ISNN 3030-3907

2026
JUNE
ISSUE - 6
VOLUME -6

DEVELOPMENT OF SCIENCE

scientific journal



Development of science

Ilmiy jurnal

IYUN 2026
Issue - 6, VOLUME – 6

ISSN 3030 -3907

Ilmiy jurnal OAK rayosatining 2024 yil 27 sentabrdagi 361-son qaroriga asosan 02.00.00 kimyo fanlari hamda 2025 yil 12 fevral 367- son qaroriga asosan 05.00.00 texnika fanlari bo‘yicha falsafa doktori (PhD) va fan doktori (DSc) ilmiy darajasiga talabgorlarning dissertatsiya ishlari yuzasidan dissertatsiyalari asosiy ilmiy natijalarini chop etish tavsiya etilgan milliy ilmiy nashrlar ro‘yxatiga kiritilgan.

BOSH MUHARRIR:
Temirov Alisher Hoshim o‘g‘li – texnika fanlari falsafa doktori (PhD),Dotsent.
TAHRIRIYAT KENGASH RAISI:
Fozilov Sadriddin Fayzullayevich – texnika fanlari doktori, professor
TAHRIRIYAT KENGASH A’ZOLARI

TEXNIKA FANLARI/ TECHNICAL SCIENCES/ ТЕХНИЧЕСКИЕ НАУКИ

Bozorov G‘ayrat Rashidovich
Buxoro Davlat Texnika Universiteti, professor
Abdurahmonov Olim Rustamovich
Buxoro Davlat Texnika Universiteti, professor
Adizov Bobur Zamirovich
O‘zbekiston Respublikasi Fanlar akademiyasi Umumiy va noorganik kimyo instituti, professor
Hayitov Ruslan Rustamjonovich
Buxoro Davlat Texnika Universiteti, professor
Jumayev Qayum Karimovich
Buxoro Davlat Texnika Universiteti, professor.
Berdiyev Dorob Murodovich
Islom Karimov nomidagi Toshkent Davlat Texnika Universiteti, professor
Maxmudov Rafiq Amonovich
Buxoro Davlat Texnika Universiteti, professor.
Tilloyev Lochin Ismatilloyevich
Buxoro Davlat Texnika Universiteti, dotsent.
Raxmonov Qahramon Sanakulovich
Buxoro Davlat Texnika Universiteti, dotsent.
Barakayev Nusratullo Rajabovich
Islom Karimov nomidagi Toshkent Davlat Texnika Universiteti, professor.
Jo‘rayev Xayrullo Fayziyevich
Buxoro Davlat Texnika Universiteti, professor.
Panoyev Erali Rajabboyevich
Buxoro Davlat Texnika Universiteti, dotsent.
Ataullayev Sherzod Nabiullyevich
Buxoro Davlat Texnika Universiteti, dotsent.
Murodov Malik Negmurodovich
Buxoro Davlat Texnika Universiteti, dotsent.
Axmedova Ozoda Bahronovna
Islom Karimov nomidagi Toshkent Davlat Texnika Universiteti.
Safarov Bahri Jumayevich
Buxoro Davlat Texnika Universiteti, dotsent.
Xaydarov Axtam Amonovich
Buxoro Davlat Texnika Universiteti, dotsent.
Safarov Jasur Alijon o‘g‘li
Buxoro Davlat Texnika Universiteti, dotsent.

KIMYO FANLARI/ CHEMICAL SCIENCES/ ХИМИЧЕСКИЕ НАУКИ

Axmedov Voxid Nizomovich
Buxoro Davlat Texnika Universiteti, professor
Umarov Baqo Bafoyevich
Buxoro Davlat Universiteti, professor
Kodirov Abduaxad Abduraximovich
Qarshi davlat Universiteti, professor
Qurbonov Mingniqul Jumagulovich
Qarshi davlat Universiteti, dotsent
Turemuratov Sharibay Naurizbaevich
O‘zbekiston Respublikasi Fanlar akademiyasi Qoraqalpog‘iston bo‘limi Bosh ilmiy kotibi, professor
Tursunov Kaxor Shonazarovich
Qarshi davlat Universiteti, professor
Do‘stov Hamro
Buxoro Davlat Texnika Universiteti, professor
Panoyev Nodir Shavkatovich
Buxoro Davlat Texnika Universiteti, dotsent.
Olimov Bobur Bahodir o‘g‘li
Buxoro Davlat Texnika Universiteti, dotsent.
Kasimov Sherzod Abduazimovich
Termiz Davlat Universiteti, professor.
Aliqulov Rustam Valiyevich
Termiz Davlat Universiteti, professor
Tropov Xamza Tursunovich
Samarqand Davlat Universiteti, professor

IQTISODIYOT FANLARI/ ECONOMICS/ ЭКОНОМИЧЕСКИЕ НАУКИ

Nizamov Aslitdin Badreddinovich
Buxoro Davlat Texnika Universiteti, professor.
Boboyev Akmal Choriyevich
Buxoro Davlat Texnika Universiteti, professor.
Ergashxojayeva Shaxnoza Djasurovna
Toshkent davlat iqtisodiyot universiteti, professor
Saidaxmedova Nodira Ilxomovna
Toshkent davlat iqtisodiyot universiteti, professor
Muxsinov Bekzod Toxirovich
Buxoro Davlat Texnika Universiteti, dotsent.
Shakarov Qulmat Ashirovich
Renessans ta'lim universiteti,dotsent.
Jumaeva Zulfiya Qayumovna
Buxoro Davlat Texnika Universiteti, dotsent.
Norova Salomat Yusupovna
Buxoro Davlat Texnika Universiteti, dotsent.
Avezova Shaxnoza Maxmudjonovna
Buxoro Davlat Texnika Universiteti, dotsent.
Jalilov Jamshid G‘anijonovich
Toshkent davlat iqtisodiyot universiteti, dotsent
Xasanova Gulruh Djumanazarovna
Buxoro Davlat Texnika Universiteti, dotsent.

