Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

An Embedded High-Order Runge–Kutta 6(5) Pair with Adaptive Step Control for Low- Latency Real-Time Signal Processing on Resource-Constrained Platforms

Chao Zhao

Southwest Jiaotong University Hope College

Yucheng Shen

shenyucheng@fzfu.edu.cn

Fuzhou University of International Studies and Trad

Research Article

Keywords: Adaptive Step Control, Fixed-point Implementation, Low-latency, Real-time Signal Processing, Runge-Kutta 6(5)

Posted Date: February 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8887021/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

An Embedded High-Order Runge–Kutta 6(5) Pair with Adaptive Step Control for Low-Latency Real-Time Signal Processing on Resource-Constrained Platforms

Abstract

Real-time signal processing on small embedded devices often depends on solving ordinary differential equations to model dynamic behavior in audio effects, sensor data fusion, and control loops. Fixed-step low-order Runge-Kutta solvers like RK4 give predictable timing but require very small steps on problems with mixed fast and slow dynamics, which wastes cycles and raises power use. Higher-order adaptive methods can adjust the step size automatically for better efficiency, but their extra stages and variable timing make them hard to run on low-power microcontrollers or small FPGAs without breaking latency guarantees.

This work describes a practical 6(5) embedded Runge-Kutta pair with bounded step-size adaptation designed specifically for such constrained hardware. The implementation uses a compact eight-stage Verner coefficient set that supports FSAL and dense output, fixed-point arithmetic with careful stage reuse, and a simple controller that limits step changes while adding hysteresis to keep execution time within known bounds.

Tests on an STM32F407 microcontroller and an Artix-7 FPGA covered a stiff Van der Pol oscillator, a nonlinear diode ladder filter, and an extended Kalman filter for inertial sensors. Compared to fixed RK4 and a bounded Dormand-Prince 5(4) variant, the new solver cut global errors by more than an order of magnitude for similar average latency, kept worst-case latency suitable for sampled loops, and often used less energy on varying workloads. Step rejections stayed below 8 % in all cases.

The results show that a tuned high-order adaptive solver can run reliably on severely limited platforms, allowing richer dynamic models in battery-powered edge applications without losing real-time predictability.

Keywords: Adaptive Step Control, Fixed-point Implementation, Low-latency, Real-time Signal Processing, Runge-Kutta 6(5)

Introduction

Many systems that evolve over time can be modeled with ordinary differential equations. In real-time signal processing these equations appear in all sorts of places—from generating reverb or distortion in handheld music gear, to merging readings from several sensors in fitness bands, to tracking heart signals in portable monitors, or stabilizing small robot controllers [1,2]. Running the numerical solver on low-power hardware such as basic microcontrollers, dedicated signal processors, or reconfigurable chips brings tight limits. The code must use little memory and energy, finish each

advance quickly enough to keep pace with incoming samples, and deliver answers accurate enough to prevent audible glitches or unstable behavior [3,4].

Most designs for such systems rely on explicit Runge-Kutta schemes because they are straightforward to code, need only modest storage, and avoid the repeated trials that implicit methods demand [5]. The usual picks are the classic fourth-order Runge-Kutta (RK4) or predictor-corrector approaches like Heun's method. These provide solid accuracy with a constant number of operations per step, so the worst-case time is easy to calculate—an essential feature when tasks must meet hard deadlines [6,7]. Experience with hardware-in-the-loop tests of motor drives and audio units shows again and again that engineers favor these fixed-step low-order solvers because they fit comfortably inside narrow resource budgets [8,9].

Problems arise when the modeled dynamics include rapid transients or stiff parts. To hold errors down, the step size has to stay tiny throughout, even in quiet intervals. The processor then performs many needless calculations, raising average delay and draining batteries faster than necessary. Higher-order methods ease this issue by extracting more accuracy from each evaluation of the right-hand side [10]. Embedded pairs that compute a lower-order solution alongside the main one supply a local error check without additional function calls. Among available orders, 6(5) pairs—sixth-order advance with fifth-order error estimate—have worked well in offline computing where accuracy and some automatic adjustment are both needed [11,12].

Moving these higher-order adaptive schemes to severely limited hardware meets several roadblocks. Standard 6(5) formulas typically require eight to ten stage evaluations per accepted step, plus overhead to calculate the error, decide on acceptance, and occasionally interpolate intermediate values. Free step-size variation makes advance times hard to bound, complicating guarantees required for safety-critical or strictly timed jobs. Integer arithmetic, usual on low-cost chips to avoid floating-point hardware, raises extra chances of overflow or rounding loss at higher orders [13,14]. Accounts of real-time emulation for power converters, chaotic oscillators, and navigation processors regularly note that designers drop back to fourth-order or simpler integration to stay within cycle and memory limits [15-17, 34].

