RC22239 (W0110-022) October 4, 2001
Computer Science

IBM Research Report

Synchronous Interlocked Pipelined CMOS

Hans Jacobson, Prabhakar N. Kudva, Pradip Bose, Peter W. Cook, Stanley E. Schuster
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Eric Mercer
Verification and Analysis Department
IBM Austin Research Laboratory

Synchronous Interlocked Pipelined CMOS

Hans Jacobson Prabhakar Kudva
Design Automation Department
IBM T.J. Watson Research Center

Pradip Bose Peter Cook Stanley Schuster
High Performance Systems Department
IBM T.J. Watson Research Center

Eric Mercer
Verification and Analysis Department
IBM Austin Research Laboratory

Abstract

In a circuit environment that is becoming increasingly sensitive to dynamic power dissipation and noise, and where cycle time available for control decisions continues to decrease, locality principles are becoming paramount in controlling advancement of data through pipelined systems. Achieving fine grained power down and progressive pipeline stalls at the local stage level is therefore becoming increasingly important to lower dynamic power consumption while keeping introduced switching noise under control and to avoid global distribution of timing critical stall signals.

It has long been known that the interlocking properties of asynchronous pipelined systems have a potential to provide such benefits. However, it has not been understood how such interlocking can be achieved in synchronous pipelines. This report presents a novel technique based on local clock gating and handshake protocols that achieves interlocking characteristics in synchronous pipelines similar to that of asynchronous pipelines. Furthermore, the technique provides improved storage capacity of synchronous pipelines and queues, allowing up to twice as many data items to be stored compared to traditional synchronous approaches. The presented technique is directly applicable to traditional synchronous pipelines and works equally well for two phase clocked pipelines based on transparent latches, as well as one phase clocked pipelines based on master-slave latches and flip-flops.

Background

In our search for new techniques to achieve lower power in synchronous circuits and systems at IBM we have carefully studied, as well as developed [4], asynchronous pipeline techniques. Asynchronous pipelines [5] have several properties that have the potential to benefit todays and tomorrows circuit design. Two of the more promising benefits of asynchronous pipelines we feel are computing on demand and the locality principles that stem from pipeline interlocking. However, there are considerable resources available in design tools and design expertise for synchronous design techniques that cannot be discounted when designing commercial chips. It is therefore desirable to find some middle ground in techniques that can provide some of the benefits of asynchronous circuits in a synchronous context. The elastic and interlocked synchronous pipeline techniques presented in this report are the result of our efforts to achieve this goal.

1 Introduction

Power dissipation is becoming a major design constraint, not only in portable, but also in high-performance VLSI systems. As clock and latch power is nearing 70% of the total power consumption in synchronous integrated circuits that do not employ clock gating, power aware techniques that perform computations only on demand are becoming necessary to meet power budgets. At the same time, growing transistor density and lower transistor thresholds are causing increased switching currents and reduced noise margins. As a consequence, simultaneous switching noise and power supply ringing [1] due to large variances in switching currents is becoming a concern. To implement computing on demand while guarding against large variances in switching current, fine grained clock gating at the stage level is becoming an increasingly important technique.

Pipeline stalls in the backward direction of a pipeline is another concern. Stalls are not only a concern from a perspective of power and effects on switching current but also from the point of view of signal locality. Compared to the data recirculation approach often used today, clock gating is a low power alternative to implement pipeline stalls. Stalls have traditionally been performed at the unit level, rather than at the more fine grained stage level. Coarse grained stalls can cause large cycle to cycle variance in switching currents and also require global propagation of stall signals. With stall signals already on the critical path in some of todays designs, cycle time may come to be affected as wire delays do not scale well with technology. Locality principles are therefore becoming increasingly important in pipelined design. Due to the concerns with switching currents and wire delays, it is becoming increasingly difficult to design for, and cost-effectively implement, stalling of synchronous pipelines.

