LETTER
Snake: An asynchronous pipeline for ultra-low-power applications
Zhi-jiu Zhu1.2a, Yi Yu1,2, Xu Bai1,2, Shu-shan Qiao1.2b, Yong Hei2

Abstract Voltage scaling is an effective technique for ultra-low-power applications. However, PVT variation degrades the robust of traditional synchronous pipelines severely when voltage scales into the sub-threshold region. In this paper, we propose a register-based bundled-data asynchronous pipeline that can operate robustly in sub-threshold, called Snake. By looping the match delay line, the Snake halves the design overhead compared to other asynchronous pipelines. We also propose a practical asynchronous design methodology which is compatible with commercial EDA and needs only a few modifications to synchronous design flow. Monte-Carlo SPICE simulation shows that the pipelined multiplier applying the proposed techniques operates stably in 0.2V and achieves minimum power 1.3nW in 0.2V, minimum energy 1.07pJ per cycle in 0.3V. It provides 6.7 times superiority over synchronous baseline design with 22% area overhead. Comparison with other works in the state of art shows the proposed techniques are quite competitive.

Key words: asynchronous, sub-threshold, ultra-low-power, design methodology
Classification: Integrated circuits (logic)

1. Introduction
Striding toward the world of IOT and AI, it is indispensable to transfer and compute a mass of data, which poses great challenges to integrated circuits in terms of performance and energy[1, 2]. In many applications confined by the battery, energy exceeds performance to become the major concern in design trade-off. The demand for energy efficiency has motivated designers to scale voltage to the sub-threshold region due to the quadratic relation between dynamic power and supply voltage[3, 4]. However, along with the voltage scaling down to sub-threshold, the circuit delay degrades severely and become extremely sensitive to process, voltage and temperature (PVT) variation, which easily leads to timing errors.

The traditional method of sparing timing margin to cover the PVT variation is no longer feasible because of the unacceptable overhead. To deal with the problem, resilient design techniques are proposed[4, 5, 6]. Their basic principle is to monitor the critical path timing and react to the timing situation in situ. Once the timing is violated, it is corrected by error detection logic before error process logic increases the supply voltage to relieve the workload. However, these synchronous resilient techniques have defects in common. First, they are susceptible to metastability if data arrives near the clock transition, rather than after[7, 8]. Second, error detection window requires a large number of padding in short paths to avoid misdetection, which may introduce more power consumption than saving[6]. Third, error correction usually requires architecture modification[8]. Last, it requires error detection and correction within one clock cycle in the presence of degraded delay and slow LDO reaction[9].

Recently, researchers have shown great interest in the potential of asynchronous circuits being applied for variation tolerance[8, 10, 11]. Different from synchronous circuits, asynchronous circuits use local clocks to synchronize between neighbor registers. The meticulous neighbor handshaking renders the circuits inherently resilient, thus more variation-tolerant, energy-efficient and speed-fast. Nevertheless, asynchronous designs face three main challenges. First, the design overhead may outbalance the benefit. Second, asynchronous designs are much more complicated, especially the asynchronous timing constraint. Third, they lack commercial EDA supporting.

In this paper, we present two contributions with respect to the challenges. First, we propose a novel bundled-data[12] asynchronous pipeline for sub-threshold operation, called Snake. It is characterized by a looped request delay line (LRDL), which almost halves the design overhead compared to other asynchronous pipelines. In addition, the asynchronous pipeline is based on registers rather than latches, which simplifies the relative timing constraint (RTC)[13]. Second, we propose a practical design methodology perfectly compatible with commercial EDA, with only a few modifications to synchronous flow.

To demonstrate the effectiveness of the proposed techniques, we study a case using a 16x16 pipelined multiplier targeting SMIC 55nm CMOS technology. The
post-layout SPICE simulation and Monte-Carlo iterations show that the multiplier operates successfully in 0.2V with minimum power 1.3nW, and achieves minimum energy 1.07pj per cycle in 0.3V. The rest of this paper is organized as follows. Section 2 reviews the related work of asynchronous designs in sub-threshold and methodology. Section 3 presents the proposed Snake pipeline, its RTC, and performance. Section 4 introduces the asynchronous design methodology. Section 5 shows the experiment results and analysis. Section 6 concludes the paper.

2. Related work

2.1 Asynchronous sub-threshold design

In [10], Chang employs piecewise paralleled delay line as critical path replica (CPR) to alleviate the variation-induced mismatch between CPR and logic delay in a bundled-data asynchronous pipeline. To further decrease performance and energy penalty due to return-to-zero time in four-phase handshaking, he substitutes CPR buffers with And-gate type buffers skewed toward pulling down.