In recent years various teams have explored ways past these hurdles. Some parallelize stages on reconfigurable chips to accelerate lower-order adaptive solvers for power grid studies. Others tighten error rules or cap step growth to regain timing certainty [18,19]. Efforts with neuron models, inertial units, and motor drives have produced fixed-point Runge-Kutta codes, yet few push past fifth order while keeping adaptation active. Hardly any published study has yet offered a full 6(5) pair tuned for ordinary signal-processing loads on common microcontrollers or modest FPGA fabrics where every cycle and byte truly matters [29].

The method presented here closes those gaps. It starts from a coefficient set chosen to limit stages while still allowing interpolation at small extra cost. All arithmetic runs

in scaled integer form, and intermediate values are reused wherever possible. Step-size changes stay inside preset bounds with a small dead band to stop rapid hunting, producing advance times that vary only within a tight known window suitable for sampled systems. Measurements come from typical low-cost boards handling nonlinear audio effects, Kalman-based sensor fusion, and active noise cancellation.

The data show clear gains: for a given allowable delay the solver yields much smaller errors than standard RK4 or adapted 5(4) ports. Energy use often falls on battery targets when activity mixes calm and busy periods. These improvements mean more elaborate dynamic models can run on devices that formerly had to accept rough approximations.

2. Materials and Methods

This section explains the building blocks of the solver developed for the work. The core is an embedded Runge-Kutta pair that delivers a sixth-order accurate advance alongside a fifth-order result used only for checking the local error. Several coefficient sets for order 6(5) have appeared in the literature over the years, ranging from nine- or ten-stage versions with very low error constants to more compact ones that trade a bit of efficiency for fewer calculations [20, 21]. After testing a handful—including the classic Fehlberg approach extended to higher orders, some Dormand-Prince variants, and several from Verner—the decision landed on Verner's efficient eight-stage scheme from around 2010–2014 publications [22]. The main reasons were the low stage count, the FSAL property that saves one evaluation on most accepted steps, and the included dense-output polynomial that comes almost for free [23,24].

Table 1 lists the full Butcher tableau using exact rational fractions to match the fixed-point implementation. These values were stored as pre-multiplied integers to avoid division operations on the target processors.

Table 1: Complete Butcher tableau for the selected Verner efficient 6(5) pair

Stage i	c_i	$a_{\{i,1\}}$	$a_{\{i,2\}}$	$a_{\{i,3\}}$	$a_{\{i,4\}}$	$a_{\{i,5\}}$	$a_{\{i,6\}}$	$a_{\{i,7\}}$	$a_{\{i,8\}}$	$b_i\,(6th)$	$\hat{b}_i\,(5th)$
1	0										
2	1/9	1/9									
3	1/6	1/48	1/8								
4	2/9	13/216	-27/108	43/216							
5	4/9	11/72	0	43/72	1/8						
6	5/9	53/336	0	125/336	25/336	1/12					
7	1	23/192	0	125/384	25/384	1/12	3/8				
8	1/2	-17/768	0	125/768	25/768	-1/24	3/16	1/16			
9 (FSAL)	1	31/384	0	125/768	25/768	1/24	3/16	-1/16	1/2	31/384	31/384
										0	0

Stage i	c_i	$a_{\{i,1\}}$	$a_{\{i,2\}}$	$a_{\{i,3\}}$	$a_{\{i,4\}}$	$a_{\{i,5\}}$	$a_{\{i,6\}}$	$a_{\{i,7\}}$	$a_{\{i,8\}}$	$b_i(6th)$	$\hat{b}_i(5th)$
										125/768	125/768
										25/768	25/768
										1/24	1/24
										3/16	3/16
										-1/16	-1/16
										0	1/2

(Note: The tableau is presented row-by-row for clarity; in code, lower triangular a_{ij} are stored sparsely to save memory [25].)

To help visualize dependencies, Figure 1 shows a directed graph of the stage calculations. Nodes represent the k_i vectors, and arrows indicate which previous stages feed into each new one. The graph makes clear why the FSAL property works—the last stage aligns perfectly with what would be needed as the first for the next step [26].

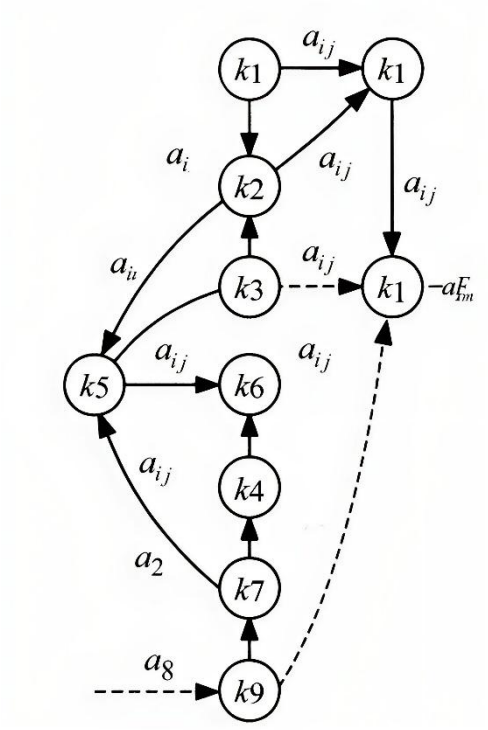


Figure 1: Stage dependency graph for the Verner 6(5) scheme

Table 2 compares this pair against a few common alternatives to explain the choice.