PEDAGOGIKA FANLARI/ PEDAGOGICAL SCIENCES / ПЕДАГОГИЧЕСКИЕ НАУКИ

Siddiqova Sadoqat G‘afforovna
Buxoro Davlat Texnika Universiteti, dotsent.
Zoyirov Erkin Xalilovich
Buxoro Davlat Texnika Universiteti, DSc, professor
Gaffarov Feruz Hasanovich
Puchon Universiteti, *professor*
Ruziyev Davron Yuldashevich
Buxoro Davlat Universiteti, professor
Muradova Firuza Rashidovna
Buxoro Davlat Texnika Universiteti, professor.
Yunusova Gulandon Samiyevna
Buxoro Davlat Texnika Universiteti, professor.
To‘xtayeva Zebo Sharifovna
Buxoro Davlat Texnika Universiteti, professor.
Navro‘zova Gulchixra Nigmatovna
Buxoro Davlat Texnika Universiteti, dotsent.
Uzoqova Laylo Polvonovna
Buxoro muhandislik – texnologiya instituti, dotsent.
To‘raqulova Marjona Qiyom qizi
Buxoro Davlat Texnika Universiteti, dotsent.
Muradova Zarina Rashidovna
Buxoro Davlat Texnika Universiteti, dotsent.
Murodov Sanjar Aslonovich
Buxoro Davlat Texnika Universiteti, dotsent.
Shodiyev Jahongir Jo‘raqulovich
Buxoro Davlat Texnika Universiteti, dotsent
Gaffarov Laziz Hasanovich
Buxoro muhandislik – texnologiya instituti, dotsent.

Tahririyat a’zolarning to‘liq ro‘yxati jurnalning www.devos.uz saytida keltirilgan.

Tahririyat hay’ati a’zolarining tarkibini o‘zgartirish hamda qo‘shimcha kiritish yoki tahririyat a’zoligidan chiqarish bosh muharrir tomonidan amalga oshiriladi.

ISSN 3030-3907

Development of science ilmiy jurnali 2024 yil 15 aprel kuni №256445 sonli guvohnoma bilan Davlat ro‘yxatidan o‘tgan.

To‘plam elektron ko‘rinishda (pdf formatida) mualliflarga taqdim etiladi. To‘plamga kiritiladigan maqolalarning mazmuni, ulardagi statistik ma’lumotlar va me’yoriy hujjatlarning to‘g‘riligiga hamda har qanday fikr - mulohazalarga mualliflarning o‘zlari mas’ul hisoblanadilar. Talablarga javob bermaydigan maqolalar to‘plamga kiritilmaydi. Tashkiliy qo‘mita maqola matnini qisqartirish, qisman tuzatish kiritish, seksiyalarga joylashtirish huquqiga ega.

ISSN 3030-3907

LAZY SYNCHRONIZATION WITH INTERPOLATED STATE FOR DISCRETE–CONTINUOUS CO-SIMULATION OF MICROCONTROLLER-DRIVEN CIRCUITS

Adizbek Ergashev
Tashkent University of Information Technologies (TUIT), Tashkent, Uzbekistan

Abstract. Cycle-accurate co-simulation of a microcontroller and the circuit it drives is dominated by the cost of advancing peripheral models on every instruction, which precludes interactive speed on commodity hardware. We model the problem as the synchronous product of a timed transition system (the CPU) and a constraint-graph solver (the circuit), coupled by explicit state-lifting functions, and formalize a two-tier scheduler that evaluates peripherals only once per N CPU cycles. We define a peripheral monotonicity condition and prove that under it the lazy schedule is observationally equivalent to the per-cycle eager schedule at synchronization boundaries (Theorem 1). For high-read-frequency timer registers, the main source of firmware-visible drift, we give a closed-form interpolation algorithm and prove it returns the eager value while the timer configuration is unchanged (Theorem 2). A cost model predicts a throughput knee near $N = t_s / (t_\delta - t_{\delta f})$, and a circuit solver and a multi-MCU topology binder are shown to meet the per-synchronization budget with proven complexity bounds. A six-workload drift benchmark confirms zero architectural-state drift on monotone-read workloads across N up to 4096 and event drift bounded by N (worst case 258 cycles at N = 512); the scheduler yields an 11.7–43.6× speedup over per-cycle evaluation. The methods are implemented in the Logiclabuz platform [1].

Keywords: co-simulation; cycle-accurate emulation; scheduler design; interpolation; lazy evaluation; embedded systems.

