International Journal of Research and Review in Applied Science, Humanities, and Technology
Vol 1, Issue 2 December 2024 ISSN: 3048-975X
https://ijrasht.com/

Low-Power Front-End VLSI Design Techniques for Energy-Efficient System-on-Chip Architectures

Kavyashree Raveendranath, Digital Design Engineer, Meta, United States of America kavs2803@gmail.com

Swamy TN, Assistant Professor, ECE Department, Dr. Ambedkar Institute of Technology, Bangalore, Karnataka, India tnswamy.ec@drait.edu.in

Abstract: Power consumption is a primary design constraint in modern System-on-Chip (SoC) architectures. Front-end VLSI techniques (RTL through logic synthesis) are crucial for achieving energy efficiency before committing to costly back-end processes. This paper surveys contemporary front-end low-power techniques, proposes an integrated RTL-level methodology combining activity reduction (clock gating, operand isolation), power-intent driven synthesis (multi-voltage domains and multi-threshold assignment), algorithmic/approximate computing at HLS/RTL, and dynamic voltage and frequency scaling (DVFS) orchestration. We implement a prototype SoC subsystem (RISC-V core + accelerator + memory interface) in an RTL simulation flow and evaluate power, timing, and area using gate-level power estimation. Results from our simulation experiments show cumulative dynamic power reductions of 45–62% and leakage reductions up to 48% relative to a baseline RTL without power optimizations, while keeping critical path timing penalty under 7%. The paper includes practical design guidelines, pseudocode for an automated RTL power-optimizer, and a comparative analysis with recent literature.

Keywords: low-power VLSI, front-end, RTL optimization, clock gating, multi-Vt, DVFS, high-level synthesis, approximate computing, SoC energy efficiency

1. Introduction

Modern SoC designs target mobile, edge, and IoT markets where energy efficiency is as critical as raw performance. While process scaling and back-end techniques (cell libraries, power grids) play a role, significant power savings are unlocked earlier in the design flow — at the front-end (architecture, microarchitecture, RTL, synthesis). Front-end choices determine switching activity, module partitioning into power domains, and opportunities for algorithmic reduction of operations. This paper focuses on front-end VLSI design techniques and proposes a practical methodology to integrate multiple orthogonal techniques to minimize total power (dynamic + leakage) with minimal impact on area and timing.

Key front-end levers explored:

- Architectural: DVFS, power domains, power/clock gating policies
- RTL: clock gating, operand isolation, register file banking, data-path width reduction, state retention
- Logic synthesis: multi-Vt assignment, threshold gating (MTCMOS), technology mapping tuned for switching reduction
- HLS/algorithmic: approximate computing, pruning, bit-width optimization

We benchmark our approach on a representative SoC subsystem and report simulation results that demonstrate effective energy savings while preserving functional correctness and quality-of-service (QoS).

2. Motivation and Problem Statement

As feature sizes shrink, static/leakage power becomes significant alongside dynamic power. Designers must satisfy constraints across power, performance and area (PPA). Front-end design decisions often carry irreversible cost in later stages; therefore, an automated, front-end centric low-power methodology that is EDA-tool friendly is necessary.

Problem statement: Design a front-end methodology that (1) reduces dynamic power via switching/activity reduction and clock domain management, (2) reduces leakage through multi-threshold and power domain isolation, (3) integrates algorithmic approximations when acceptable, and (4) provides an automated RTL flow that yields measurable energy savings with bounded timing impact.

3. Literature Review

(Selected contemporary works and surveys are summarized to highlight where the proposed methodology fits.)

1. **Clock gating & RTL power optimizations** — Clock gating remains the most effective RTL technique to reduce dynamic power in sequential circuits; recent surveys and industry practices emphasize automated clock gating insertion and power-aware RTL coding styles as essential for early power reduction.

2. **DVFS and run-time management** — DVFS is broadly used to trade performance for dynamic energy savings; recent studies quantify DVFS benefits in embedded systems and propose fine-grain DVFS controlled by workload prediction.
3. **Near-threshold computing** — Near-threshold operation reduces energy per operation but increases sensitivity to variability; architecture and microarchitectural strategies (voltage islands, resilient logic) are proposed to manage this tradeoff.
4. **Multi-threshold CMOS (MTCMOS) and power gating** — MTCMOS and sleep transistor insertion reduce leakage during idle periods; optimization strategies and cluster assignment heuristics have been explored for effective leakage reduction.
5. **HLS and approximate computing** — High-level synthesis combined with approximation techniques (reduced precision, loop perforation) provides sizeable energy savings when error tolerances exist. Recent work shows approximation at HLS can be integrated with RTL optimizations.
6. **Power-aware EDA flows** — Major EDA vendors provide power-aware synthesis and signoff flows emphasizing power intent (UPF/CPF), power-aware placement and extraction — these are vital for front-end decisions to carry through to physical design.

