

# An Asynchronous Constant-Time Counter for Empty Pipeline Detection

Jonathan Tse and Derek Lockhart

December 2009, Computer Systems Laboratory, Cornell University

{jon, lockhart}@csl.cornell.edu

**Abstract**—Leakage losses have become a significant portion of power consumption in nanoscale circuits. Power gating techniques are effective in mitigating these losses, especially for low duty cycle systems and pipelines. It is of particular importance to determine whether a pipeline is empty before power gating it. If tokens are still in flight and useful computation is still being performed, preemptively shutting off a pipeline could destroy state, data, and execution correctness. We propose an asynchronous constant-time counter for use in empty pipeline detection, assuming the power savings from power gating the pipeline amortize the additional power consumption of the counter.

## I. INTRODUCTION

Continued technology scaling has brought new challenges in chip power management as voltage levels have failed to scale commensurately with decreasing transistor sizes. While dynamic power still plays a significant role in circuit power dissipation, static power due to device leakage is becoming an ever increasing component of total chip power consumption [1], [2]. Asynchronous circuits elegantly mitigate dynamic power consumption as they inherently implement the equivalent of a fine-grained clock-gating network—unused components naturally consume zero dynamic power. However, static power is more troublesome, especially when taking into account the additional area overhead of asynchronous circuits compared to their synchronous counterparts.

One technique for reducing leakage currents that has been successfully applied to both synchronous and asynchronous circuits is the power gating of unused components [3], [4]. This approach isolates idle circuits from their power rails when not in use, effectively reducing leakage currents. Entire pipelines can be power gated when not in use to save power; however, the system must first ensure the pipeline is empty in order to avoid data loss and ensure execution correctness. A number of techniques are available for empty pipeline detection [5], but one of the lowest-overhead approaches simply monitors token flow at the entrance and exit boundaries of the pipeline.

This technique requires an efficient counter implementation to keep track of tokens. It is critical that the token counter not delay the entrance or exit of data

in the pipeline because additional latency will ripple into other parts of the system and adversely affect pipeline throughput. Additionally, the counter latency should remain constant irrespective of the number of bits, since aggressively pipelined systems minimize the latency of each pipeline stage at the cost of having more stages.

For our project, we present an implementation of a constant-time counter used for empty pipeline detection. The counter is capable of servicing increments, decrements, and zero-value check events in constant time. Increments are triggered on token entrance into the pipeline and decrements on token exit. The zero-value—representing the empty state of a pipeline—is checked after servicing an increment or decrement. A controller is used to arbitrate between arriving increment and decrement events, so that the simultaneous arrival of increment and decrement events can be canceled out. This effectively allows the counter process to save power by taking no action. In the case of a counter servicing a pipeline operating at full throughput, this canceling out effect should be very common.

## II. SYSTEM DESIGN

### A. Counter Controller Design

Since the counter must ensure that increments and decrements occur atomically, the system, seen in Figure 1, must guarantee that the counter does not receive an increment (decrement) command while a decrement (increment) request is currently being serviced. Additionally, we'd like to ensure that simultaneous increments and decrement requests, the net effect of which is a no-op, are acknowledged but not acted upon. A controlling process for the counter is necessary to ensure these two requirements are met.

Note that the control must detect not only the presence of an increment or decrement request, but also the absence of one. As such, we also need a process to properly implement negated probes, which are inherently unstable—a probe will remain true until it is acknowledged, but a probe that is false may become true at any time. Fortunately, we can create a stable

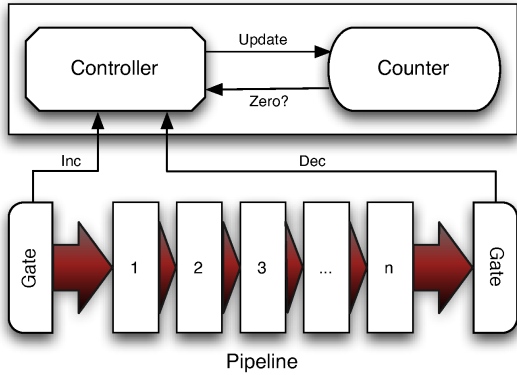


Figure 1. Overall counter system, comprising the pipeline under observation, counter controller, and constant time counter stack.

data channel to represent a probe's state and use an observing process to update this data channel [6]. This new data channel acts as a proxy for the probe which can be queried for negated probe status without violating stability assumptions. Such a process is shown below:

*Negated Probe*  $\equiv$

$$*[[\bar{A} \rightarrow PA!\mathbf{true}, A | \neg\bar{A} \rightarrow PA!\mathbf{false}]]$$

The controller is constructed in the following manner: the increment (*Inc*) and decrement (*Dec*) signals each utilize a dedicated negated probe process which monitors the probe state and updates the proxy channels (*IncP/DecP*) as necessary. Another process waits for the true condition of either the *Inc* or *Dec* channel, copies the status of the proxy channels into local variables, and then communicates the appropriate update command to the counter (*C\_Inc* or *C\_Dec*). Note that if both the *IncP* and *DecP* channels evaluate to true, a skip command is issued instead of propagating the increment and decrement requests to the counter. The control process is shown here, where *IncP* and *DecP* are the proxy channels from the negated probe process as described above:

*Counter Controller*  $\equiv$

$$*[[\overline{Inc \vee Dec}; \\ IncP?x, DecP?y; \\ [x \wedge y \rightarrow \mathbf{skip} \\ \square x \wedge \neg y \rightarrow C\_Inc \\ \square \neg x \wedge y \rightarrow C\_Dec \\ ]]]$$

### B. Constant-Time Counter Design

The counter is designed to provide constant-time increment and decrement operations, as well as low-overhead updates of zero status. The fundamental design element is a single-bit counter, several of which are

connected together to create a counter with multi-bit capacity as seen in Figure 2. Each single-bit counter accepts an increment or decrement signal and, in the event of a carry operation, sends an increment or decrement signal to the next counter. Additionally, each single-bit counter maintains a local copy of the bit it represents.

A novel feature of this counter is the use of a sticky-zero bit to provide a low-overhead determination of zero status. The sticky-zero bit of each single-bit counter represents whether or not all counters above it—those closer to the MSB—are zero. By inspecting the data and sticky-zero bits of the LSB counter, we can determine if the entire multi-bit counter is empty. Each single-bit counter has a channel to read the sticky-zero status of higher bits (*ZeroU*) and a channel to communicate sticky-zero status (*Zero*) to lower bits.

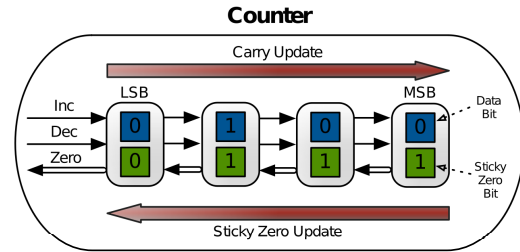


Figure 2. Detail of constant-time counter stack.

The single-bit counter process operates by waiting on a communication action on either its increment (*C\_Inc*) or decrement (*C\_Dec*) channel, both of which are dataless. An increment action always results in the transmission of a false value on the one-bit *Zero* channel, since an increment action always implies that the multi-bit counter is non-zero. A decrement action transmits the AND of the sticky-zero bit (*sz*) and the local bit value (*x*) on the *Zero* channel. Transmission of the zero value on the *Zero* channel is always the first communication handshake to complete in order to ensure that multi-bit counter updates remain constant time regardless of the number of bits. Note that increment and decrement actions will result in a  $x := \neg x$  transition, and that the communication on *Zero* occurs before the  $x := \neg x$  transition. This ensures the transmission of *sz* AND *x* is correct, as *x* will eventually be  $\neg x$  at the end of the execution of the process.

After the *Zero* channel handshake is complete, the single-bit counter process then proceeds with completing the increment or decrement handshake, the carry channel communication, and the local bit value update, all of which may happen in parallel. The carry channel communication (*C\_IncU/C\_DecU*) only occurs when a carry operation is necessary, otherwise a skip is executed. A separate process, pulled out during process decompo-

sition, is used to update the status of the sticky-zero bit. The CHP for a single-bit counter is shown below:

$$\begin{aligned} \text{Single Bit Counter} \equiv & \\ * [ & \overline{Inc} \rightarrow \text{Zero!false}; Inc, \\ & [x \rightarrow IncU \parallel \neg x \rightarrow IncF], \\ & x := \neg x \\ & \parallel \overline{Dec} \rightarrow \text{Zero!}(sz \wedge x); Dec, \\ & [x \rightarrow DecF \parallel \neg x \rightarrow DecU], \\ & x := \neg x \\ & ] ] \\ \parallel & \\ * [ & \text{ZeroU?sz} \end{aligned}$$

### C. Interleaved Counter Design

SPICE analysis of our initial design indicated that the counter latency had a much greater effect on throughput than we anticipated. In practice, increments and decrements arrived out of phase in such a way as to cause to common modes of operation: one where the simultaneous increment and decrement case occurs, followed by the halving of throughput case, where increments and decrements are serviced atomically. This can be seen in Figure 3 below.