Annotatsiya. Mikrokontroller va u boshqaradigan zanjirni takt aniqlikdagi (cycle-accurate) birgalikda simulyatsiya qilish har bir buyruqda periferiya modellarini ilgarilatish xarajati bilan cheklanadi, bu esa oddiy uskunada interaktiv tezlikka erishishga to'sqinlik qiladi. Biz muammoni vaqtli o'tish tizimi (protsessor) va cheklovlar grafi yechuvchisi (zanjir) ning sinxron ko'paytmasi sifatida modellashtiramiz, ularni oshkora holat-ko'taruvchi funksiyalar orqali bog'laymiz va periferiyani har N ta protsessor taktida faqat bir marta baholaydigan ikki bosqichli rejalashtiruvchini (scheduler) formallashtiramiz. Periferiya monotonligi shartini aniqlaymiz va ushbu shart ostida dangasa (lazy) jadval sinxronizatsiya chegaralarida har takt bo'yicha qat'iy (eager) jadvalga kuzatuv jihatidan ekvivalent ekanligini isbotlaymiz (1-teorema). Mikrodastur uchun ko'rinadigan og'ishning asosiy manbai bo'lgan, tez-tez o'qiladigan taymer registrlari uchun yopiq ko'rinishdagi interpolyatsiya algoritmini taklif qilamiz va taymer konfiguratsiyasi

ISSN 3030-3907

o'zgarmagan holatda u qat'iy jadval qiymatini qaytarishini isbotlaymiz (2-teorema). Narx modeli o'tkazuvchanlik «tirsagi»ni N = t_s / (t_δ − t_δf) atrofida bashorat qiladi; zanjir yechuvchisi va ko'p mikrokontrollerli topologiya bog'lovchisi har bir sinxronizatsiya byudjetiga isbotlangan murakkablik chegaralari bilan mos kelishi ko'rsatiladi. Oltita ish sinovidan iborat, og'ish sinovi monoton o'qiladigan ish yuklarida N = 4096 gacha arxitektura holatining nol og'ishini, hamda N bilan chegaralangan hodisa og'ishini (N = 512 da eng yomon holatda 258 takt) tasdiqlaydi; rejalashtiruvchi har takt bo'yicha baholashga nisbatan 11.7–43.6 baravar tezlashishni beradi. Usullar Logiclabuz platformasida amalga oshirilgan.

Kalit so'zlar: birgalikda simulyatsiya; takt aniqlikdagi emulyatsiya; rejalashtiruvchi loyihasi; interpolyatsiya; dangasa baholash; o'rnatilgan tizimlar.

Аннотация. Потактово-точное совместное моделирование микроконтроллера и управляемой им цепи определяется стоимостью продвижения моделей периферии на каждой инструкции, что препятствует достижению интерактивной скорости на обычном оборудовании. Мы моделируем задачу как синхронное произведение временно́й системы переходов (процессор) и решателя графа ограничений (цепь), связанных явными функциями подъёма состояния, и формализуем двухуровневый планировщик, вычисляющий периферию лишь раз в N тактов процессора. Мы определяем условие монотонности периферии и доказываем, что при нём ленивое расписание наблюдаемо эквивалентно поцикловому (энергичному) расписанию на границах синхронизации (Теорема 1). Для часто читаемых регистров таймера — главного источника видимого прошивке расхождения — мы приводим алгоритм интерполяции в замкнутой форме и доказываем, что он возвращает значение энергичного расписания, пока конфигурация таймера не изменена (Теорема 2). Стоимостная модель предсказывает «колено» пропускной способности вблизи N = t_s / (t_δ − t_δf); показано, что решатель цепей и связыватель топологии нескольких микроконтроллеров укладываются в бюджет на одну синхронизацию с доказанными границами сложности. Тест расхождения на шести нагрузках подтверждает нулевое расхождение архитектурного состояния на нагрузках с монотонным чтением вплоть до N = 4096 и расхождение событий, ограниченное N (в худшем случае 258 тактов при N = 512); планировщик даёт ускорение в 11.7–43.6 раза по сравнению с потактовым вычислением. Методы реализованы в платформе Logiclabuz.

Ключевые слова: совместное моделирование; потактово-точная эмуляция; проектирование планировщика; интерполяция; ленивые вычисления; встраиваемые системы.

1. INTRODUCTION.

In any cycle-accurate co-simulation of a microcontroller and the continuous-domain circuit it drives, the dominant cost is advancing the peripheral models on every instruction. A typical AVR-class chip ticks several timers, a USART, an SPI, an I²C, an ADC, and an interrupt controller, each requiring at least a function call and register bookkeeping per cycle; on a 16 MHz target this is roughly 10^8 peripheral updates per simulated second, exceeding the interactive real-time budget on commodity JavaScript engines by about an order of magnitude. The natural response is *lazy synchronization*: advance the CPU in tight loops of length N and evaluate peripherals once per loop. This is folklore in emulator engineering but is rarely formalized, leaving two questions open. First, under what condition does the lazy schedule produce the same trace as the eager one for firmware compiled against the silicon's semantics? Second, how should one treat registers such as the free-running timer counter TCNT0, which a delay loop may read many times per interval and where even small staleness is firmware-visible?