In [11], Liu adapts the MOUSETRAP asynchronous pipeline[14] to sub-threshold operation. He adds an extra delay to the control paths that make the latches opaque, thus widening the capture window and improving the ability to capture data delayed by random PVT variation.

In [8], Hand introduces an asynchronous template called Blade, which incorporates error detection concept from synchronous resilient techniques. Variation-induced violation is notified to previous and successive stages to trigger the recovery mechanism between neighbor stages. Owing to the meticulous handshaking, metastability and architectural modification in synchronous counterpart are avoided.

2.2 Asynchronous methodology

In general, Asynchronous design methodology can be classified into two groups[15]. One is to design dedicated languages that best capture the fine-grain concurrency and distributed synchronization of asynchronous pipelines, along with the synthesis and optimization tools[17, 18, 19]. Although it is a more ideal method, it requires redesign using new languages and conflicts with existing RTL designs. Moreover, the readily-available synthesis tools are at a higher level, leaving little optimization opportunity for designers. The other is to provide compatibility with existing synchronous languages and EDA tools. [20, 21, 22] redesign asynchronous standard cell (ASTC) libraries in NCL[22] style. After synchronous synthesis, they translate the netlists to ASTC. In [23], Cortadella utilizes a bundled-data template for the translation, allowing the use of commercial optimized STC. However, he fails to optimize the RTC during the synthesis and placement, thus the RTC may not be met.

3. Asynchronous template

This section first introduces the asynchronous control unit and a simple FIFO to exemplify the operation of Snake. Then we present two primitives, JOIN and FORK, to extend the applications in complicated nonlinear pipelines. Last, we elaborate the timing constraint and performance of Snake.

3.1 Asynchronous FIFO

In a classical bundled-data control unit[12], the request signal indicates the arrival of data while the acknowledgment signal denotes the success of receiving data. There is a one-sided timing constraint for bundled-data: data must arrive before request. Therefore, a delay line propagating the request is required to match the combinational logic of data, which is the main overhead of asynchronous templates.

The Snake is different. Fig. 1 shows the structure of the asynchronous control unit (ACU) in Snake. To reduce the overhead, the request is acknowledged halfway. By subtle design, the first half and second half delay line for propagating request can share, forming a looped request delay line (LRDL). The LRDL halves the design overhead while keeping the performance and variation tolerance. The request signal propagates in LRDL likes a snake, which is the origin of the pipeline name.

![Fig. 1 Proposed ACU](image)

It is essential to understand the function of each gate before we analyze the behavior of FIFO. The register with an inverter on its data path is used to convert specific direction transition at the clock pin to arbitrary direction transition at the output pin. To be specific in Fig. 1, a positive transition of req_N can trigger a transition of ack_N in the upper register while a negative transition of req_N can trigger a transition of done_N in the under register. The NOR gate is used to block the next request before the current request is handled. The XOR gate serves as OR function for transitions at the inputs. Any transition of done_N or ack_N produces a transition of req_N+1.
Consider the two-stage pipelined FIFO in Fig. 2. Each ACU generates the local clock for multi-bits wide data registers. The handshaking timing is shown in Fig. 3. Initially, all req and ack signals are reset to 0. ACU sends a request to ACU\(_{N+1}\) by setting req\(_{NL}\). In the meanwhile, the positive transition of req\(_{NL}\) triggers the data registers to send new data. After passing through LRDL\(_N\), the positive transition of req\(_{NR}\) is sensed by the upper register in ACU\(_{N+1}\) then an ack\(_N\) transition is sent back to ACU\(_N\), resetting req\(_{NL}\). After passing through LRDL\(_N\) again, the negative transition of req\(_{NR}\) is captured by under register in ACU\(_{N+1}\), setting req\(_{(N+1)L}\). Then the positive transition of req\(_{(N+1)L}\) triggers registers in stage N+1 to accept the new data, which indicates the end of handshaking in the current stage and the start of handshaking in the next stage.

Two thorough observations may help comprehension: 1) the moment req\(_{NL}\) is set, request along with data is sent to the next stage. In the meanwhile, NOR in ACU\(_N\) is gated by req\(_{NL}\) to block new request to ACU\(_N\) in case of flushing previous request and data. 2) req\(_N\) acts like 4-phase while ack\(_N\) acts like 2-phase. To be clear, in complete handshaking, req\(_N\) is set and returns to zero while ack\(_N\) just inverts once. In other words, req\(_N\) is state-signaling while ack\(_N\) is transition-signaling.