Contrary to synchronous pipelines, asynchronous pipelines are not affected by the mentioned problems to the same extent. The stages in an asynchronous pipeline are interlocked through request-acknowledge protocol handshakes that ensure correct progression of data through the pipeline. This interlocking provides several benefits. One benefit of asynchronous interlocking is fine grained power down at the pipeline stage level. A stage is only requested to compute when a computation is required. In pipelines with low utilization significant power can be saved when computation is performed only on demand. Another benefit of interlocking is the inherent locality of control decisions when controlling the progression of data through the pipeline. This allows stalls to be performed on a local basis, one pipeline stage at a time. This way of progressively stalling a pipeline avoids global distribution of stall signals and also keeps the cycle to cycle variance in switching currents low.

To our knowledge, no method to similarly stall synchronous pipelines at a fine grained level has been reported in literature. This report presents a cost-effective solution to this problem through a technique based on clock gating that achieves backward interlocking between stages in a synchronous pipeline that is similar to the acknowledge interlocking found in asynchronous pipelines. This *elastic synchronous pipeline* (ESP) achieves progressive, stage by stage, stalling of synchronous pipelines at no delay overhead and at a very small area cost. Furthermore, this report presents an extension of these elastic pipelines to fully *interlocked synchronous pipelines* (ISP), where each stage is interlocked with its neighboring stages in the forward as well as backward direction.

Contributions

The synchronous interlocking technique presented in this report provides several benefits to the design of synchronous pipelines including fine grained power down, cost-effective stalling of high frequency pipelines, improved queue storage properties, and the potential to allow a direct mapping between synchronous and asynchronous implementations. The following list provides a brief overview of the potentials of interlocked synchronous pipelines.

1. The presented technique provides local interlocking between individual stages in a synchronous pipeline through the use of local clock gating and handshake protocols. This interlocking has the potential to:
   - Achieve cycle to cycle fine grained power down at the stage level.
   - Save area and power by implementing stalls through clock gating without having to introduce extra latches.
   - Make it easier to design for, and cost-effectively implement, pipeline stalls in high-frequency pipelines.
   - Reduce cycle to cycle variance in switching currents.
   - Improve slack on stall signals thereby potentially improving cycle time in designs where stall signals are on the critical path.
2. The interlocking technique improves the storage properties of synchronous pipelines. The storage properties of interlocked synchronous pipelines can:
   - Provide up to two times the normal storage in synchronous pipelines and queues without introducing extra latches.
   - Enable power and area savings by allowing queues to be sized smaller while still providing the same average case performance.
3. The interlocking technique has the potential to enable straightforward mappings between synchronous and asynchronous implementations. Synchronous interlock handshake protocols correspond directly to asynchronous handshake protocols thereby providing the potential to:
   - Facilitate direct translation of synchronous pipeline segments into asynchronous pipeline segments and vice versa.
   - Enable design exploration across synchronous and asynchronous domains to a degree not previously possible.

While all these items are worth discussing in detail, this report will mainly focus on presenting the basics of interlocking synchronous pipelines, the property from which the listed benefits originate.

Outline

This report will first give a brief overview of asynchronous pipelines in Section 2. This is followed by a presentation of traditional synchronous pipelines in Section 3 and how fine grained power down and stalling is typically performed in such pipelines. The elastic synchronous pipelines presented in Section 4 introduces the concept of backward interlocking in ordinary synchronous pipelines such that localized progressive stalls can be implemented. Section 5 extends these elastic pipelines to fully interlocked synchronous pipelines that are interlocked in both the forward and the backward direction. Section 6 discusses the improved storage property of elastic synchronous pipelines and its application to queue structures. Section 7 outlines the potential of performing direct mappings between synchronous and asynchronous implementations of pipelined designs made possible by the introduction of interlocked synchronous pipelines. Section 8 discusses how the presented pipelines were formally verified and how such pipelines can be made testable through scan chains. Section 9 provides conclusions.

2 Asynchronous pipelines

Asynchronous pipelines have several interesting properties that are hard to achieve in synchronous pipelines. Fine grained power down, localized stalls, low cycle to cycle variance in switching currents, and the ability to use transparent latches without risking data races are some of the benefits of asynchronous pipelines. These properties are a result of the way the stages of an asynchronous pipeline are interlocked to their neighboring stages in both the forward and backward direction.