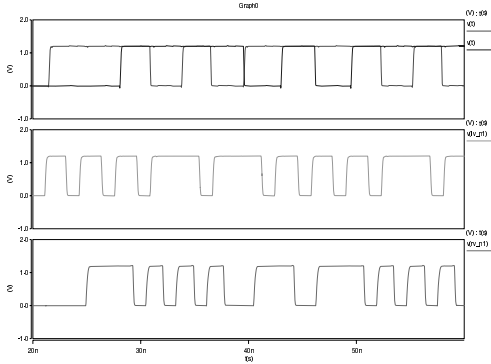


Figure 3. Stretched increment and decrement handshakes. The middle trace is the probe of the increment channel, and the bottom trace is that of the decrement channel. The topmost trace represents the local variables  $x$  and  $y$  within the counter controller. Note the ideal case is for both  $x$  and  $y$  to be simultaneously true. While this does occur, it does not occur in steady state. Also note the clear stretching of the channel probes, indicating exclusive, atomic servicing of either an increment or decrement event.

To address this issue, we implemented an interleaved counter arrangement, an idea borrowed from a previously proposed interleaved asynchronous adder design<sup>1</sup>. The modified designs, seen in Figure 4, uses two deterministic splits, one for  $Inc$  and one for  $Dec$ , and two complete counter structures. One counter keeps track of

odd tokens, and the other even. The new empty pipeline test is the AND of the empty test from each counter.

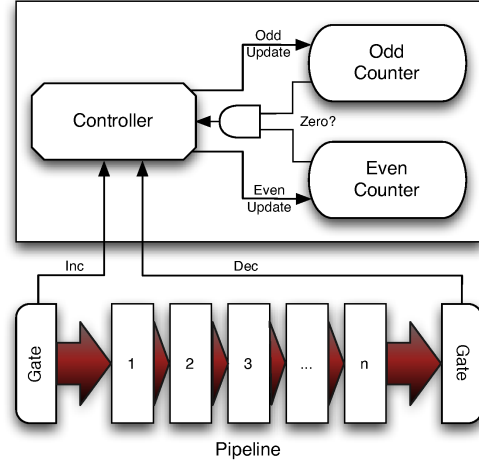


Figure 4. Interleaved counter system.

Below is the CHP for the deterministic split:

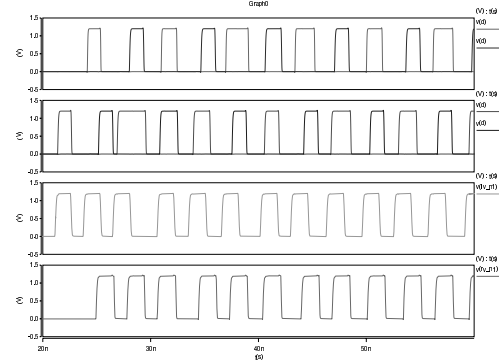
$$\begin{aligned} \text{Deterministic Split} \equiv & \\ * [ & \overline{In} \rightarrow [\neg x \rightarrow Odd \parallel x \rightarrow Even]; \\ & x := \neg x, In] \end{aligned}$$


Figure 5. Interleaved increment and decrement handshakes. The bottom two traces are the probes of increment and decrement, respectively. The top two traces show the interleaving behavior of increments and decrements for the odd and even counters, respectively.

Although the updated interleaved counter design successfully achieves full throughput, one side-effect is an unfavorable change in the common case of operation. Increment and decrement events arriving at each counter controller are almost always perfectly interleaved, as seen in Figure 5, all but eliminating the occurrence of simultaneous request arrivals resulting in no-ops. A positive consequence of an interleaved design is that it permits a drastic reduction in the number of bits

<sup>1</sup>Sheikh and Manohar, to be submitted to IEEE ASYNC 2010

needed by the counter structures, since each counter switches between 0 tokens and 1 token in the general case. While we cannot reduce the system to only one bit per counter and remain QDI, reducing the number of inactive single bit counters does allow for a reduction in transistor count, area, and leakage current. Additionally, the interleaved counter design also reduces the number of forward transitions from 32 to 34.

