-SoC (ARM HPS + Linux). The event_logger SystemVerilog is unchanged across the port — confirming the IP's platform portability.

System Architecture

Two-domain design: HPS software path + FPGA hardware path



Key property: the visualization path (event_logger → conduit → task_visualizer → VGA pins) is entirely inside the FPGA. After a single CPU write, the screen reflects the new state within one clock cycle, with no software polling.

Event Logger Internals

Timestamp capture, storage, and deadline check on every write

event_logger.sv — core capture logic (simplified)

```
always_ff @(posedge clk) begin
    if (write_event && !buffer_full) begin
        buf_ts[wr_ptr] <= cycle_counter;       // capture in 1 cycle
        buf_ev[wr_ptr] <= writedata[15:0];
        wr_ptr         <= wr_ptr + 1;

        if (in_event_type == TASK_START) begin
            current_task_id   <= in_task_id;   // → conduit
            last_start_ts     <= cycle_counter;
            start_outstanding <= 1;
        end
        else if (in_event_type == TASK_END) begin
            if (start_outstanding && deadline_cycles != 0)
                if ((cycle_counter - last_start_ts)
                                    > deadline_cycles)
                    deadline_missed_reg <= 1;  // → conduit
            start_outstanding <= 0;
        end
    end
end
```

1. Timestamp = cycle_counter at the exact clock edge of the bus write. No race with software.

2. Buffer split into ts[256] and ev[256] arrays. Each maps cleanly onto one M10K block.

3. current_task_id is exposed as a conduit signal — fed continuously to task_visualizer.

4. Deadline comparison is a single subtraction + compare: completes in the same cycle as the TASK_END write.

Avalon-MM Register Map

Six word-addressed registers at base 0xFF200000 (HPS-to-FPGA Lightweight bridge)

Word Offset	Byte Address	Name	Access	Description
0	0xFF200000	EVENT_WRITE	W	[15:8] event_type, [7:0] task_id — triggers timestamp capture
1	0xFF200004	READ_TIMESTAMP	R	32-bit timestamp at current rd_ptr
2	0xFF200008	READ_INFO	R	[15:8] event_type, [7:0] task_id at current rd_ptr
3	0xFF20000C	STATUS	R/W	Read: status bits. Write any value: advance rd_ptr.
4	0xFF200010	CONTROL	W	[0]=clear all, [1]=clear deadline_missed only
5	0xFF200014	DEADLINE	W	Deadline in clock cycles. 0 = monitor disabled.

STATUS register bit layout

[8:0] entry_count (0..256)
[16] overflow_flag
[17] buffer_full
[18] deadline_missed (sticky)

Event type codes

0x01 EVT_TASK_START
0x02 EVT_TASK_END

Deadline Monitor

Single-cycle hardware violation detection with sticky reporting

1 Configure

Software writes a 32-bit deadline (in clock cycles) to the DEADLINE register before tasks start. Setting DEADLINE = 0 disables monitoring entirely.

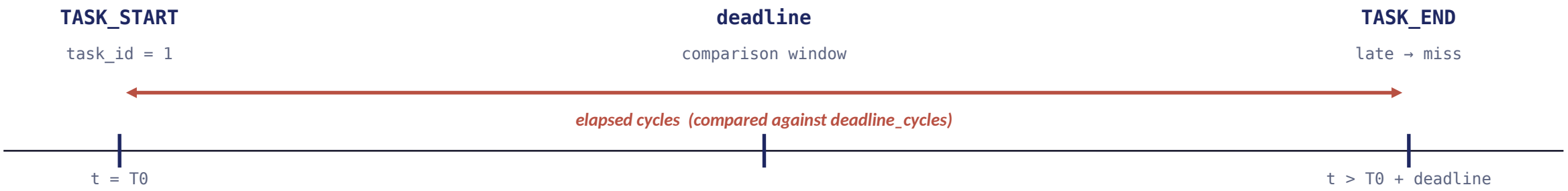
2 Track

On every TASK_START, hardware latches the current cycle_counter into last_start_ts and asserts start_outstanding. No software involvement.