Gap identified: While each technique is effective individually, there is limited literature providing an integrated front-end RTL flow that orchestrates HLS approximations, RTL gating, multi-voltage assignment, and DVFS control with automated heuristics — especially focusing on SoC subsystems where multiple IPs interact.

4. Proposed Methodology

4.1 Overview

We propose a **Front-End Power-Intent Driven RTL Optimization Flow (FPD-RTL)** that integrates the following modules:
1. **Architectural partitioning module** — Suggests voltage islands and DVFS candidate domains based on workload and performance constraints.
2. **RTL low-power transformations** — Automated insertion of clock gating, operand isolation, and state retention; register bank restructuring and bit-width pruning.
3. **Power-intent and synthesis orchestration** — Generate UPF, apply multi-Vt assignment heuristics, and instruct synthesis for power-aware optimization.
4. **Approximation advisor (optional)** — At HLS/algorithmic level suggests safe approximations (reduced bit-width, error-tolerant operators).
5. **DVFS controller (software/hardware co-design)** — A runtime policy that adapts voltage/frequency to workload, scheduled by the OS or a hardware monitor.

The flow is largely automated, starting from an annotated RTL (pragmas to mark QoS and approximate regions) and producing power-intent files and RTL patches.

4.2 Detailed Components

4.2.1 Architectural Partitioning
- Analyze data-path usage and identify idle periods per IP using cycle-accurate simulation traces.
- Create voltage islands for clusters with independent performance deadlines.
- Candidate DVFS domains are chosen where performance slack exists. Use utilization threshold U_th (e.g., 60%) to mark a domain as DVFS-eligible.

4.2.2 RTL Transformations
- **Clock gating**: Insert gating on register banks where enable conditions are static or predictable. Use both automatic RTL insertion (via synthesis tool) and manual safe gating for custom datapaths.
- **Operand isolation**: For combinational blocks guarded by register outputs, add isolation logic to prevent switching when input operands are invalid/unchanged.
- **Register banking & width pruning**: Split large register file accesses to reduce switched bits; perform bit-width analysis to detect MSB bits with low entropy and safely reduce width.

4.2.3 Power-Intent Driven Synthesis
- Produce UPF with defined power domains and retention cells.

- Apply multi-Vt assignment: assign high-Vt cells to non-timing critical paths, low-Vt to timing critical paths. Use an iterative heuristic based on slack distribution.

4.2.4 HLS Approximation Advisor

- For accelerators processing tolerant data (e.g., DSP, ML), propose reduced precision or approximate operators (adders with truncated LSBs).
- Estimate impact on QoS using a quality metric function Q = f(error_metric); include designer thresholds for Q_min.

4.2.5 DVFS Controller

- Hardware monitor collects performance counters and power estimates; a small policy engine selects P-states based on predicted workload.
- Use reactive DVFS with a safety margin to avoid frequent up/down oscillations.

4.3 Automation Algorithm (High Level)

We include a pseudocode for the RTL power optimizer (Section 7). The flow takes annotated RTL + constraints and outputs optimized RTL + UPF.

5. Mathematical Models & Power Estimation

5.1 Dynamic Power

Dynamic switching power:
P_{dyn} = \sum_i \alpha_i C_i V_{dd}^2 f
where \alpha_i is switching activity for net/cell i, C_i is capacitance, V_{dd} supply, and f clock frequency.

Aim: reduce \alpha_i (clock/operand gating), lower V_{dd} (DVFS), and reduce C_i (bit-width pruning, RTL restructuring).

5.2 Leakage Power

Leakage (static) is modeled as:
P_{leak} = \sum_i I_{leak,i} \cdot V_{dd}
MTCMOS reduces I_{leak} in idle blocks by switching them to high-Vt or cutting off via sleep transistors.

5.3 Multi-Objective Cost Function

We optimize a cost:
J = w_p \cdot \frac{P_{total}}{P_{base}} + w_a \cdot \frac{A}{A_{base}} + w_t \cdot \frac{T_{crit}}{T_{base}}
where P_{total}=P_{dyn}+P_{leak}, A area, T_{crit} critical path delay; weights reflect designer priorities (e.g., prioritize power: w_p=0.7, w_a=0.15, w_t=0.15).

6. Experimental Setup

6.1 Benchmark SoC Subsystem

We implemented a representative SoC subsystem in RTL including:

- 32-bit RISC-V core (RV32I subset)
- ML accelerator (convolutional primitive)
- DMA + AXI-Lite memory interface
- Peripheral timer and UART

(Modules coded in SystemVerilog; HLS used for accelerator in C++ when evaluating approximate variants.)

6.2 Flow & Tools (simulation / estimation)

- RTL simulation: ModelSim/Verilog for functional validation.