An interlocked asynchronous pipeline (IAP) is illustrated in Figure 1. In an asynchronous pipeline a request signal is used to let a downstream stage know when valid data is available on its inputs and a new computation is required. This provides for a fine grained power down when there is no computation to be performed as no request is generated in such cases. This request signal provides an interlocking in the forward direction of the pipeline. To guard against data races between latches, an

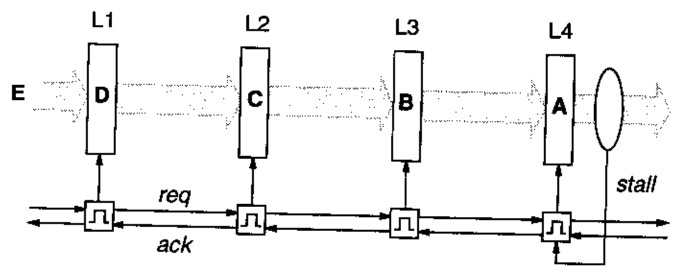


Figure 1. Asynchronous pipeline with data dependent stall condition.

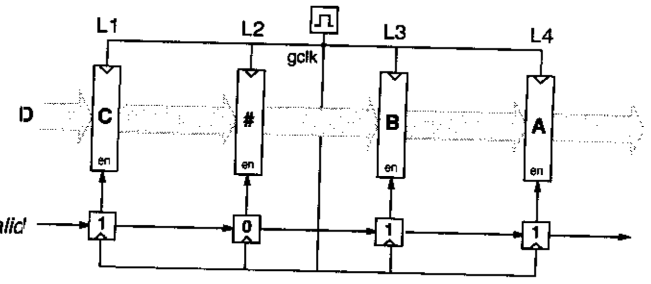


Figure 2. Synchronous pipeline with valid based clock gating.

asynchronous pipeline is also interlocked in the backward direction. This is achieved by an acknowledge signal that is used to let an upstream stage know that data has been safely latched and that new data now can be generated. This way, a bubble is created before data can move between stages, and there is no risk of data races between latches. Since no new data may be generated by the upstream stage until an acknowledge has been received, the upstream stage has to stall until such an acknowledge arrives. This backward interlocking is what provides the beneficial stalling properties of asynchronous pipelines.

Together, the forward and backward interlocking available in asynchronous pipelines provide a powerful technique that not only can achieve fine grained power down and race free latching but also may provide better signal locality and reduced variance in switching currents in pipelines implementing stalls.

3 Synchronous pipelines

Synchronous pipelines traditionally prevent data races between latches, not by interlocking, but by alternating the transparency and opaqueness of latches in adjacent stages. There are two main approaches of this technique. One approach is based on transparent latches where a two-phase clock is used such that only every other pipeline stage is active at a time, the latches in inactive stages are closed (opaque) and act as barriers preventing data races between the open (transparent) latches of active stages. The other approach is to use a one-phase clock with master-slave latches where the master and slave latches are alternating between transparent and opaque modes such that there is never a combinational path between two master latches or two slave latches. It is worth noting the similarity between these two methods. The only fundamental difference being that the two phase pipeline has combinational logic between each array of latches while the one phase pipeline only has combinational logic between slave and master latches. Although the techniques presented in this report are mainly illustrated in the context of two phase pipelines, they work equally well with both types of pipelines. The techniques also work with pipelines based on R/S latches and flip-flops.

3.1 Forward interlocking

Asynchronous pipelines have the potential to reduce power consumption since computations are only performed on demand. A pipeline stage does not perform a computation unless requested to. This is achieved by employing request signals in the forward direction of the pipeline implementing a forward interlocking such that data is only latched when there is a computation to perform.

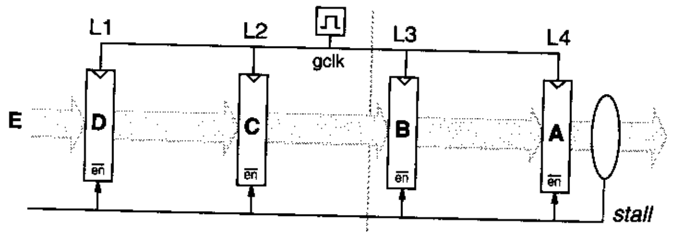
Similar techniques can be used to power down stages in synchronous pipelines. A valid signal that propagates along with the data can be used to indicate when data is valid and a computation should be performed. If the valid signal indicates that data is not valid the clock to the corresponding pipeline stage is gated for the duration of that clock cycle. Subsequently, a computation is performed only when needed. This stage by stage clock gating thus performs a function similar to the request signal in an asynchronous pipeline and achieves a fine grained power down of the pipeline. Figure 2 illustrates a synchronous pipeline with a valid bit that propagates alongside the data in synchronous lock-step and gates the clock when there is no valid data present in a pipeline stage. In the figure, A, B, and C indicates valid data and is accompanied by a valid signal with value 1. The hashmark "#" indicates invalid data and is accompanied by a valid signal with value 0.