3 Detect

On every TASK_END, hardware computes (cycle_counter − last_start_ts) > deadline_cycles in one cycle. Violation → deadline_missed_reg ← 1 (sticky).

Timing in cycles (50 MHz, 1 cycle = 20 ns)



Detection latency = 1 clock cycle = 20 ns. Software polling would take 200-500 ns minimum and could miss sub-µs overruns entirely.

VGA Visualization

Frame-buffer-free, combinational pixel pipeline driven directly by event_logger conduit

Background color from current_task_id

task_id = 0	0x20, 0x20, 0x20	*(grey)*
task_id = 1	0x20, 0x40, 0xC0	*(blue)*
task_id = 2	0x20, 0xA0, 0x40	*(green)*
task_id = 3	0xE0, 0x80, 0x20	*(orange)*
task_id = 4+	0x80, 0x20, 0xC0	*(purple)*

Centered digit rendering

An 8×8 1-bit bitmap ROM stores glyphs for digits 0–9. Each ROM pixel is scaled 16× horizontally and vertically, producing a 128×128 white digit centered at (320, 240) on the 640×480 frame.

8×8 ROM glyph for '2'

```
 X X X X . .
X . . . X .
. . . X . .
. . X . . .
. X . . . .
X . . . . .
X X X X X .
```

Deadline miss override

When deadline_missed=1, every pixel is set to a flashing red pattern (toggles bright/dark on hcount[7]). Overrides background and digit until the flag is cleared via CONTROL[1].

Design choice: no frame buffer. Pixel color is computed combinationally from (px, py, current_task_id, deadline_missed). Only synthesised patterns can be drawn — sufficient for our visualization.

Live demo will show all colors, digit rendering, and deadline-miss red flash on the actual DE1-SoC.

Qsys / Platform Designer Integration

Exporting the conduit so task_visualizer can read event_logger signals

Integration workflow

1. Define the component in event_logger_hw.tcl with three interfaces: clock, reset, avalon_slave — plus the new conduit "viz" with current_task_id and deadline_missed.

2. In Platform Designer, instantiate event_logger_0 and connect clock/reset/avalon_slave to the HPS lightweight bridge.

3. Right-click "viz" → Export. This makes the conduit visible at the soc_system top-level boundary as named ports.

4. Generate HDL. The resulting soc_system.v has output ports event_logger_0_viz_current_task_id [7:0] and event_logger_0_viz_deadline_missed.

5. In soc_system_top.sv (outside Qsys), wire these top-level ports directly to task_visualizer.

event_logger_hw.tcl (excerpt)

```
# Avalon-MM slave (omitted)

# Conduit: visualization signals
add_interface viz conduit end
set_interface_property viz \
    associatedClock clock
set_interface_property viz \
    associatedReset reset

add_interface_port viz \
    current_task_id \
    current_task_id Output 8

add_interface_port viz \
    deadline_missed \
    deadline_missed Output 1
```

Linux Software Stack

User-space C program drives the FPGA via /dev/mem on ARM HPS

test_logger.c (essential I/O path)

```
int fd = open("/dev/mem", O_RDWR | O_SYNC);
void *map = mmap(NULL, 0x200000,
                 PROT_READ | PROT_WRITE,
                 MAP_SHARED, fd, 0xFF200000);
volatile uint32_t *base = (volatile uint32_t*)map;

/* Configure */
base[4] = 1;          /* CONTROL: clear all */
base[5] = 100000;     /* DEADLINE = 100k cycles = 2 ms */

/* Within each pthread: */
base[0] = (0x01 << 8) | task_id;  /* TASK_START */
for (int j = 0; j < N; j++) sum += j;
base[0] = (0x02 << 8) | task_id;  /* TASK_END   */

/* After workload: drain buffer */
uint32_t cnt = base[3] & 0x1FF;
for (uint32_t i = 0; i < cnt; i++) {
    uint32_t ts   = base[1];
    uint32_t info = base[2];
    printf("%u,%u,%u\n", ts, (info>>8)&0xFF, info&0xFF);
    base[3] = 1;                  /* advance rd_ptr  */
}
```

O_SYNC

Maps the bridge non-cacheable. Every store immediately propagates to the AXI bus — not deferred by ARM L1/L2 cache.

volatile

Forces the compiler to issue real loads/stores; prevents reordering or coalescing of register accesses.

mmap()

Connects /dev/mem at 0xFF200000 into the process address space. Requires root privileges.

pthreads

Two POSIX threads simulate concurrent tasks. Each gets a unique task_id and a different inner-loop size.