We elaborate on the advantages of the asynchronous pipeline as follows:
1) It uses registers to store states of control and data signals, rather than latches in typical asynchronous templates. On one hand, it makes the design glitchless and robust even under sub-threshold operation. On the other hand, it is free from the timing loop that latch templates undergo, simplifying the RTC constraint.
2) The LRDL multiplex the first half and second half request delay line, thus halving the design overhead, further suitable for ultra-low-power applications.

3.2 Nonlinear pipeline: JOIN and FORK
In real systems, pipelines are usually nonlinear, which means a stage may be requested to receive data from multi stages, or requests to send data to multi stages. To extend the applications in nonlinear pipelines, we provide elegant nonlinear interfaces: JOIN and FORK elements, as is shown in Fig. 4.

The NAND gate in JOIN element serves as an AND function for the positive transitions of requests while the NOR gate serves as an AND function for the negative transitions of requests. Muller C Element[24] in FORK element serves as an AND function for the transitions of acknowledgments.

With the simple elegant interfaces, the nonlinear stage can communicate with multi predecessor stages and successor stages reliably.

3.3 Timing constraint and performance
As is mentioned above, a one-sided RTC must be met that the request arrives earlier than data by setup time of the register. It poses a constraint for the length of LRDL: 
\[2 \cdot T_{LRDL} + T_{JOIN} + T_{FORK} > T_{setup} + T_{logic} \] (1)
Where T\(_{JOIN}\) and T\(_{FORK}\) are the total delay of gates in JOIN and FORK elements respectively, and T\(_{logic}\) is the propagation delay of data.

By introducing T\(_{margin}\), Eq. (1) can be expressed as:
\[T_{LRDL} = \frac{1}{2}(T_{setup} + T_{logic} + T_{margin} - T_{JOIN} - T_{FORK}) \] (2)

The NOR gate in ACU blocks the next request (NR) when current request (CR) is not acknowledged, which pose a hold constraint for the pipeline that CR should arrive at NOR earlier than NR. In Fig. 2, the moment when CR propagates to req\(_{NL}\) after a transition of ack\(_N\), it produces a negative transition to release NR blocked by...
NOR in ACU. Thereafter, CR propagates in stage N while NR propagates in both stage N-1 and N before they arrive at the inputs of NOR in ACU. The constraint can be expressed as:

\[ T_{LRDL,N} - T_{ACU,N} + T_{JOIN,N} + T_{FORK,N} + T_{LRDL,N} > T_{FORK,N} + T_{LRDL,N} + T_{reg,N+1} + T_{XOR,N+1} \]  

(3)

It is simplified as:

\[ T_{LRDL,N} - T_{ACU,N} + T_{JOIN,N} > T_{reg,N+1} + T_{XOR,N+1} \]  

(4)

Because all the ACU share the same structure,

\[ T_{ACU,N} > T_{reg,N+1} + T_{XOR,N+1} \]  

(5)

Therefore, hold constraint Eq. (3) is satisfied inherently in the pipeline.

Glitch-free is guaranteed by the constraints above, along with the proposed pipeline timing. With the application of bundled-data scheme, sequential hazards in the data paths are eliminated[14, 26], as long as the setup constraint is satisfied. By restricting the transitions of multi-inputs in combinational gates, combinational hazards in the control paths are defeated[27, 28]. To be specific, in JOIN element, reqN and req1N only transit in the same direction in the stage request, where it is similar in FORK element. In NOR and XOR in ACU, only one input transits at a time due to the hold constraint and the protocol timing.

We use cycle time[28] to describe the performance of the asynchronous pipeline. Cycle time is the time interval between successive data items when the pipeline operates at maximum speed. A cycle for stage N starts from a positive transition of reqN and ends at the subsequent positive transition of reqN+1. Therefore the cycle time of the stage is

\[ T_{cycle} = 2 \cdot T_{LRDL} + T_{JOIN} + T_{FORK} \]

(6)

\[ = T_{setup} + T_{logic} + T_{margin} \]  

(7)

Eq. (7) indicates that the cycle time of a stage depends on the logic of this stage, irrelevant to other stages. The performance is superior to synchronous design, in which the cycle time of a stage is the global clock cycle time, depending on the longest stage.