The valid signal technique has been explored in the context of saving power in synchronous pipelines [1]. However, to our knowledge, it has not been considered in the context of achieving interlocking between pipeline stages in synchronous pipelines.

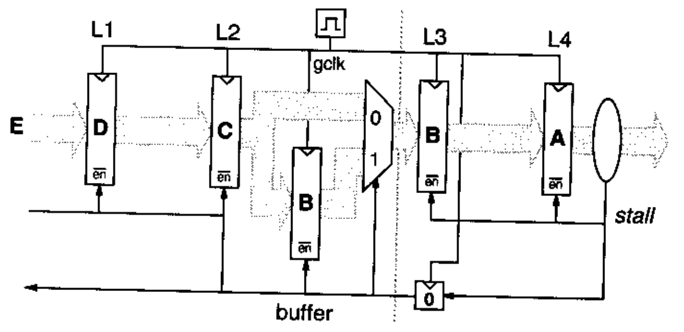
3.2 Backward interlocking

Asynchronous pipelines have the ability to make control decisions regarding the advancement of data on a local basis. Decisions whether to halt or restart a pipeline stage can therefore be done independent from other pipeline stages. Keeping control signals local offers potential benefits. The slack on control signals may be improved as such signals can be kept local thus avoiding distribution over global wires. This is important for stall signals which are often on the critical path. Another benefit from keeping control decisions local is that a pipeline can be brought to a halt and then restarted one stage at a time rather than all stages at once. This may reduce the cycle to cycle variance in switching currents.

In an asynchronous pipeline, the locality of control signals is achieved through stage level interlocking in the backward direction of the pipeline. An acknowledge signal is used to indicate to upstream stages whether the current stage is ready to receive new data or not. By not generating an acknowledge signal in a pipeline stage, the pipeline is brought to a halt, one stage at a



a) Synchronous pipeline with global stall



b) Synchronous pipeline with stall buffer

Figure 3. Synchronous pipeline with and without stall buffer.

time, as no new acknowledges are propagated backward in the pipeline.

Traditionally, synchronous pipelines have been stalled at the global level where all stages of either the entire pipeline, or a multi-stage unit, are stalled at the same time. Lately however, cycle time and switching current issues have started placing restrictions on how many stages can be stalled during the same cycle. This is due to problems caused by delays on long wires and significant cycle to cycle variance in switching capacitance. The difficulty with stalling synchronous pipelines progressively, at a finer granularity, is that data is lost at stall boundaries. Figure 3(a) illustrates this problem in a synchronous pipeline with master-slave latches. The dotted line in the figure illustrates a stall boundary. The stall boundary indicates the place in the pipeline where upstream stages will not receive the stall signal in time before the next clock edge arrives due to cycle time constraints. While the stages downstream of the stall boundary receive the stall signal and correctly come to a halt, the stages upstream of the boundary do not see the stall signal in time and therefore latch new data as usual. In the figure, as stage 3 is stalled, it will keep data item B. Stage 2, however, does not see the stall signal in time and will therefore latch new data the next clock cycle. The result is that data item C that is stored in stage 2 will be overwritten and lost.

Traditional approaches to handle stalls have been to insert buffer stages in parallel to the pipeline [2] at stall boundaries. The buffer stage is used to temporarily store the data that would otherwise be overwritten. An example of this is illustrated in Figure 3(b). Stall boundaries have traditionally been introduced at the unit level as stalls are relatively easy and cost-effective to handle at that level of granularity. However, stalling at the unit level may result in large cycle to cycle variances in switching currents. Signal slack may also be impacted due to the stall signal having to propagate throughout the unit over long wires. As technology scales, with relative increase in wire delays and demand for shorter cycle times, stall signals cannot propagate as far. As a result, buffer stages may have to be introduced at a finer granularity. At the finest level of granularity, where stalling is performed on a stage by stage basis, introducing extra buffer stages would double the number of latches in a pipeline. Clearly, this approach is not cost-effective in terms of area and power at more fine grained levels.