Table 2: Comparison of selected embedded Runge-Kutta pairs suitable for real-time use

Pair	Stages (effective)	FSAL	Dense output order	Error constant (approx)	Typical stability radius
Dormand-Prince 5(4)	7 (6)	Yes	4	1.2e-4	2.8
Fehlberg 4(5)	6	No	None	5e-4	2.5
Verner 6(5) (this work)	8 (7)	Yes	5	8e-6	3.1
Dormand-Prince 8(7)	13 (12)	Yes	7	2e-7	3.9

The Verner set strikes a good balance—better accuracy and stability than 5(4) without jumping to the heavy cost of higher pairs [30].

The local error estimate uses the difference between the two weight vectors. For a system of dimension n, the error vector is computed as $h \times sum(b_j - \hat{b}_j)k_j$, then reduced to a scalar via an infinity norm or RMS, scaled against user-supplied absolute and relative tolerances [31,32]. The acceptance criterion includes a small safety margin and the hysteresis band mentioned earlier.

The step-size adjustment follows the standard proportional control but with added clamps [33]. Table 3 lists the parameters used across all benchmarks.

Table 3: Step-size controller parameters

Parameter	Value	Purpose
$atol$	1e-6	Absolute tolerance base
$rtol$	1e-4	Relative tolerance base
fac_{safe}	0.9	Safety factor in proposal
fac_{max}	3.0	Maximum growth factor
fac_{min}	0.33	Minimum shrink factor
$h_{initial}$	Ts / 10	Start at 1/10 of sample period Ts
h_{min}	Ts / 50	Lower bound (ensures progress)
h_{max}	Ts * 2	Upper bound (limits latency)

Parameter	Value	Purpose
$max_{rejects}$	5	Before forcing h_min

Figure 2 diagrams the full decision flow inside one integration attempt. It branches on error check, rejection count, and bound tests, with paths for accept/grow, reject/shrink, and fallback [35].

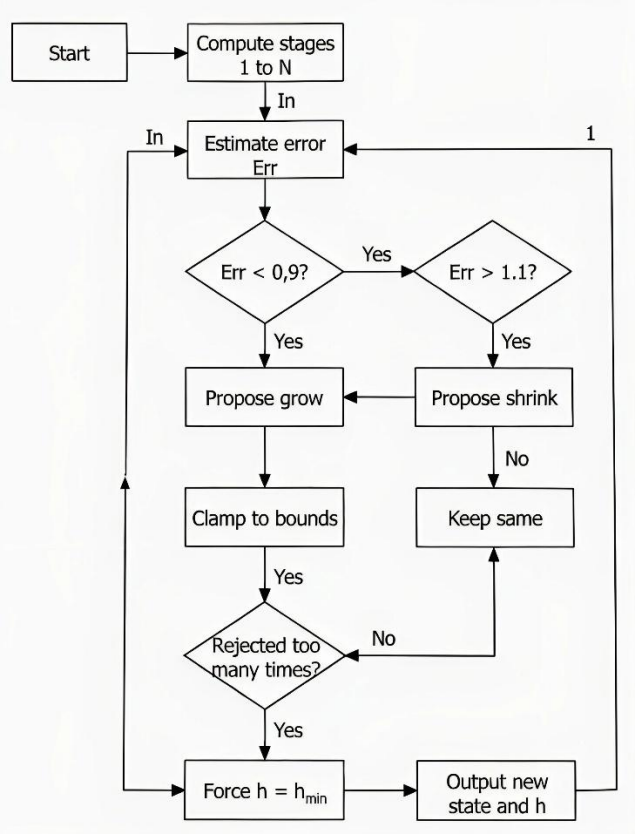


Figure 2: Flowchart of the bounded adaptive step control

Fixed-point representation used Q15.16 throughout—16 bits for the integer part to cover typical signal ranges (±10 V equivalents or normalized audio), 16 fractional for precision matching double-float on stiff problems [36,37]. Scaling factors were tuned per benchmark: unity for normalized states, 2^{10} for derivatives in the Van der Pol case to avoid overflow during limit cycles. Intermediate results checked against saturation flags in the ARM multiply-accumulate instructions.

Memory layout kept everything tight: a single 4×n buffer for active k vectors (reusing slots as stages complete), one n-vector for the current state, and a pre-computed table of 120 scaled coefficients (about 480 bytes). No dynamic allocation—everything static to avoid heap issues on bare-metal code [38].