4. Design methodology

The design flow is automated by Synopsis EDA tools through some homemade TCL scripts. The scripts include EDA built-in commands for circuit analysis, and common program code for information process, such as set, if, foreach. The flow is shown in Fig. 5.

![Fig. 5 Design flow](image)

It starts from a synthesizable RTL design. After synthesized to a synchronous netlist, it is imported in PrimeTime (PT) for topology analysis. The needed topology information includes stage critical paths and stage connections. A stage is the abstract terminals of a multi-bits wide data path, represented by multi-bits widen registers in the circuits. The registers in the same stage are identified by setting a stage name using built-in command set_user_attribute. Stage critical paths are acquired by iterating every pipeline between two stages using built-in command get_timing_paths. By analyzing the startpoints and endpoints of the stage critical paths further, we can get the stage connection information, which is stored in arrays, a TCL data structure. For each stage, one array stores its predecessor stages and the other array stores its successor stages.

The netlist is then imported in Verdi for ECO. By iterating every stage, the clock in netlist is replaced with distributed asynchronous templates according to the stage connection information acquired in the topology analysis. In detail, the clock net is removed by built-in command _ecoDeleteConnection. The ACU, JOIN, and FORK are inserted by built-in command _ecoAddInst. The connection between asynchronous units and data registers is done by built-in command _ecoMakeConnection. As an example, the following code shows the ACU insertion and the connection with stage registers.

```tcl
foreach stage $all_stages {
    _ecoAddInst -master ACU -inst $ACU_STAGE
    foreach register $stage_registers {
        _ecoDeleteConnection -input $register
        _ecoMakeConnection -input $register
    }
}
```

The raw netlist from ECO does not meet constraint in Eq. (1) in the absence of LRDL. Therefore we insert LRDL in the pipeline to optimize the timing in DsignmentCompiler (DC) by built-in command set_min_delay. The length of each LRDL is determined through Eq. (2), with the timing information of stage critical paths. In the back-end, the clock tree synthesis in regular flow is replaced with asynchronous timing optimization, where the timing constraint is the same as that in frontend. Apart from this, other steps keep the same with regular backend flow.

As can be seen, the proposed design methodology has the following merits:

1) It is compatible with existing RTL design. It is crucial because commercial designs are based on IP, which does not allow designers to modify the design.

2) It is compatible with commercial EDA. The added steps are automated by Synopsis EDA using built-in commands.

3) RTC optimization is simple because it is derived from the register-based template.
5. Experiment results

We use a 16x16 pipelined multiplier as a benchmark to evaluate the proposed techniques. The initial RTL is synthesized at 200MHZ, in 1.08V, SS corner, and 125°C targeting SMIC 55nm process. Thereafter, the asynchronous design is implemented as a layout through the proposed flow. For comparison, a synchronous baseline design is also implemented at the same condition through traditional design flow. Then the layouts are extracted with parasitic RC for accurate SPICE simulation.

We run 100 Monte-Carlo iterations for each supply voltage. The results show that the circuit operates robustly in 0.2V, almost the breakdown voltage of STC. Fig. 6 shows the timing diagram for a multiply operation in 0.3V, SS corner and 0°C. Fig. 7 shows the distribution of energy and cycle time over 100 Monte-Carlo iterations in 0.3V. We investigate the energy consumption and performance further by operating the pipeline at maximum throughput, fed with the random operands. The performance is evaluated by cycle time while the energy is evaluated by energy consumption per cycle or average power consumption over a cycle. Fig. 8 shows the relation of the energy and cycle time with supply voltage. As can be seen, the minimum power point is 0.2V with 1.3nW, while the minimum energy point is 0.3V with 1.07pJ.

![Fig. 6 Simulation waveform in 0.3V, SS corner, 0°C](image)

![Fig. 7 Energy and cycle time distribution over 100 Monte-Carlo iterations in 0.3V](image)

Compared to synchronous baseline design, the asynchronous design occupies 22% more area, which is expected to be even less if the data width is bigger or the clock distribution in the original design is more complicated. As for energy, the proposed technique reduces the energy, which is 8.24pJ per cycle in the baseline, by 6.7 times. The energy efficiency comes in three ways: 1) the tolerance for variation in sub-threshold supply voltage; 2) eliminated unnecessary transitions in clock paths; 3) resilient cycle time for each stage.