As illustrated by the above examples, implementing cost-effective stalls in synchronous pipelines, not to mention finding a solution to backward interlocking, is a difficult problem. To our knowledge, no one has investigated how synchronous pipelines can be made to work with fine grained stalls without inserting extra buffer stages. In the following section we will look into how ordinary synchronous pipelines can be made to implement backward interlocking and thereby achieve cost-effective progressive stalling at the stage level, without the need for extra buffer stages.

4 Elastic synchronous pipelines (ESP)

Just as forward interlocking, that is, the decision to clock gate based on valid bits propagating in the forward direction of the pipeline can be performed at a local level, stage by stage, we would like to achieve a similar interlock in the backward direction thus allowing stalls to take place locally on a stage by stage basis.

The approach to achieve backward interlocking in synchronous pipelines presented in this report implements the ability to reuse latches already present in the pipeline to act as buffer stages during stalls. This section will first illustrate how this can be achieved in a synchronous system through sequential storage in a pair of latches. Application to pipelines and queues will be considered later in this section.

4.1 Sequential read and store in adjacent latches

Consider how two data items can be stored in a pair of latches, L1 and L2, connected in serial as illustrated in Figure 4. The latching of data is governed by a synchronous clock. For the sake of simplicity we assume that the latches open and close on opposite edges of the clock as in, for example, adjacent stages of a two phase clocked pipeline. The sequential storage of data in the latches is achieved through clock gating. Consider how two data items A and B can be sequentially stored in, and then read from, the latches.

Storing A and B. Assume the clock is low, latch L1 is open, and latch L2 is closed. Data item A is applied to the input of L1. When the next rising edge *e1* of the clock arrives, latch L1 will close and store data item A and latch L2 will open up. When the next falling edge *e2* of the clock arrives, latch L2 will close and store data item A and latch L1 will open up. Data item B is now applied to the input of latch L1. Assume that the clock to latch L2 is now gated. When the next rising edge *e3* of the clock arrives, latch L2, which is now gated, will remain closed and

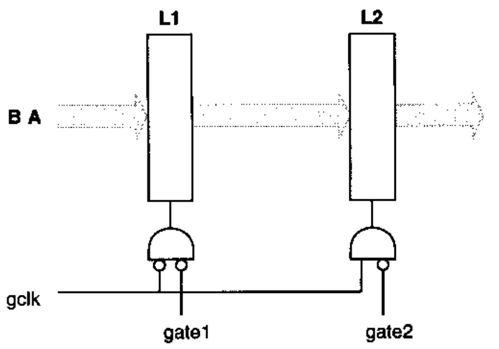


Figure 4. Latch pair.

continue to store data item A. Latch L1 will close and store data item B. Assume that the clock to latch L1 is now gated. When the next falling edge *e4* of the clock arrives, latch L1, which is now gated, will remain closed and continue to store data item B. The data values A and B are now held in latches L2 and L1 respectively until the clock to the latches is ungated again.

Reading A and B. Assume the clock is low and latches L1 and L2 both are gated. Assume the clock to latch L2 is now ungated. When the next rising edge *e5* of the clock arrives, the environment latches will close and store data item A from the output of latch L2. Latch L2 which is no longer gated will open, thus no longer storing data item A. Assume the clock to latch L1 is now ungated. When the next falling edge *e6* of the clock arrives, latch L2 will close and store data item B. Latch L1 which is no longer clock gated will now open up, thus no longer storing data item B. When the next rising edge *e7* of the clock arrives, the environment latches will close and store data item B from the output of latch L2.

In the described fashion, any two data items can be sequentially stored and read in a pair of latches, even when the latches are adjacent and clocked by the same synchronous clock. As we will see later, this way of storing data items plays a fundamental part of the elastic nature of the pipelines presented in this report.