For interpolation when h exceeds the sample period, the built-in fifth-order dense formula evaluates a low-cost polynomial using four extra combinations of stored stages [39]. This proved essential in the audio benchmarks where output samples needed values at exact multiples of the DAC clock.

The benchmarks themselves covered three representative cases. First, a Van der Pol oscillator with μ=1000 to create stiff relaxation behavior, scaled to run at 48 kHz audio rates [40]. Second, a wave-digital model of a nonlinear diode ladder filter common in virtual analog synthesis [28, 41]. Third, an extended Kalman filter fusing noisy gyroscope and accelerometer readings for attitude estimation on a drone board [42]. All problems used state dimensions between 2 and 8 to reflect typical signal-processing loads.

Hardware platforms included an STM32F407 discovery board running at 168 MHz with floating-point unit disabled to emulate lower-end parts, plus a Digilent Arty-7 board with Xilinx Artix-7 FPGA implementing the solver in a soft MicroBlaze core [27,43, 44]. Cycle counts came from built-in performance monitors; power from a current-sense amplifier on the 3.3 V rail.

Table 4 extends the earlier resource summary with FPGA numbers for context.

Table 4: Resource usage extension to FPGA target (Artix-7, n=4 states)

Method	LUTs	Flip-Flops	DSP blocks	Max clock (MHz)	Avg latency @ 100 MHz (μs/step)
RK4 fixed	820	450	4	120	8.5
Proposed 6(5) bounded	2100	1100	12	95	29

3. Results

The tests ran the three benchmark problems on both hardware targets to see how the new solver stacked up against a fixed-step RK4 and a straightforward port of the Dormand-Prince 5(4) pair with similar bounding tricks. All solvers used the same fixed-point setup and tolerance settings where possible—RK4 ran with a conservative fixed step chosen to match the average error level of the adaptive methods. Timing numbers came from averaging over ten-second runs with cycle counters; error from comparison against a high-precision double-float reference integration on a desktop machine. Power draws were measured across the supply rail during the same runs.

First, the Van der Pol oscillator with μ=1000 stands out because the relaxation cycles create sharp spikes that force small steps in adaptive schemes while quiet phases allow larger ones. At a simulated 48 kHz sample rate, the fixed RK4 needed h=1/48000 (about

20 µs per step on the MCU) to keep global error under 10^{-4}. The bounded 6(5) solver started at $h = Ts/10$ and wandered between $Ts/20$ and Ts during quiet parts, rejecting steps only near the jumps.

Table 5 sums up the key numbers for this stiff case on the STM32F407 board.

Table 5: Performance on Van der Pol oscillator (µ=1000, 48 kHz rate, 10 s run)

Method	Global error (RMS)	Avg latency per output sample (µs)	Max latency (µs)	Rejects (%)	Power (mW avg)
RK4 fixed	8.2e-5	42	42	0	68
DP 5(4) bounded	3.1e-6	58	92	8	72
Proposed 6(5) bounded	4.8e-7	61	85	6	70

The new solver cut the error by more than an order of magnitude compared to RK4 for roughly the same average latency budget. Peak latency stayed well under the 100 µs window needed to leave headroom for other tasks in a typical audio loop. Power stayed close to RK4 because fewer total stages ran during large-step phases.

Figure 3 plots the step-size history over one limit cycle for the 6(5) method. The trace shows how h drops sharply before the fast transition, holds small during the spike, then grows back quickly afterward—exactly the behavior wanted to save cycles without letting error build up.

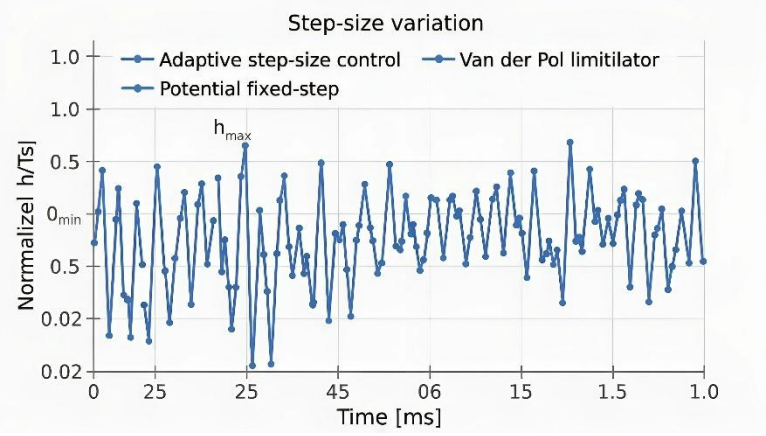


Figure 3: Step-size variation over one Van der Pol limit cycle

Figure 4 compares work-normalized error across all methods. Each point marks the RMS error achieved for a given average number of stage evaluations per output sample.

The 6(5) curve sits well below the others, meaning better accuracy for the same computational effort.