This paper answers both. We model the co-simulation as the synchronous product of a timed transition system (the CPU) and a constraint-graph solver (the circuit), coupled by explicit state-lifting functions (Section 3). We define *peripheral monotonicity at interval N* and prove that the lazy schedule is observationally equivalent to the eager schedule under it (Theorem 1, Section 4). For high-read-frequency timer registers we give a closed-form interpolation that recovers the eager value exactly while the timer configuration is unchanged (Theorem 2, Section 5). A cost model predicts the throughput knee that sets the default interval. We then bound the cost of the circuit solver and a multi-MCU topology binder (Propositions 1–4, Section 6) and validate the theory empirically (Section 7). The methods are implemented and in continuous use in the Logiclabuz platform [1]; the present paper is its algorithmic companion.

2. RELATED WORK

The reference AVR emulator simavr [2] and the avr8js core of Wokwi [3] tick each peripheral on every CPU cycle and do not document or formalize a lazy mode. The closest peer-reviewed work in event-driven full-platform simulation is GVSoC [4], a configurable RISC-V IoT simulator with an event-driven peripheral scheduler; it describes the engineering of the pattern but gives no formal correctness statement, which is the gap this paper addresses. Concurrent work on automatic peripheral modeling, FlexEmu [5] and the specification-guided approach of Zhou et al. [6], targets the orthogonal problem of deriving peripheral semantics rather than scheduling their evaluation cheaply. QEMU [7] trades cycle accuracy for translation throughput, and gem5 [8] uses event-driven scheduling at a per-instruction budget far larger than an interactive emulator allows.

ISSN 3030-3907

On the continuous side, SPICE-class engines [9] and co-simulation standards such as FMI [10] couple heavyweight models over millisecond-scale steps; recent MCU-plus-analog work pairs a RISC-V instruction-set simulator with SystemC-AMS for power modeling [11]. These occupy the heavy, desktop end of the design axis; our contribution is the light, per-cycle, browser-deployable end. The abstract framing of coupling two transition systems through a small interface follows Ptolemy II [12], and our interpolation is a special case of computing predictable future inputs lazily on read [13], specialized here to linear-update counters with a closed-form exactness guarantee. Four contributions distinguish this work: the monotonicity condition for lazy equivalence (Theorem 1); the interpolation-exactness theorem for linear counters (Theorem 2); the empirical confirmation of zero observable drift; and the topology-inferred multi-MCU binder, for which we found no peer-reviewed prior art in the simulator setting.

3. FORMAL MODEL

The CPU is a timed transition system, the circuit is a constraint graph, and the two are composed by an explicit pair of state-lifting functions.

Definition 1 (Microcontroller transition system). A microcontroller is a tuple $M = (Q, \Sigma, q_0, \delta, \tau)$ where Q is the state space (program counter, register file, status register, stack pointer, and a data space d that includes the memory-mapped peripheral registers); Σ is the input alphabet of opcodes and external events; q_0 is the reset state; $\delta : Q \times \Sigma \to Q$ is the AVR instruction semantics; and $\tau : Q \times \Sigma \to \mathbb{N}$ is the per-instruction cycle cost, so the cycle counter advances by $\tau(q, \sigma)$ on each transition.

Definition 2 (Circuit constraint graph). A circuit is a tuple $C = (N, E, \psi, \mathrm{comp})$ where N is a set of *nets* (maximal sets of equipotential pins), E is a set of components, $\psi : N \to \mathbb{R} \cup \{\perp\}$ is the net-voltage assignment, and comp maps each component pin-pair to one of eight electrical models (voltage source, ground, resistor, diode, short, voltage divider, open, ADC input). The solved state augments ψ with the per-component triple (V, I, P) computed by the solver of Section 6.

Definition 3 (Coupling). The forward coupling $\Phi : Q \times C \to C$ updates the circuit's driven-net voltages from the CPU's GPIO output registers (the per-port product of DDR and PORT bits) and PWM duty; the backward coupling $\Phi^{-1} : C \times Q \to Q$ writes the solved ADC and digital-input values back into the CPU. The *eager reference schedule*, against which every lazy schedule is compared, re-solves the circuit and applies both couplings on every cycle: $C_c = \mathrm{solve}(\Phi(q_\{c-1\}, C_\{c-1\}))$ and $q_c = \delta(\Phi^{-1}(C_c, q_\{c-1\}), \mathrm{opcode}(q_\{c-1\}))$. This is what simavr and avr8js do.

Definition 4 (Firmware-observable equivalence). Two states q, q' are observationally equivalent if they agree on pc, sp, sreg, the register file, and on $d(\cdot,$

ISSN 3030-3907

a) for every address a the firmware reads (every SRAM or peripheral register touched by an LDS, LD, IN, or skip-on-bit instruction). A lazy schedule that differs from the eager one only in registers the firmware never reads is, by this definition, observationally equivalent. This is the relation the theorems use.

4. THE TWO-TIER SCHEDULER

4.1 Algorithm and cost model

The eager schedule pays $O(\text{cost}(\delta) + \text{cost}(\text{solve}))$ per cycle, with cost(solve) exceeding cost(δ) by an order of magnitude in practice. The two-tier scheduler (Algorithm 1) runs the CPU in inner loops of length N using δ_fast, which is δ without peripheral side effects, then synchronizes peripherals and the circuit once per loop. Firmware writes to peripheral registers still commit to the data space immediately, so the firmware always observes its own writes; only the *hardware behavior* triggered by those writes is deferred to the next synchronization. Figure 1 contrasts the two schedules.