The comparison of our work with other state of art is summarized in Table I. It indicates that the proposed techniques are superior over other works in the state of art in sub-threshold ultra-low-power applications.

| Technology | Architecture | Voltage | Area | Cycle time | Power | Energy |
|------------|--------------|---------|------|------------|-------|--------|
| [29] JSSCC'11 | 8-bit Multiplier | 84mV | 0.019mm² | 66us | 29.9nW | 1.97pJ |
| [30] VLSI'17 | 16-bit Multiplier | 0.8V | N/A | 2.7ns | 1.4mW | 3.8pJ |
| [31] LSSC'18 | 16-bit FIR | 0.85V | 1.32mm² | 971us | 99.1nW | 96pJ |
| [32] JSSC'19 | 8-bit MAC | 0.4V | 2048um² | 50ns | 2.2mW | 0.11pJ |
| This work | 16-bit Multiplier | 0.3V | 0.0441mm² | 16us | 1.3N | 1.07pJ |

6. Conclusion

In this paper, we propose a novel asynchronous pipeline, called Snake, for ultra-low-power applications. The inherent resilience of the asynchronous template makes it variation-tolerant in sub-threshold. Based on registers rather than latches, the template simplifies the timing constraint and makes the design more robust in sub-threshold. By acknowledging halfway and looping the delay line, the pipeline halves the design overhead. In addition, we propose a practical asynchronous design methodology with only a few modifications to synchronous flow. It is compatible with existing RTL design and commercial EDA. Besides, the RTC optimization in the flow is quite simple.

Monte-Carlo SPICE simulation manifests that the...
multiplier applying the proposed techniques works robustly in 0.2V. It achieves minimum power 1.3nW in 0.2V, and minimum energy 1.07pJ per cycle in 0.3V at maximum throughput. Compared to synchronous baseline design, it decreases energy by 6.7 times with 22% area overhead. It is also quite competitive compared to other works in the state of art.

Acknowledgments

This work is supported by National Natural Science Foundation of China (61474135).