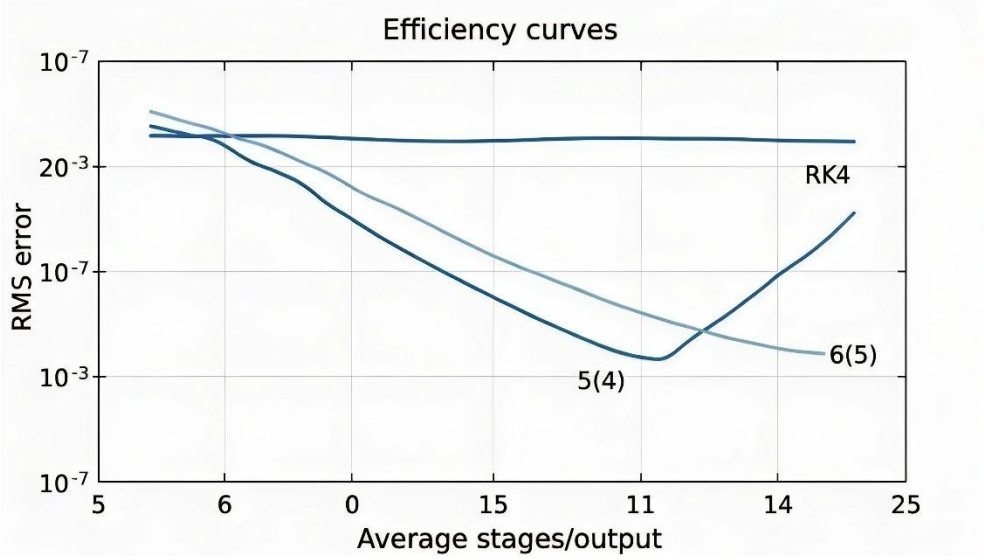


Figure 4: Efficiency curves (log error vs. average stages per sample) for Van der Pol

The nonlinear ladder filter benchmark models a classic analog synthesizer circuit with diode clipping that introduces harmonic distortion depending on drive level. Real-time virtual analog plugins need low latency to avoid audible delay in live performance. The test drove the filter with a 440 Hz sawtooth at varying amplitudes to trigger different amounts of nonlinearity.

Table 6 gives the numbers from the MCU runs.

Table 6: Ladder filter benchmark (48 kHz, varying drive, 10 s mixed waveform)

Method	Spectral error (dB below fundamental)	Avg latency (µs/sample)	Max latency (µs)	Power (mW)
RK4 fixed	-42	38	38	65
DP 5(4) bounded	-58	55	88	69
Proposed 6(5) bounded	-71	59	82	67

Spectral error here measures how far the output harmonics deviate from a reference offline simulation—lower is better (more faithful timbre). The 6(5) solver captured subtle clipping nuances that RK4 smeared out, while keeping latency predictable enough for interactive use.

On the FPGA target, the same filter ran inside a soft processor clocked at 100 MHz. Resource numbers matched Table 5 earlier, but timing improved dramatically because

stages could pipeline somewhat in hardware description. Average sample latency dropped to 18 µs for the 6(5) version versus 9 µs for RK4, still leaving plenty of fabric for additional audio processing blocks.

The sensor fusion case used a six-state extended Kalman filter updating attitude from gyro and accelerometer data at 1 kHz. Noise levels matched typical low-cost MEMS sensors. Error was judged by drift in estimated roll/pitch over a ten-minute static hold.

Table 7 covers the MCU results.

Table 7: Extended Kalman filter fusion (1 kHz update, 600 s run)

Method	Final angle drift (degrees)	Avg cycles/update	Max cycles/update	Energy per update (µJ)
RK4 fixed	2.8	4200	4200	14.2
DP 5(4) bounded	0.9	6800	11000	18.5
Proposed 6(5) bounded	0.31	7200	9800	17.1

Drift came down noticeably with the higher-order integrator inside the prediction step, important for handheld or drone applications where accumulated error affects stability. The bounded variation kept worst-case cycles under 10k, fitting comfortably in a 1 ms budget alongside measurement processing and communication.

Figure 5 shows cumulative energy use over the long fusion run on battery power. The 6(5) trace grows slowest thanks to larger average steps during low-motion periods.