4.2 Application to pipelines and queues

Sequential storing of data through clock gating is applicable also to pipelines and form the fundamental basis of how to achieve backward interlocking in a synchronous pipeline. The backward interlocking is based on letting each stage generate a stall signal to its upstream neighbor, indicating when the stage is not ready to receive new data.

Consider a two phase clocked synchronous pipeline based on transparent latches. In such a pipeline adjacent stages are not active simultaneously. When a given stage is computing the adjacent upstream and downstream stages are idle. The latches of active stages are transparent and the latches of idle stages are opaque. As a result, only every other stage stores data at any given time. It is important to note that while idle stages store data in their latches, that data is no longer useful as the data has already moved on to the next, active, stage in the pipeline. Subsequently, only active stages contain data while idle stages contain bubbles. This is a fundamental property of the elastic pipelines presented in this report since this means that half of the stages in a synchronous pipeline are "empty" and can potentially be used as buffer stages to stall the pipeline progressively. Let's take a closer look at how an ordinary synchronous two phase pipeline can be made elastic.

Under normal operating conditions, the data latches for an active stage are open. When such a stage generates a stall signal we close the data latches the next clock edge and make sure they remain closed until the stall condition goes away. Closing the data latches is achieved by gating the clock with the stall signal. The stall signal in turn is propagated backward in the pipeline and is kept in synchronous lock-step to the pipeline by latching it at each pipeline stage. The stall signal thus propagates only one stage per clock edge, and is thereby kept local to each stage. Note that the stall signal latches themselves are not clock gated. As outlined above, it is sufficient to add a latch and a gating function to each pipeline stage in order to transform an ordinary two phase clocked synchronous pipeline into an elastic pipeline.

Stalling an elastic pipeline

Consider the two phase pipeline example in Figure 5. Note that the latches used to propagate the stall signal backward in the pipeline are clocked on the opposite edge of the data latches of their associated stage. Also note that the delay gates normally used to balance the data and stall latch clocks are not shown. The data latches of each stage are clock gated by the output of their associated stall latch. Consider the waveform trace illustrated in Figure 6. The figure illustrates the clocks, stall, and data signals for each stage. The characters in the data waveforms illustrate distinct data items as they move through the pipeline. A box containing a character indicates that the data latches of the stage are closed and that the corresponding data item is currently stored in that stage. A line indicates that the data latches are open and not storing data.

The sub-trace between the dotted lines is illustrated in more detail in Figure 7. In the sub-trace the data stream A,B,C,D,E is applied to the pipeline. The bold text in the trace indicates when data (or stall) is stored in the corresponding latch, i.e., the latch is opaque. This includes clock gated conditions. Grey non-bold text indicates that data is propagated through the latch, i.e., the latch is transparent. The shaded polygon in the trace illustrates how the stall condition propagates backward through the pipeline. The shaded polygon corresponds to the similarly shaded regions in Figure 6. The stall condition can be thought of as a sliding window moving in the backward direction of the pipeline. Outside the window, data is stored in every other pipeline stage as normal for a two phase pipeline. Within the window, data is "compacted" such that data is stored in every pipeline stage. This ability to on demand increase the amount of storage available in the pipeline is what prompted the name elastic pipelines.

Returning to the trace in Figure 7, assume the clock is high and the pipeline is in steady state operation with two data items continuously present in the pipeline. The data latches in stages 1 and 3 at this time are closed and store data items B and A

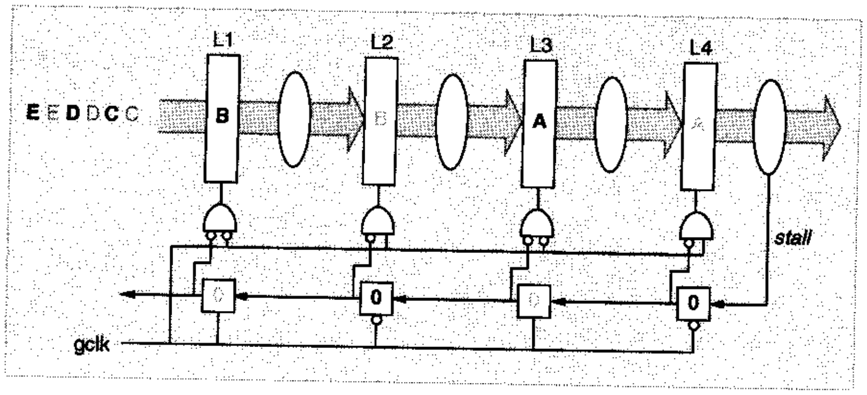


Figure 5. Two-phase clocked elastic synchronous pipeline implementing progressive stall through backward interlock.