### III. EVALUATION

The constant-time counter presented above was fully implemented using the following design methodology: a behavioral description of the circuit was created using high-level CHP, from which handshaking expansions were generated, reshuffled, and then transformed into valid bubble-reshuffled production rule sets. A number of iterations of the counter using various HSE reshufflings were produced until an implementation with appropriate constant-time behavior and a 34 transition forward latency was achieved. Transistor sizing, and folding where necessary, were created for the production rules and the resulting circuits were simulated using SPICE to obtain performance and power numbers. This process was repeated to implement the interleaved counter design. For our evaluation, we instrumented a bare 25-stage pipeline of buffers with our counter designs. Power consumption, throughput, leakage statistics for the bare pipeline of buffers (BP), the pipeline instrumented with a counter (C), and the same pipeline instrumented with a pair of interleaved counters (IC) is presented below in Table I.

Table I  
PERFORMANCE AND POWER RESULTS

	Freq. (MHz)	Dynamic ( $\mu$ W)	Static (nW)
BP	489.50	691.09	212.61
C	263.96	479.79	353.64
IC	317.80	824.04	807.17

### IV. CONCLUSION

The results of our analysis indicate that the implementation of our empty-pipeline detection counter not only works, but is capable of performing updates and zero-status determination in constant time. Despite the constant-time nature of the design, the counter still has a non-trivial impact on pipeline performance.

The high-performance interleaved counter implementation incurred a 35% performance cost when compared to the bare pipeline pipeline of buffers. The non-interleaved implementation suffered an even larger 46% performance degradation. This is largely due the inclusion of the additional handshakes on the pipeline's critical path: any arriving (departing) token must wait for

the controller to initiate an update of the counter before entering (exiting) the pipeline.

As for power numbers, the interleaved counter increases the dynamic power of the system by 20%, while nearly quadrupling its static power. The non-interleaved counter demonstrates a much more reasonable static power overhead, and even manages to reduce the dynamic power when compared to the bare pipeline of buffers. The reduced static power overhead of the non-interleaved counter is clearly a result of its lower area overhead, since only one counter is used instead of two. Despite having similar bit counts, the interleaved counter has double the control circuitry and the addition of the deterministic splits. The reduced dynamic power of the non-interleaved counter is due to both the reduced frequency and the power saved by simultaneous increments and decrements becoming no-ops.

One important thing to note is that these results only consider the case where the pipeline is always active and never receives the benefit of power-gating. A system with long periods of inactivity could potentially achieve a marked reduction in overall total dissipated power, despite the energy overhead of the counter circuitry. Determining if this technique is viable for a given system would require profiling of the target pipeline's behavior to establish whether the power cost of the counter would be amortized by the energy saved by power gating. In future work we'd like to explore this trade off in more detail to determine what kinds of systems could benefit from our empty-pipeline detection hardware.

### ACKNOWLEDGEMENTS

The authors would like to thank Prof. Rajit Manohar for his instruction and guidance and Carlos Tadeo Otero Ortega for his invaluable assistance in building SPICE test harnesses.

### REFERENCES

- [1] A. Keshavarzi, K. Roy, and C. Hawkins, "Intrinsic leakage in low power deep submicron cmos ics," *Test Conference, 1997. Proceedings., International*, pp. 146 – 155, Nov 1997.
- [2] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits," *Proceedings of the IEEE*, vol. 91, pp. 305 – 327, Feb 2003.
- [3] H. Deogun, D. Sylvester, and K. Nowka, "Fine grained multi-threshold cmos for enhanced leakage reduction," *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, p. 4, Jan.
- [4] T. Lin, K.-S. Chong, B.-H. Gwee, and J. Chang, "Fine-grained power gating for leakage and short-circuit power reduction by using asynchronous-logic," *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pp. 3162 – 3165, May 2009.
- [5] D. Fang, F. Akopyan, C. T. O. Otero, and R. Manohar, "Conditional slack matching," pp. 1–11, Sep 2009.
- [6] R. Manohar, M. Nystrom, and A. Martin, "Precise exceptions in asynchronous processors," *Advanced Research in VLSI, 2001. ARVLSI 2001. Proceedings. 2001 Conference on*, pp. 16 – 28, Mar 2001.