- Synthesis: Synopsys Design Compiler (power-aware synthesis with UPF).
- Power estimation: PrimeTime PX (or equivalent gate-level power estimation); when gate-level libraries not available, we used back-annotated gate estimates combined with switching activity from RTL simulation to estimate dynamic power.
- Technology corner: Generic 28nm CMOS cell library for evaluation (to keep results technology-representative).
- DVFS simulation: modeled by scaling V_{dd} and frequency following typical delay vs voltage scaling curves.

Note: The actual physical layout and post-layout parasitics were not produced in this front-end evaluation; power numbers are derived from gate-level power estimation using synthesis libraries and simulated switching activities. Results are therefore indicative but representative of front-end benefits — precise silicon numbers would require full backend signoff.

6.3 Baseline vs Optimized Configurations

- **Baseline**: RTL coded without explicit power intent, no clock gating pragmas, full precision in accelerators, single voltage domain.
- **Optimized Flow (FPD-RTL)**: All modules annotated; automated clock gating and operand isolation enabled; multi-Vt assignment applied; accelerator with two approximation levels; DVFS domains for compute vs IO.

7. Algorithm / Pseudocode

7.1 RTL Power Optimizer — High Level Pseudocode

Input: rtl_sources, constraints (timing/area/QoS), annotations (approx_regions)

Output: optimized_rtl, power_intent_upf, synthesis_directives

```
1: simulate_cycle_accurate(rtl_sources) -> activity_traces
2: domain_candidates <- identify_voltage_islands(activity_traces, constraints)
3: for each module in rtl_sources:
4:     if module.annotated_approx:
5:         propose approx_variants = generate_approx_options(module)
6:         evaluate approx_variants via fast_model -> QoS_estimate
7:         select variant if QoS_estimate >= QoS_threshold
8:     end if
9:     gating_points <- find_clock_enable_conditions(module, activity_traces)
10:    insert_clock_gates(module, gating_points)
11:    insert_operand_isolation(module, activity_traces)
12:    prune_bitwidths(module, entropy_analysis(activity_traces))
13: end for
14: upf <- create_upf(domain_candidates, retention_policy)
15: synthesis_cmds <- create_synthesis_directives(multi_vt_strategy, upf)
16: run_synthesis(synthesis_cmds) -> netlist, slacks
17: adjust_multi_vt_assignments(netlist, slacks)
18: estimate_power(netlist, activity_traces) -> P_total
19: if P_total improved and timing constraints met: return outputs
20: else: iterate with relaxed parameters or rollback
```

This algorithm iterates to meet timing and QoS while minimizing power.

8. Results

8.1 Metrics Used

- Dynamic power (mW)
- Leakage power (mW)
- Total energy per task (mJ)

- Area overhead (% relative to baseline)
- Critical path delay / timing slack (ns)
- QoS metric for accelerator (e.g., top-1 accuracy drop for ML use case)

8.2 Key Quantitative Results (simulated / estimated)

All numbers are derived from gate-level power estimation with switching activity from RTL simulation and multi-Vt/leakage models from the 28nm library.

Configuration	Dynamic Power (mW)	Leakage (mW)	Total Power	Area Overhead	Timing Penalty
Baseline (no optim.)	210.4	32.1	242.5	100%	0%
+ Clock gating & operand isolation	138.6	31.2	169.8	101.5%	+1.8%
+ Multi-Vt assignment & power gating	129.8	17.6	147.4	103%	+3.2%
+ HLS approx (level A) + DVFS (conservative)	98.6	17.6	116.2	103.8%	+5.0%
+ HLS approx (level B, aggressive) + DVFS (aggressive)	80.3	16.7	97.0	104.5%	+6.8%

Observations:
- Clock gating + operand isolation yields ~30% dynamic power reduction vs baseline.
- Multi-Vt and power gating reduce leakage by ~45% (from 32.1 mW to 17.6 mW).
- Combined effect with approximation + DVFS can yield up to ~60% dynamic power savings for workloads tolerant to approximation; total power reductions of 45–60% are observed across tested workloads.
- Timing penalty remains within acceptable bounds (≤7% for most aggressive configuration).

8.3 Energy per Task

We measured energy to execute a representative workload (ML inference + memory transfer):
- Baseline energy: 5.12 mJ per task
- FPD-RTL (conservative optim): 2.95 mJ (42.4% reduction)
- FPD-RTL (aggressive approx+DVFS): 1.98 mJ (61.3% reduction)

8.4 QoS Tradeoffs (Accelerator Approximation)
- Approximation Level A: <1.2% accuracy drop (acceptable in many edge applications)
- Approximation Level B: ~3.8% accuracy drop (suitable only if higher error tolerance exists)

9. Discussion
- **Effectiveness of front-end optimizations:** The results underline that front-end techniques yield large cumulative savings even before physical design. Clock gating addresses the dominant dynamic term by lowering switching activity; multi-Vt & power gating address leakage.
- **Approximation + DVFS synergy:** Approximation reduces computational load and switching; DVFS scales Vdd and frequency accordingly, producing multiplicative energy savings.
- **Automation value:** By automating RTL transformations guided by activity traces and QoS constraints, designers can rapidly explore tradeoffs and converge on acceptable PPA points.
- **Limitations:** Our evaluation uses gate-level power estimates without full post-layout parasitics and on a 28nm generic library; absolute numbers will vary with process and layout. Also, frequent DVFS transitions can cause overhead; policy tuning is necessary.
- **Comparison with literature:** The trends align with recent surveys emphasizing RTL power practices, DVFS benefits, and HLS approximation potentials.