Measured Results

Hardware-captured data from a 10-second pthread workload run

40	0	0	0.014 %
events captured	buffer overflows	deadline misses	period error

Per-task timing (measured by hardware, reported by Python analyser)

Metric	Measured	Target	Error	Note
Task 1 execution time	5.84 µs (292 cycles)	(workload-defined)	–	*1000-iteration sum loop*
Task 2 execution time	27.4 µs (1372 cycles)	≈ 5× Task 1	+ 6 %	*5000 iterations — ratio matches*
Task 1 period	500.07 ms	500.00 ms (usleep)	0.014 %	*very stable across 10 iterations*
Task 2 period	1000.13 ms	1000.00 ms (usleep)	0.013 %	*Linux scheduler noise within µs*
Buffer fill	40 / 256 (15.6 %)	≤ 256	–	*no overflow, no data loss*
Deadline misses	0	0 (deadline = 2 ms)	–	*all tasks well under 2 ms*

Validation: Task 2 / Task 1 execution-time ratio of 4.7× matches the 5× loop-size ratio (small deviation from compiler optimisation). Sub-0.02% period error confirms the 50 MHz cycle counter is precise. No overflows or deadline misses under the chosen 2 ms threshold.

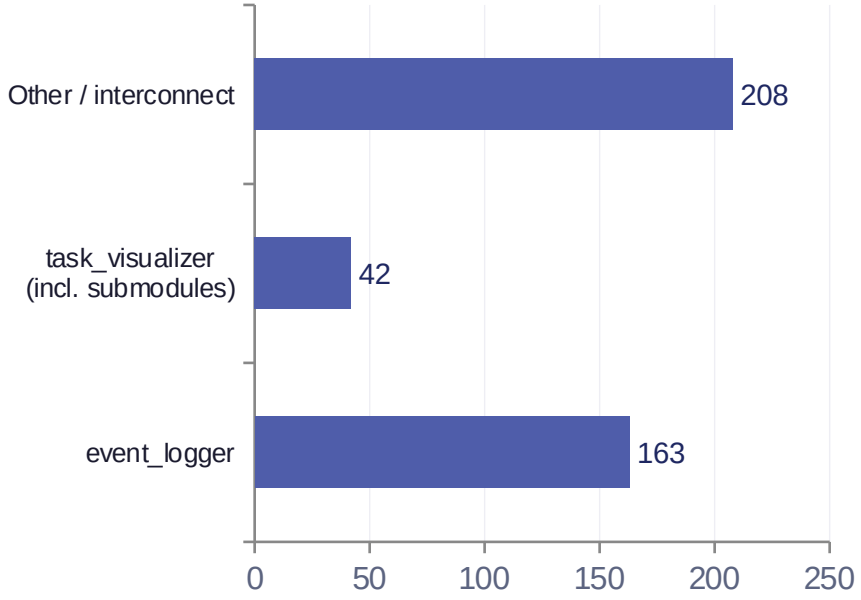
FPGA Resource Utilization

From soc_system.fit.summary — Quartus Prime 21.1, Cyclone V 5CSEMA5F31C6

Total system resources

Resource	Used	Available	Utilization
Logic utilization (ALMs)	413	32,070	**1 %**
Total registers	823	—	**—**
Total block memory bits	12,288	4,065,280	**< 1 %**
RAM blocks (M10K)	2	397	**< 1 %**
DSP blocks	0	87	**0 %**
PLLs	0	6	**0 %**
DLLs	1	4	**25 %**
Total pins	362	457	**79 %**