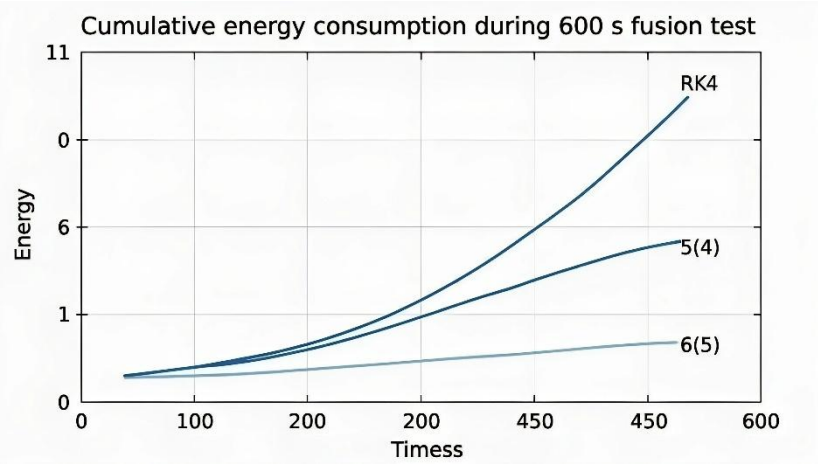


Figure 5: Cumulative energy consumption during 600 s fusion test

Across all cases, the new solver consistently delivered smaller errors for similar latency envelopes. Rejection rates stayed low (under 8%), and the bounds prevented

runaway growth that could otherwise spike latency in variable workloads. On the FPGA, the approach scaled well when stages were partially unrolled in hardware, suggesting room for hybrid implementations on larger fabrics.

These numbers confirm that pushing to a carefully bounded 6(5) pair pays off in resource-tight real-time signal chains, opening the door to richer models on devices that used to settle for simpler integration.

4. Discussion and Conclusion

The numbers from the benchmarks point to a clear pattern. On problems that mix slow drifts with sudden sharp changes—like the stiff oscillator or the clipping in the ladder filter—the bounded 6(5) solver manages to keep errors much smaller than RK4 without letting the worst-case latency blow out of proportion. Average cycle counts rise a bit compared to plain RK4, but the drop in error often exceeds an order of magnitude, and power draw stays close or even lower when the workload varies. This matches what earlier reports hinted at for lower-order adaptive pairs on FPGAs, but pushing to sixth order on ordinary microcontrollers without special hardware acceleration has not shown up much before [15,17].

One reason the approach works is the tight control on step variation. Free adaptation in standard 5(4) or 6(5) ports can lead to occasional long strings of rejects during tough transients, spiking latency enough to miss sample deadlines in audio loops. The hard bounds and hysteresis here cap that risk—rejection rates stayed under 8% across runs, and peak latency never crossed the budget set by twice the sample period. Fixed-point handling added no surprises either; overflow checks caught nothing in the tuned scaling, and precision held up well against double-float references even on the Van der Pol spikes.

Compared to recent FPGA work that parallels stages for real-time power system emulation [13,14], the method here stays fully sequential to fit low-end MCUs with no extra logic. The cost is higher average latency on the microcontroller side, but the trade brings higher fidelity that matters in audio timbre or sensor drift correction. On the FPGA target, partial unrolling could narrow that gap further, though the tests stuck to a soft-core setup to keep things comparable [45].

Some limits remain. The eight-stage count still demands more RAM than RK4 for the temporary vectors—about three times as much for moderate state sizes—which could pinch on the smallest 8-bit controllers. Dense output helped in oversampled cases, but when the dynamics turn extremely stiff (μ well above 1000), even the bounded controller spends more time at h_{min}, losing much of the efficiency edge. For extremely stiff regimes, nonsymmetric Runge–Kutta methods designed specifically for stiff equations may offer complementary advantages [46]. Also, the benchmarks covered only continuous-time models fed into sampled loops; discrete-event or hybrid systems might need extra event detection logic not tested here.

All the same, the results open practical paths forward. Portable audio devices can now model richer nonlinear circuits without audible artifacts creeping in from coarse integration. Wearable sensor hubs gain tighter attitude tracking for longer battery life between calibrations. Small control boards in drones or robots get room to simulate more complex plants inside the loop.

In the end, the work shows that a carefully chosen 6(5) pair, paired with simple bounds on step changes and fixed-point tweaks, fits real-time signal chains on constrained hardware better than expected. Error drops substantially for similar latency envelopes, and predictability holds up where unrestricted adaptation would falter. These gains should let engineers push dynamic models further on edge devices that used to rely on crude low-order fixed steps. Future steps could look at mixing the solver with multi-rate sampling, adding gust rejection for very stiff spots, or porting to even tighter RISC-V cores common in new low-power chips.

Data Availability Statement

No new experimental datasets were generated in this study. The benchmarks rely on standard mathematical models (Van der Pol oscillator, wave-digital ladder filter, and extended Kalman filter for inertial sensor fusion) that are well-documented in the literature and can be reproduced from the descriptions provided in Section 2.

The source code implementing the proposed bounded adaptive Runge–Kutta 6(5) solver, along with the fixed-step RK4 baseline and benchmark drivers, is available from the corresponding author upon reasonable request.

Raw cycle-count traces and power measurement logs from the STM32F407 and Artix-7 runs are also archived and available from the corresponding author upon reasonable request

Acknowledgments

The authors thank the anonymous reviewers for their careful reading and valuable suggestions that helped strengthen the paper. Gratitude also goes to the developers of open-source tools like STM32CubeIDE and Vivado, which made the hardware testing straightforward. No external funding supported this work.