References

[1] L. Atzori, et al.: “The Internet of Things: A survey,” Comput. Netw. 54 (2010) 2787 (DOI:10.1016/j.comnet.2010.05.010).
[2] D. Shin, et al.: “DNPU: An Energy-Efficient Deep-Learning Processor with Heterogeneous Multi-Core Architecture,” IEEE Micro 38 (2018) 85 (DOI: 10.1109/MM.2018.053631145).
[3] L. Lin, et al.: “A 595pW 14pCycle microcontroller with dual-mode standard cells and self-startup for battery-indifferent distributed sensing,” 2018 IEEE International Solid-State Circuits Conference - (ISSCC) (2018) 44 (DOI: 10.1109/ISSCC.2018.8310175).
[4] Y. Zhang, et al.: “8.8 iRazor: 3-transistor current-based error detection and correction in an ARM Cortex-R4 processor,” in 2016 IEEE International Solid-State Circuits Conference (ISSCC) (2016) 160 (DOI: 10.1109/ISSCC.2016.7417956).
[5] K. Chae and S. Mukhopadhyay: “A Dynamic Timing Error Prevention Technique in Pipelines With Time Borrowing and Clock Stretching,” IEEE Trans. Circuits Syst. Regul. Pap. 61 (2014) 74 (DOI: 10.1109/TCSC.2013.2268272).
[6] S. Kim and M. Seok: “Variation-Tolerant, Ultra-Low-Voltage Microprocessor With a Low-Overhead, Within-a-Cycle In-Situ Timing-Error Detection and Correction Technique,” IEEE J. Solid-State Circuits 50 (2015) 1478 (DOI: 10.1109/JSSC.2015.2418713).
[7] S. Beer, et al.: “Metastability in better-than-worst-case designs,” 2014 20th IEEE International Symposium on Asynchronous Circuits and Systems (2014) 101 (DOI: 10.1109/ASYNC.2014.21).
[8] D. Hand, et al.: “Blade – A Timing Violation Resilient Asynchronous Template,” 2015 21st IEEE International Symposium on Asynchronous Circuits and Systems (2015) 21 (DOI: 10.1109/ASYNC.2015.13).
[9] D. Ernst, et al.: “Razor: a low-power pipeline based on circuit-level timing speculation,” 22nd Digital Avionics Systems Conference. Proceedings (2003) 7 (DOI: 10.1109/MICRO.2003.1253179).
[10] I. J. Chang, et al.: “Exploring Asynchronous Design Techniques for Process-Tolerant and Energy-Efficient Subthreshold Operation,” IEEE J. Solid-State Circuits 45 (2010) 401 (DOI: 10.1109/JSSC.2009.2036764).
[11] J. Liu, S. M. Nowick, and M. Seok: “Soft MOUSETRAP: A Bundled-Data Asynchronous Pipeline Scheme Tolerant to Random Variations at Ultra-Low Supply Voltages,” 2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems (2013) 1 (DOI: 10.1109/ASYNC.2013.29).
[12] C. L. SEITZ: “System timing,” Introd. VLSI Syst. (1993) http://ai.eecs.berkeley.edu/people/conway/VLSI/VLSText/PPV3/2s/V3. Ch7-2s.PDF.
[13] K. Stevens, R. Ginosar, and S. Rotem: “Relative timing,” Proceedings. Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems (1999) 208 (DOI: 10.1109/ASYNC.1999.761535).
[14] M. Singh and S. M. Nowick: “MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems 15 (2007) 684 (DOI: 10.1109/TVLSI.2007.898732).
[15] S. M. Nowick and M. Singh: “Asynchronous Design—Part 2: Systems and Methodologies,” IEEE Des. Test 32 (2015) 19 (DOI: 10.1109/MDAT.2015.2413757).
[16] C. Ebergen: “A formal approach to designing delay-insensitive circuits,” Distrib. Comput. 5 (1991) 107 (DOI: 10.1007/BF02252954).
[17] D. Edwards: “Balsa: An Asynchronous Hardware Synthesis Language,” Comput. J. 45 (2002) 12 (DOI: 10.1093/compjnl/45.1.12).
[18] P. A. Beere, et al.: “Proteus: An ASIC Flow for GHZ Asynchronous Designs,” IEEE Des. Test Comput. 28 (2011) 36 (DOI: 10.1109/MDT.2011.114).
[19] R. B. Reese, et al.: “Uncle - An RTL Approach to Asynchronous Design,” 2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems (2012) 65 (DOI: 10.1109/ASYNC.2012.14).
[20] M. Moreira, et al.: “Semi-custom NCL Design with Commercial EDA Tools: Is it Possible?”, in 2014 20th IEEE International Symposium on Asynchronous Circuits and Systems (2014) 53 (DOI: 10.1109/ASYNC.2014.15).
[21] Z. Shin, et al.: “Design of a clockless MSP430 core using mixed asynchronous design flow,” IEICE Electron. Express 14 (2017) 20170162 (DOI: 10.1587/elex.14.20170162).
[22] K. M. Fant and S. A. Brandt: “NULL Convention Logic: a complete and consistent logic for asynchronous digital circuit synthesis,” Proceedings of International Conference on Application Specific Systems, Architectures and Processors (1996) 261 (DOI: 10.1109/ASAP.1996.542821).
[23] J. Cortadella, et al.: “Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 25 (2006) 1904 (DOI: 10.1109/TCAD.2005.869058).
[24] I. E. Sutherland: “Micropipelines,” Commun ACM 32 (1989) 720 (DOI: 10.1145/63526.63532).
[25] S. M. Nowick and M. Singh, “High-Performance Asynchronous Pipelines: An Overview,” IEEE Des. Test Comput., vol.28, no. 5, pp.8–22, Sep. 2011.
[26] S. H. Unger, “Hazards, critical races, and metastability,” IEEE Trans. Comput., vol.44, no. 6, pp.754–768, Jun. 1995.
[27] S. H. Unger, Asynchronous sequential switching circuits. Wiley, New York, NY, 1969.
[28] T. E. Williams: “Self-timed Rings and Their Application to Division,” Ph.D. Dissertation, Stanford University, Stanford, CA, USA (1991).
[29] N. Lotze and Y. Manoli: “A 62mV 0.13um CMOS standard-cell-based design technique using schmitt-trigger logic,” 2011 IEEE International Solid-State-Circuits Conference (2011) 340 (DOI: 10.1109/ISSCC.2011.5746345).
[30] J. P. Cerqueira and M. Seok: “Temporarily Fine-Grained Sleep Technique for Near- and Subthreshold Parallel Architectures,” IEEE Trans Very Large Scale Integr Syst 25 (2017) 189 (DOI: 10.1109/TVLSI.2016.2576280).
[31] J. P. Cerqueira, J. Li, and M. Seok: “A 1W- and kHz-Class Feedforward Leakage Suppression: Is it Possible?,” in 2019 22nd Digital Avionics Systems Conference (2019) 1 (DOI: 10.1109/ASYNC.2019.8882139).