ALM breakdown by entity



task_visualizer internals

`vga_counters:` 16.0 ALMs (31 registers)
`digit_rom:` 13.7 ALMs (25 LUTs, combinational)
`top color logic:` 12.4 ALMs

Key observations

- Total design uses only **1%** of available ALMs — substantial headroom for integration.
- 2 M10K blocks total: `buf_ts[256] (8,192 bits) + buf_ev[256] (4,096 bits) = 12,288 bits`.
- High pin count (79%) is from the HPS interface — the FPGA fabric usage itself is minimal.

Synthesis & Timing Closure

Quartus Prime 21.1 full-flow compilation, 50 MHz target clock

+2.110 ns

Worst-case setup slack

Margin against 20 ns clock period

+0.077 ns

Worst-case hold slack

All paths meet hold requirement

0

Compilation errors

Clean synthesis + fit + assembler

~400

Warnings

All from HPS subsystem, expected

Interpretation

Setup slack +2.110 ns: the critical combinational path takes at most **`17.89 ns`** between two flip-flops. At 50 MHz the period is 20 ns, so timing closes with 10.5 % margin.

Hold slack +0.077 ns: shortest path is just safe — no hold violations.

The ~400 warnings are entirely from the HPS subsystem: DDR3 calibration, unconstrained I2C/USB clocks. None affect the custom IP functionality.

Compile stages (elapsed)

Stage	Time
Analysis & Synthesis (quartus_map)	**`2:50`**
Fitter (quartus_fit)	**`0:54`**
Assembler (quartus_asm)	**`0:12`**
Timing Analyzer (quartus_sta)	**`0:24`**

*Total ≈ **`4:20`** for a full compile.*

Bitstream output: soc_system.sof (7,349,916 bytes ≈ 7.0 MB) programmed to FPGA via JTAG; live demo follows.

Engineering Challenges

Three issues we encountered and resolved during development

1. Qsys conduit not re-read

Problem: After updating event_logger_hw.tcl to declare the new viz conduit, the generated soc_system.v still lacked the new ports. quartus_sh --flow compile regenerated Qsys before the GUI re-ingested the .tcl.

Resolution: Delete soc_system/synthesis/, db/, output_files/. Reopen Platform Designer, right-click event_logger_0 → Edit (this forces re-read), export viz, Generate HDL, then run full compile.

2. VGA color not updating

Problem: After successful programming, the VGA showed dark grey indefinitely even while test_logger was running. A manual test (test_color.c) writing different task_ids did change the color, confirming the conduit was functional.

Resolution: The original event_logger.sv cleared current_task_id to 0 on TASK_END. Since tasks complete in 6–28 µs but a VGA frame is 16.7 ms, the colored state was invisible. Removed the clear; current_task_id now holds until next TASK_START.

3. Platform pivot DE0-Nano → DE1-SoC

Problem: Original design targeted DE0-Nano + NIOS-V soft core + FreeRTOS, but only DE1-SoC boards were available. Quartus 21.1 (which we used) does not support NIOS-V on Cyclone V.

Resolution: Ported to ARM Cortex-A9 HPS + Linux. event_logger.sv itself required no changes; only the bus master and the software access method (mmap on /dev/mem) differ. Demonstrates the IP's cross-platform portability.

Conclusion

A hardware-side event logger with single-cycle deadline detection provides 20 ns timing resolution and ~100 ns measurement overhead — substantially better than any software approach — at a cost of fewer than 1000 ALMs on the Cyclone V FPGA.

Deterministic profiling

20 ns resolution, hardware-captured timestamps, immune to OS jitter.

Single-cycle enforcement

Deadline check completes in 1 clock cycle; software polling cannot match this.

Live visualization

Frame-buffer-free VGA renderer reflects task state in real time.

Co-designed integration

Avalon-MM register interface enables clean Linux user-space access.

Thank you. Questions?

Teresa Co • Handong He • Xiao Lu | CSEE 4840 • Spring 2026