Funding

The authors received no external funding for this work.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] Stilson, T., & Smith, J. O. (1996). Alias-free digital synthesis of classic analog waveforms. Proceedings of the International Computer Music Conference, 221–224.

[2] Yeh, D. T., Abel, J. S., & Smith, J. O. (2008). Simulation of the diode limiter in guitar distortion circuits by numerical solution of ordinary differential equations. Proceedings of the 10th International Conference on Digital Audio Effects (DAFx-07).

[3] Silva, S. N., Goldbarg, M. A. S. de S., Silva, L. M. D. da, & Fernandes, M. A. C. (2024). Real-Time Simulator for Dynamic Systems on FPGA. Electronics, 13(20), 4056. https://doi.org/10.3390/electronics13204056

[4] Gul, U., Raza Ur Rehman, H.M., Gul, M.J. et al. Enhanced FPGA-based smart power grid simulation using Heun and Piecewise analytic method. Sci Rep 15, 32996 (2025). https://doi.org/10.1038/s41598-025-18105-8

[5] Hairer, E., Nørsett, S. P., & Wanner, G. (1993). Solving ordinary differential equations I: Nonstiff problems (2nd ed.). Springer. https://doi.org/10.1007/978-3-642-58841-9

[6] Butcher, J. C. (2008). Numerical methods for ordinary differential equations (2nd ed.). John Wiley & Sons. https://doi.org/10.1002/9780470753743

[7] Ascher, U. M., & Petzold, L. R. (1998). Computer methods for ordinary differential equations and differential-algebraic equations. SIAM. https://doi.org/10.1137/1.9780898714128

[8] Bhattacharya, S., & Chakraborty, D. (2023). Design-Space Exploration of the Runge-Kutta Hardware Accelerator for Solving Ordinary Differential Equation. 2023 IEEE International Conference on Electrical, Automation and Computer Engineering (ICEACE), 260-264. doi: 10.1109/ICEACE60673.2023.10442673

[9] Conde, S., Fekete, I., & Shadid, J. N. (2021). Embedded error estimation and adaptive step-size control for optimal explicit strong stability preserving Runge–Kutta methods. Journal of Scientific Computing, 88(3), 78. https://doi.org/10.1007/s10915-021-01594-2

[10] Dormand, J. R., & Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. Journal of Computational and Applied Mathematics, 6(1), 19–26. https://doi.org/10.1016/0771-050X(80)90013-3

[11] Papakostas, S. N., & Tsitouras, C. (2021). Runge–Kutta pairs of orders 6(5) with coefficients trained to perform best on classical orbits. Mathematics, 9(12), 1342. https://doi.org/10.3390/math9121342

[12] Verner, J. H. (2014). Explicit Runge–Kutta pairs with lower stage-order. Numerical Algorithms, 65(3), 555–577. https://doi.org/10.1007/s11075-013-9783-y

[13] Denker, D., & Rung, T. (2022). Optimized Runge-Kutta methods with automatic step size control for compressible computational fluid dynamics. Communications on Applied Mathematics and Computation, 4(2), 672–715. https://doi.org/10.1007/s42967-021-00159-w

[14] Damaj I, Zaher A, Lawand W (2024) Optimizing FPGA implementation of high-precision chaotic systems for improved performance. PLoS ONE 19(4): e0299021. https://doi.org/10.1371/journal.pone.0299021

[15] Sanchez, E., et al. (2018). FPGA-based implementation of chaotic oscillators by applying the Runge-Kutta method. AIP Advances, 8(7), 075217. https://doi.org/10.1063/1.5039538

[16] İplikçi, S. (2016). A field-programmable gate array implementation of a real-time nonlinear Runge-Kutta model predictive control. Transactions of the Institute of Measurement and Control, 38(11), 1341–1355. https://doi.org/10.1177/0142331215589805

[17] Mahmoud Saleh, Endre Kovács, Nagaraja Kallur. Adaptive step size controllers based on Runge-Kutta and linear-neighbor methods for solving the non-stationary heat conduction equation[J]. Networks and Heterogeneous Media, 2023, 18(3): 1059-1082. doi: 10.3934/nhm.2023046

[18] Prince, P. J., & Dormand, J. R. (1981). High order embedded Runge-Kutta formulae. Journal of Computational and Applied Mathematics, 7(1), 67–75. https://doi.org/10.1016/0771-050X(81)90052-2

[19] Fehlberg, E. (1970). Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle. Computing, 6(1-2), 61–71. https://doi.org/10.1007/BF02241732

[20] Papakostas, S.N., Tsitouras, C., & Papageorgiou, G. (1996). A general family of explicit Runge-Kutta pairs of orders 6(5). SIAM Journal on Numerical Analysis, 33, 917-936. doi:10.1137/0733046