Algorithm 1 TwoTierStep(q, C, N)

```
target ← cycle(q) + N
while cycle(q) < target and q.running:
    q ← δ_fast(q, opcode_at(q))     # CPU step, no peripheral cost
C ← solve(Φ(q, C)); q ← Φ⁻¹(C, q)   # one synchronization per interval
return (q, C)
```

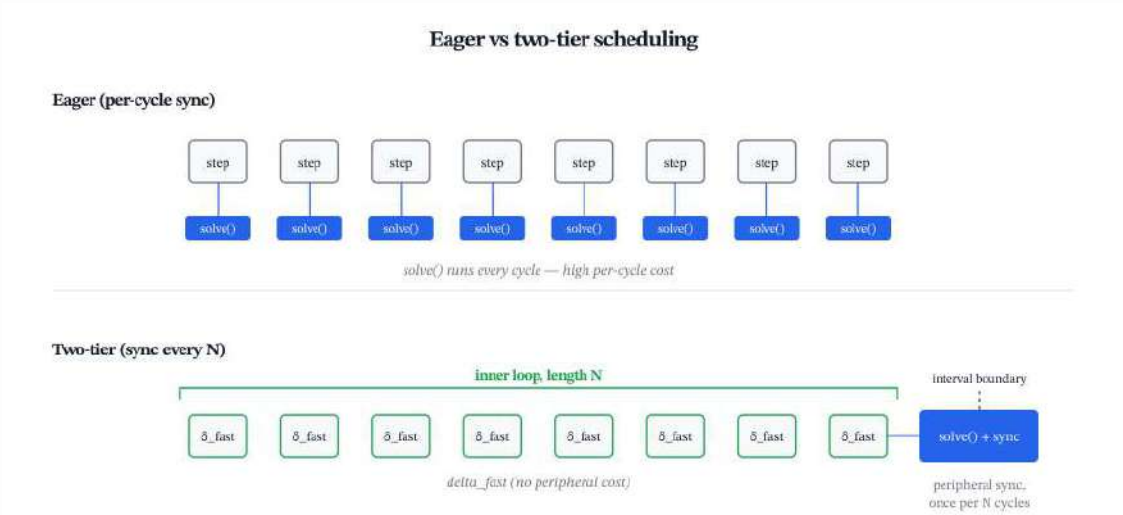


Figure 1. Eager versus two-tier scheduling. The eager schedule solves the circuit after every CPU cycle; the two-tier schedule runs N peripheral-free steps and solves once per interval.

Per interval the eager schedule costs $N \cdot t_\delta$ and the two-tier schedule $N \cdot t_{\delta f} + t_s$, where t_δ, $t_{\delta f}$, and t_s are the costs of δ, δ_fast, and one solve. The speedup

ISSN 3030-3907

is therefore $S(N) = N \cdot t_\delta / (N \cdot t_{\delta f} + t_s)$, which grows almost linearly while t_s dominates and saturates at $t_\delta / t_{\delta f}$ as $N \to \infty$. The two regimes meet at the knee $N^* = t_s / (t_\delta - t_{\delta f})$. The measured constants ($t_s \approx 4$–8 µs, $t_\delta - t_{\delta f} \approx 10$–$20$ ns) place the knee at 200–800 cycles, matching the empirical knee near $N = 512$ (Section 7) and justifying it as the default interval.

4.2 Observable equivalence

Definition 5 (Peripheral monotonicity at interval N). A peripheral register r is *monotone* over $[c, c+N)$ if one of the following holds: (1) the firmware does not read r in the interval; (2) the eager value of r at any $c' \in [c, c+N)$ is a function $f_r(c'-c, Q_{perif}(c))$ of the cycle offset and the peripheral configuration cached at c; or (3) r only takes firmware-visible values at synchronization boundaries (for instance a USART byte committed atomically at the end of its bit time). A program is monotone at interval N if every peripheral register it reads is monotone in this sense.

Theorem 1 (Lazy equivalence under monotonicity). If a circuit–program pair is peripheral-monotone at interval N, then at every synchronization boundary the lazy-schedule state is observationally equivalent to the eager-schedule state.

Proof. By induction on the boundary index k (boundary k at cycle kN). The base case holds because both schedules start at q_0 and solve once before any CPU work. Assume equivalence at boundary k. Within $[kN, (k+1)N)$, consider each executed opcode. For an opcode that reads no peripheral register, $\delta_{fast} = \delta$ by construction, so the two schedules agree on pc, registers, SREG, and data space, and hence on the next opcode. For an opcode reading a peripheral register r, monotonicity gives one of three cases: clause (1) is excluded since the read occurs; under clause (2) the lazy schedule computes f_r via Algorithm 2, which by Theorem 2 equals the eager value; under clause (3) both schedules return the value committed at the last boundary. In every case the read returns the same byte, so the opcode sequence and the post-states coincide (Sub-claim 1). Consequently the two schedules reach $(k+1)N$ agreeing on all GPIO and PWM-driving registers; since Φ and solve are deterministic functions of those registers, the synchronized circuits agree on every firmware-observable net (Sub-claim 2). The induction closes.