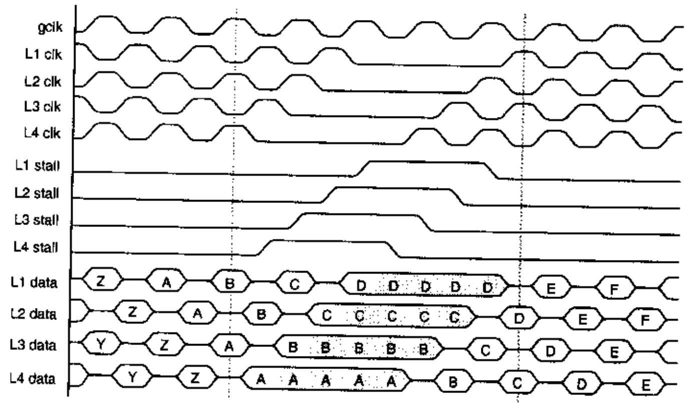
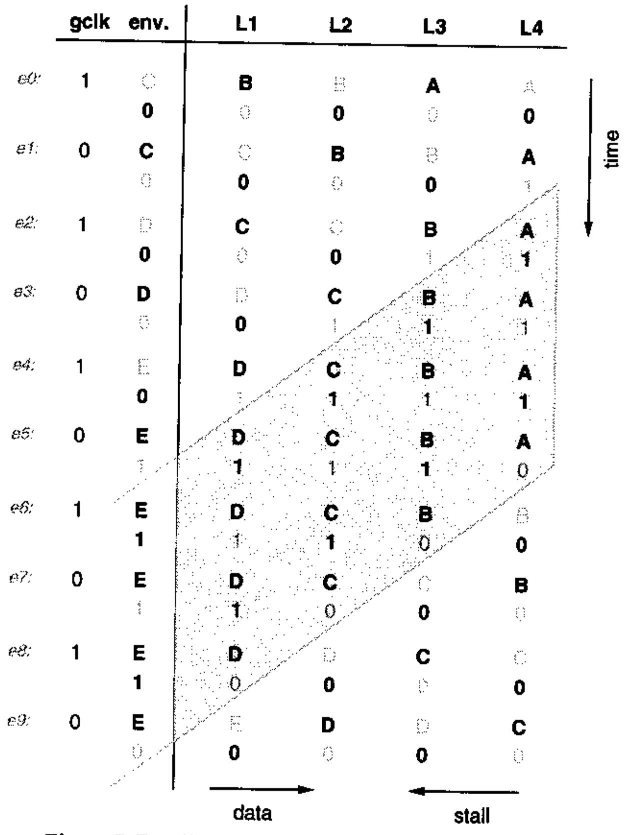


Figure 6. Waveform trace of two-phase clocked ESP.



	gclk	env.	L1	L2	L3	L4
e0:	1	C	B	B	A	A
		0	0	0	0	0
e1:	0	C	C	B	B	A
		0	0	0	0	1
e2:	1	D	C	C	B	A
		0	0	0	1	1
e3:	0	D	D	C	B	A
		0	0	1	1	1
e4:	1	E	D	C	B	A
		0	1	1	1	1
e5:	0	E	D	C	B	A
		1	1	1	1	0
e6:	1	E	D	C	B	B
		1	1	1	0	0
e7:	0	E	D	C	C	B
		1	1	0	0	0
e8:	1	E	D	D	C	C
		1	0	0	0	0
e9:	0	E	E	D	D	C
		0	0	0	0	0

time
data
stall

Figure 7. Detailed sub-trace of two-phase clocked ESP.

respectively. The data latches for stage 2 and stage 4 are at this point open and do not store any data.