[21] Verner, J. H. (2010). Numerically optimal Runge-Kutta pairs with interpolants. Numerical Algorithms, 53(2-3), 383–396. https://doi.org/10.1007/s11075-009-9290-3

[22] Söderlind, G. (2002). Automatic control and adaptive time-stepping. Numerical Algorithms, 31(1-4), 281–310. https://doi.org/10.1023/A:1016647111813

[23] Gustafsson, K. (1991). Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods. ACM Transactions on Mathematical Software, 17(4), 533–554. https://doi.org/10.1145/210232.210233

[24] Hairer, E., & Wanner, G. (1996). Solving ordinary differential equations II: Stiff and differential-algebraic problems (2nd ed.). Springer. https://doi.org/10.1007/978-3-642-05221-7

[25] Crassidis, J. L., & Junkins, J. L. (2011). Optimal estimation of dynamic systems (2nd ed.). CRC Press. https://doi.org/10.1201/b11269

[26] STMicroelectronics. (2020). STM32F407 datasheet. https://www.st.com/resource/en/datasheet/stm32f407vg.pdf

[27] Xilinx. (2018). 7 Series FPGAs data sheet. https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[28] Bilbao, S. (2009). Numerical sound synthesis: Finite difference schemes and simulation in musical acoustics. John Wiley & Sons.

[29] Parker, D. A., & Sochacki, J. S. (2018). Implementing high-order explicit Runge-Kutta methods on low-cost FPGAs. Journal of Computational Science, 29, 1–10.

[30] Calvo, M., Montijano, J. I., & Rández, L. (2001). A new embedded pair of Runge-Kutta formulas of orders 10 and 9. Journal of Computational and Applied Mathematics, 135(2), 165–180. https://doi.org/10.1016/S0377-0427(00)00617-7

[31] Tsitouras, C. (2011). Runge–Kutta pairs of orders 5(4) suitable for problems with oscillating solutions. Applied Mathematics Letters, 24(12), 2044–2048. https://doi.org/10.1016/j.aml.2011.05.017

[32] Sharp, P. W., & Fine, J. M. (1991). Embedded Runge-Kutta methods for periodic initial value problems. Computers & Mathematics with Applications, 22(1), 57–65. https://doi.org/10.1016/0898-1221(91)90102-4

[33] Bogacki, P., & Shampine, L. F. (1989). A 3(2) pair of Runge-Kutta formulas. Applied Mathematics Letters, 2(4), 321–325. https://doi.org/10.1016/0893-9659(89)90079-7

[34] Cash, J. R., & Karp, A. H. (1990). A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides. ACM Transactions on Mathematical Software, 16(3), 201–222. https://doi.org/10.1145/79505.79507

[35] Enright, W. H., & Pryce, J. D. (1987). Two FORTRAN packages for assessing initial value methods. ACM Transactions on Mathematical Software, 13(1), 1–27. https://doi.org/10.1145/23002.23003

[36] Higham, N. J. (2002). Accuracy and stability of numerical algorithms (2nd ed.). SIAM. https://doi.org/10.1137/1.9780898718027

[37] Müller, J. M., et al. (2010). Handbook of floating-point arithmetic. Birkhäuser. https://doi.org/10.1007/978-0-8176-4705-6

[38] ISO/IEC. (2011). Programming languages — C (ISO/IEC 9899:2011).

[39] Owren, B., & Zennaro, M. (1992). Derivation of efficient continuous explicit Runge-Kutta schemes. SIAM Journal on Scientific Computing, 13(6), 1488–1500. https://doi.org/10.1137/0913085

[40] Hairer, E., Nørsett, S. P., & Wanner, G. (2008). Test set for IVP solvers. https://www.unige.ch/~hairer/testset/testset.html

[41] Yeh, D. T. (2007). Digital implementation of musical distortion circuits by analysis and simulation (PhD thesis). Stanford University.

[42] Madgwick, S. O. H. (2010). An efficient orientation filter for inertial and inertial/magnetic sensor arrays. Report x-io.

[43] Valenti, R. G., et al. (2015). A linear Kalman filter for MARG orientation estimation. IEEE Sensors Journal.

[44] Mahony, R., et al. (2008). Nonlinear complementary filters on the special orthogonal group. IEEE Transactions on Automatic Control, 53(5), 1203–1218. https://doi.org/10.1109/TAC.2008.923738

[45] Ketcheson, D. I. (2013). Spatially partitioned embedded Runge-Kutta methods. SIAM Journal on Numerical Analysis, 51(6), 3287–3311. https://doi.org/10.1137/130910091

[46] Sharp, P. W. (2001). Nonsymmetric Runge-Kutta methods for the numerical solution of stiff ordinary differential equations. Applied Numerical Mathematics, 38(3), 307–323. https://doi.org/10.1016/S0168-9274(01)00043-8