The result is observational, not bit-for-bit: the schedules may differ in internal bookkeeping or in transient peripheral state the firmware never reads. Self-programming (SPM) writes flash and is excluded from the theorem; the implementation forces a synchronization on SPM completion.

4.3 Where the assumption fails

Three situations break monotonicity: an external event (USART RX arrival, a pin-change interrupt) that lands mid-interval; an asynchronous analog output such as a mid-interval PWM duty change; and a self-clearing flag (for example SPI SPIF)

ISSN 3030-3907

whose transient assertion is never seen if the operation both starts and ends within one interval. In each case the firmware-visible timing error is bounded by N cycles, about 32 µs at N = 512 and 16 MHz, which is below the perceptual threshold for interactive use; latency-critical applications can lower N at a proportional throughput cost. An adaptive scheduler that shrinks N during event-dense phases is a natural extension.

5. INTERPOLATED READS

Clause (2) of Definition 5 is realized for the timer registers a delay loop polls most often, the Timer 0 counter TCNT0 and its overflow flag TIFR0, both of which have a closed form in the prescaler state. For a free-running counter with divisor d, base b, and prescaler remainder p_0 captured at synchronization cycle c_0, the eager value at any later cycle c is $\mathrm{TCNT}(c) = (b + \lfloor (p_0 + c - c_0)/d \rfloor) \bmod 256$, and the overflow flag is set whenever that sum reaches 256. Algorithm 2 evaluates these expressions on read and otherwise returns the cached data-space byte; the arithmetic adds about five operations per read, a 1–3 % cost on a timer-polling loop in exchange for eliminating the per-cycle peripheral tick.

Algorithm 2 InterpolatedRead(a, c; b, c_0, d, p_0, TOV_base)
if a = TCNT_addr and d > 0: return $(b + \lfloor (p_0 + c - c_0)/d \rfloor) \bmod 256$
if a = TIFR_addr and d > 0: return TOV_base ∨ $(b + \lfloor (p_0 + c - c_0)/d \rfloor \geq 256)$
return data_space[a]

Theorem 2 (Interpolation exactness). If the firmware writes neither the counter (TCNT_addr), the prescaler configuration (TCCR_addr), nor the flag (TIFR_addr) in $(c_0, c]$, then Algorithm 2 returns the same TCNT(c) and overflow-flag values as the eager schedule.

Proof. Under the no-write hypothesis d, b, and p_0 are constant over $(c_0, c]$. The eager schedule advances the prescaler counter once per cycle and increments the timer whenever it reaches d, so after $c - c_0$ cycles it has produced $\lfloor (p_0 + c - c_0)/d \rfloor$ ticks, exactly Algorithm 2's value. The overflow flag is asserted on the wrapping cycle and is not cleared in the interval by hypothesis, so it equals TOV_base ∨ (counter wrapped), which Algorithm 2 also computes.

This covers the dominant firmware pattern, in which the timer is configured once and then polled, and is exactly the expansion of the standard Arduino delay() macro. When the firmware *does* reconfigure the timer mid-interval, the implementation flushes pending ticks under the old configuration before applying the write, restoring the precondition for the remainder of the interval (Section 7.1 reports the defect that motivated this). The algorithm generalizes to any linear-update counter (Timer 1, Timer 2) but not to event-driven status flags (USART, SPI, TWI), for which the bounded-error treatment of Section 4.3 applies.

ISSN 3030-3907

6. SOLVER AND TOPOLOGY BINDING

The circuit solver runs once per synchronization and must complete in microseconds at the $V \leq 100$ nets typical of educational circuits. Net construction (Algorithm 3) groups pins with union-find: wires and internal shorts union their endpoints, while resistor, diode, and voltage-divider terminals are kept distinct because current flows through them.

Algorithm 3 BuildNets(components, wires)
UF ← UnionFind(); make a set per component pin
for (a,b) in wires: UF.union(a,b)
for each internal short (p,q): UF.union(p,q)
return p ↦ UF.find(p)

Proposition 1. Net building runs in $O((V + E)\cdot\alpha(V))$ for V pins and E wires, with α the inverse Ackermann function. *Proof.* Standard union-find with path compression and union-by-rank [14].

The series solver traces each simple path from a driven net to ground and computes the steady-state current $I = (V_s - \Sigma V_f) / \Sigma R$, treating a zero-resistance driven path as a short and an insufficient source voltage as non-conducting, with PWM duty scaling V_s; per-component voltage and power follow by back-propagation.

Proposition 2. The series solver runs in $O(P\cdot L)$ per update, for P source-to-ground paths of maximum length L. *Proof.* Each path is traversed once with constant work per component.

Proposition 3. The fault detector, a rule-table pass over the solved circuit producing ten diagnostic codes, runs in $O(|E|)$. *Proof.* Each rule inspects each component a constant number of times.

For multi-MCU circuits, rather than asking the learner to declare a named bus, the binder infers it from the wiring: for each board pair it consults a per-chip table mapping peripheral functions (USART TX/RX, SPI MOSI/MISO/SCK/SS, TWI SDA/SCL) to pins and, when the pin-to-net map π makes board A's TX net coincide with board B's RX net and vice versa, installs a byte-forwarding hook between the USARTs; SPI and I²C use the analogous multi-pin match.