10. Practical Implementation Guidelines
1. **Early power intent (UPF) adoption:** Define domains early in RTL and ensure EDA tools keep power files through flow.
2. **Use automatic and manual clock gating:** Let synthesis insert safe gates but manually add custom gating where activity patterns are known.

3. **Profile workloads:** Use real traces for activity estimates; idle detection is key for power gating.
4. **Gradual approximation:** Start with conservative approximations and quantify QoS tradeoffs.
5. **Iterative multi-Vt assignment:** Use slack-aware heuristics rather than global assignment.
6. **DVFS hysteresis:** Prevent frequent P-state oscillation by introducing minimum dwell times.

11. Conclusion

This paper presents an integrated front-end methodology (FPD-RTL) combining RTL low-power coding, power-intent driven synthesis, DVFS orchestration and HLS-guided approximation to achieve substantial energy savings in SoC subsystems. Our simulated experiments show total power reductions up to ~60% in aggressive scenarios and 40–50% in conservative configurations, with timing impact under 7% and modest area overhead. Front-end optimization is a high-leverage point for energy efficiency and should be an integral part of SoC design strategy.

12. Future Work

- Validate the flow on multiple technology nodes (7nm, 5nm) and perform full back-end signoff to measure physical-design impacts.
- Integrate machine-learning based exploration for multi-objective optimization (PPA).
- Prototype DVFS hardware controller on FPGA + PMIC to measure real silicon power/latency tradeoff.
- Extend approximate computing advisor with formal quality verification for safety-critical tasks.

References

1. Anis, M., et al. (2002). Dynamic and leakage power reduction in MTCMOS circuits using an automated efficient gate clustering technique. Proceedings of DAC.
2. Hurst, A. P. (2008). Automatic synthesis of clock gating logic with controlled netlist perturbation. DAC/University preprint.
3. Pouiklis, G. (2016). Clock gating methodologies and tools: a survey. (Review article).
4. Ratto, F., et al. (2021). Mutual impact between clock gating and high-level synthesis. Electronics.
5. Zidar, J., Matić, T., Aleksi, I., & Hocenski, Ž. (2024). Dynamic voltage and frequency scaling as a method for reducing energy consumption in ultra-low-power embedded systems. Electronics, 13(5), 826.
6. Leipnitz, M. T., et al. (2019). High-Level Synthesis of approximate designs under real-time constraints. ACM/CFP (HLS + approximation).
7. Carrión-Schäfer, B. (2024). Approximate computing in high-level synthesis. (ACM/Conference/Survey PDF).
8. (DVCon / IEEE) Automation for early detection of X-propagation in power-aware simulation & verification using UPF (IEEE-1801). (Power-intent verification paper; DVCon proceedings).
9. Cadence Design Systems. (2020–2023). Power-aware implementation & low-power design white papers / resources. (Industry guidance on UPF and power flows).
10. Synopsys. (2024). A unified solution for end-to-end low power verification (white paper).
11. Synopsys (Corporate release / product note). PrimeTime PX — power analysis solution (product information / press note).
12. R. Blanton / ACM (Survey). A survey of architectural techniques for near-threshold voltage computing. (Survey article on NTC / near-threshold techniques).
13. Neutzling, A., Mishchenko, A., et al. (2019). Effective logic synthesis for threshold logic circuit design. IEEE Transactions on Computer-Aided Design (TCAD).
14. Kumar, A. K., et al. (2022). Machine learning-based microarchitecture-level power modelling and optimization. (microarchitecture & ML for power paper; methodology using PrimeTime traces).
15. Knechtel, J., et al. (2022). Design-time exploration of voltage switching against power/performance tradeoffs. (Journal article on voltage switching/DVFS tradeoffs).
16. Semiconductor Engineering. (2022–2024). Power-aware design and industry overviews (practical articles & surveys on RTL low-power best practices).
17. Hsieh, A. C., et al. (2008). A functionality-directed clustering technique for low-power MTCMOS. (Related clustering approach / gate clustering literature).
18. (Industry case study) Dialog Semiconductor / technical note and PMIC power optimization guides (practical flow notes for clock gating & synthesis).