Proposition 4. The binder runs in $O(B^2 + W)$ for B boards and W wires. *Proof.* Computing π is $O(W)$; the pairwise pass is $O(B^2)$ with a constant number of table lookups per pair.

For $B \leq 10$ in practice the binder runs in microseconds per topology change. The insight is that the drawn topology already determines the multi-MCU protocol structure, eliminating the bus-configuration step that comparable platforms require; we found no peer-reviewed prior art for this in the simulator setting.

7. EMPIRICAL EVALUATION

ISSN 3030-3907

We validate the formal claims on real firmware in the Logiclabuz implementation; all harnesses and raw data are in research/bench/.

7.1 Drift validation

The drift benchmark runs the platform at $N \in \{1, 64, 512, 4096\}$ on six AVR programs and diffs the lazy run against the $N = 1$ eager reference, both in architectural state (67 fields at ten cycle checkpoints) and in the timing of peripheral events. Table 1 shows that every monotone-read workload exhibits *zero* state drift at all N, including $N = 4096$; the timer_poll workload exercises the interpolation path, so its zero drift confirms Theorem 2 as well as Theorem 1.

Table 1.

Maximum state drift (differing fields out of 67) on monotone-read workloads.

Workload	N = 1	N = 64	N = 512	N = 4096
cpu_loop	0	0	0	0
fib	0	0	0	0
timer_poll	0	0	0	0
usart_status_poll	0	0	0	0

Table 2 reports event drift on two event-driven workloads. At every N the maximum observed drift is strictly below N, confirming the bound of Section 4.3, with mean drift near N/4. The events lost rows record reference events the lazy run cannot match; these arise from the AVR's single-bit overflow flag, which coalesces two overflows that occur between firmware reads, behavior the lazy schedule faithfully inherits rather than a scheduler defect.

Table 2.

Event drift (CPU cycles) on event-driven workloads.

Workload	Metric	N = 1	N = 64	N = 512	N = 4096
timer_isr_jitter	max drift	0	76	258	3521
timer_isr_jitter	mean drift	0	37.9	128.2	302.2
timer_isr_jitter	events lost	0	0	223	363
usart_tx_burst	max drift	0	77	79	2476
usart_tx_burst	mean drift	0	38.6	39.9	200.6
usart_tx_burst	events lost	0	322	1130	1233

Preparing this benchmark surfaced a real defect: a persistent 1-LSB drift on timer_poll at $N \geq 512$, caused by tickBatch retroactively applying a new prescaler to cycles that elapsed before an OUT TCCR0B completed. Flushing pending ticks under the old configuration before any mid-interval timer write restored the zero-drift result of Table 1, demonstrating that the benchmark detects genuine solver bugs.

ISSN 3030-3907

7.2 Throughput and solver cost.

The companion paper [1] reports the full throughput sweep; Figure 2 plots the normalized speedup S(N) it yields. Disabling the scheduler (N = 1) drops throughput far below avr8js, so the scheduler is the load-bearing optimization; raising N to 512 gives an 11.7–43.6× speedup, and the marginal gain from 512 to 4096 is at most 10 %. The empirical knee at N ≈ 512 matches the cost-model prediction of Section 4.1, confirming the model on real firmware.

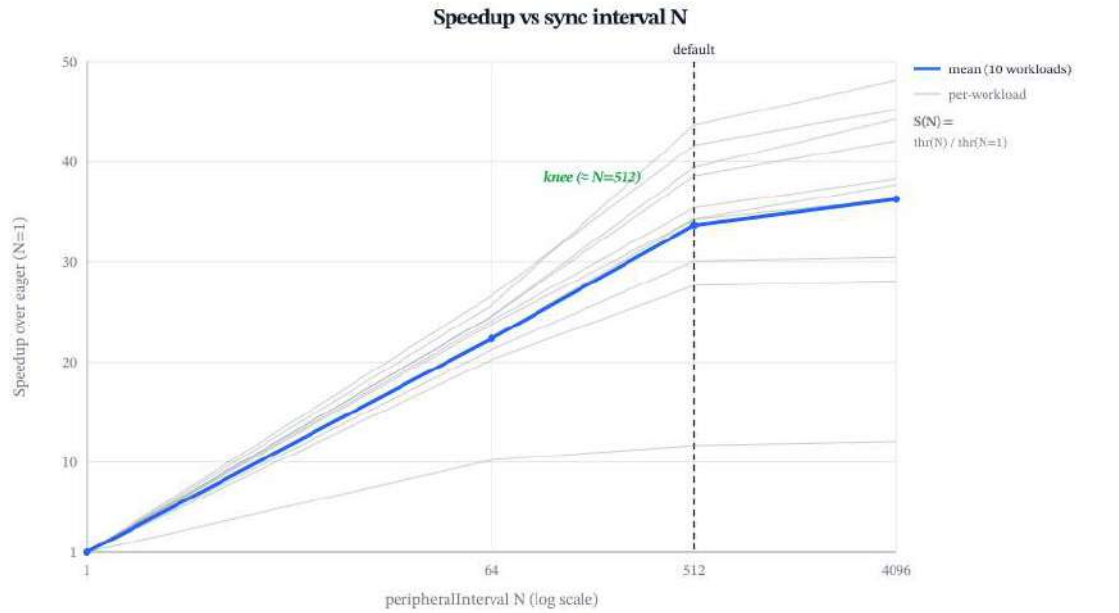


Figure 2. Normalized speedup S(N) over the eager baseline for the ten workloads, with the mean in bold. The curve rises steeply to N = 64, bends by N = 512, and is nearly flat thereafter — the knee predicted by the cost model.

The solver complexity bounds (Propositions 1–3) hold comfortably in practice: on the largest bundled circuit (12 components, ~28 nets, 6 paths) net building, series solving, and fault detection take 0.14, 0.27, and 0.05 ms respectively, well within the per-synchronization budget at N = 512.

7.3 Threats to validity.

The drift workloads are six small programs and do not yet exercise USART RX or external pin-change interrupts, the cases Section 4.3 predicts as worst-case bounded errors; the proven N-cycle bound suggests no qualitative surprise. Throughput is measured on a single host, where V8 JIT noise makes absolute rates host-dependent (the drift results, being deterministic, are not affected). The usart_status_poll workload reads a statically constant register, so a dynamic non-interpolated peripheral workload should be added to exercise the staleness regime directly. A formal solver-scaling sweep over synthetic circuits up to thousands of nets is left to future work.

8. DISCUSSION AND CONCLUSION.

ISSN 3030-3907

We modeled MCU–circuit co-simulation as a synchronous product of a timed transition system and a constraint-graph solver and gave a two-tier scheduler whose lazy schedule is observationally equivalent to the eager reference under a peripheral-monotonicity condition (Theorem 1), with a closed-form interpolation that restores exactness for the timer registers that would otherwise drift (Theorem 2). A cost model predicts the throughput knee that fixes the default interval, and a circuit solver and topology binder meet the per-synchronization budget at proven complexity bounds. The drift benchmark confirms zero observable state drift on monotone-read workloads up to N = 4096 and event drift bounded by N, while the companion platform realizes an 11.7–43.6× speedup at no observable cost.

The formal model is specialized to the MCU-and-DC-circuit setting: the monotonicity condition and the interpolation generalize to any linear-update peripheral but not to event-driven status flags, and the DC solver is not a substitute for transient analog simulation. Against general co-simulation standards such as FMI [10], whose per-step overhead is amortized over millisecond-scale physics, the contribution is a per-cycle, microsecond-budget scheduler with an explicit correctness condition. Future work includes an adaptive interval driven by the cost model, piecewise interpolation for non-monotone but predictable peripherals, and extension to non-AVR targets, for which only the instruction-decoding layer changes while the scheduler, interpolation, and solver carry over unchanged.

Acknowledgements

The author thanks the early Logiclabuz users whose firmware patterns motivated the interpolation algorithm, and the maintainers of simavr and avr8js, whose reference implementations informed the formal model.

REFERENCES

1. A. Ergashev, “Logiclabuz: A browser-native co-simulation platform for the virtualization of multi-MCU robotic systems.”.
2. J.-M. Pollet, “Simavr: A lean, mean and hackable AVR simulator for linux & co.” https://github.com/buserror/simavr.
3. U. Wasserman, “avr8js: AVR 8-bit simulator written in TypeScript.” https://github.com/wokwi/avr8js.
4. N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, “GVSoC: A highly configurable, fast and accurate full-platform simulator for RISC-V based IoT processors,” in *Proceedings of the 39th IEEE international conference on computer design (ICCD)*, 2021, pp. 409–416. doi: 10.1109/ICCD53106.2021.00071.
5. C. Lei *et al.*, “FlexEmu: Towards flexible MCU peripheral emulation,” in *Proceedings of the 2025 ACM SIGSAC conference on computer and communications security (CCS)*, 2025. doi: 10.1145/3719027.3765086.

6. W. Zhou, L. Zhang, L. Guan, P. Liu, and Y. Zhang, "What your firmware tells you is not how you should emulate it: A specification-guided approach for firmware emulation," in *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security (CCS)*, 2022. doi: 10.1145/3548606.3559386.
7. F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proceedings of the USENIX annual technical conference*, 2005, pp. 41–46.
8. N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011, doi: 10.1145/2024716.2024718.
9. L. W. Nagel, "SPICE: A computer program to simulate semiconductor circuits," University of California, Berkeley, Memorandum No. ERL-M520, 1975.
10. T. Blochwitz *et al.*, "The functional mockup interface for tool-independent exchange of simulation models," in *Proceedings of the 8th international modelica conference*, in Linköping electronic conference proceedings no. 63. Dresden, Germany: Linköping University Electronic Press, 2011, pp. 105–114. doi: 10.3384/ecp11063105.
11. M. A. Hamdi *et al.*, "Integrating SystemC-AMS power modeling with a RISC-V ISS for virtual prototyping of battery-operated embedded devices," in *Proceedings of the 21st ACM international conference on computing frontiers: Workshops and special sessions (CF companion)*, 2024. doi: 10.1145/3637543.3652873.
12. J. Eker, J. W. Janneck, E. A. Lee, *et al.*, "Taming heterogeneity — the ptolemy approach," in *Proceedings of the IEEE*, 2003, pp. 127–144.
13. R. M. Fujimoto, *Parallel and distributed simulation systems*. Wiley, 1999.
14. R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM*, vol. 22, no. 2, pp. 215–225, 1975.

ISSN 3030-3907