Clock edge 1. Once the next falling clock edge *e1* arrives, the data latches of stages 2 and 4 close, and the data latches of stages 1 and 3 open. Stage 2 now stores data item B, and stage 4 stores data item A. At the same time, *e1* causes the stall latches of stages 2 and 4 to open, and the stall latches of stages 1 and 3 to close. Assume that signal *stall* is now asserted. Since the stall latch of stage 4 is open, *stall* is propagated to stage 3 and to the gating function controlling the clock to the data latches of stage 4.

Clock edge 2. The gating function makes sure that the data latches of stage 4 remain closed when the next rising clock edge *e2* arrives. Stage 4 will thus continue to store data item A. At the same time the data latches of stages 1 and 3 will close and store data item C and B respectively, and the data latches of stage 2 will open. The stall latch of stage 3 will open and propagate the asserted stall signal to stage 2 and to the clock gating function of stage 3.

Clock edge 3. When the next falling clock edge *e3* arrives, the data latches of stage 3 will remain closed and continue to store data item B. Similarly stage 4 will continue to store data item A. The data latches of stage 2 will in turn close and store data item C, and the data latches of stage 1 will open. The stall latch of stage 2 will open and propagate the asserted stall signal to stage 1 and to the gating function of stage 2.

Clock edge 4. When the next rising clock edge *e4* arrives, the data latches of stage 2 will remain closed and continue to store data item C. Similarly the data latches of stage 3 and 4 will remain closed and continue to store data items B and A. At the same time, the data latches of stage 1 will close and store data item D. The stall latch of stage 1 in turn opens and propagates the asserted stall signal to the environment and to the gating function of stage 1. The whole pipeline has now been safely stalled without losing any data items. All data latches in the pipeline are now filled with valid data items.

What is the fundamental reason that a pipeline can be progressively stalled this way? By filling in the bubbles that always exist in two phase pipelines with data items we hide the "delay" of propagating the stall signal backward in the pipeline one stage at a time. There are N/2 data items present in the pipeline to start with, and there are N number of stages in the pipeline. There are thus N/2 empty stages that can be used as buffers. It will take the stall signal N clock edges to propagate to the start of the pipeline. During this time, N/2 new data items will enter the pipeline (in a two-phase pipeline new data enters the pipeline only every other clock edge). Subsequently, there is enough buffer storage such that all data can be safely stored. When all stages have stalled, all latches in the pipeline are closed and have valid data stored in them. There are thus N valid data items in the pipeline.

Unstalling an elastic pipeline

Similar to stalling the pipeline, unstalling the pipeline opens up the latches one stage at a time, recreating the bubbles in the pipeline such that no data is lost when data starts moving through the pipeline again. Once the whole pipeline is unstalled the occupancy of the pipeline is again down to N/2 data items.

Assume we want to unstall the pipeline that was stalled in the previous subsection. After the whole pipeline had been stalled, the clock was left high and stages 1, 2, 3, and 4 were currently clock gated and stored data items D, C, B, and A respectively.

Clock edge 5. When the next falling clock edge *e5* arrives, the data latches of all stages will remain gated (and thus closed) and continue to store their respective data items. The stall latches of stages 2 and 4 will open, and the stall latches of stages 1 and 3 will close. Assume that signal *stall* is now deasserted. This implies that the condition causing the stall has gone away and that the environment will be ready to latch the data output of stage 4 the next clock edge. The deasserted stall signal will propagate through the open stall latch of stage 4 to stage 3 and to the clock gating function of stage 4. The clock to the data latches of stage 4 is thus enabled again.

Clock edge 6. When the next rising clock edge *e6* arrives, the data latches of the environment will close and store data item A. The data latches of stage 4 are no longer gated and will open up, thus no longer storing data item A. The data latches of stages 1, 2, and 3 still remain gated. The stall latches of stages 1 and 3 will open, and the stall latches of stages 2 and 4 will close. The deasserted stall signal will propagate through the open stall latch of stage 3 to stage 2 and to the clock gating function of stage 3. The clock to the data latches of stage 3 is thus enabled again.

Clock edge 7. When the next falling clock edge *e7* arrives, the data latches of stage 4 will close and store data item B. The data latches of stage 3 are no longer gated and will open up, thus no longer storing data item B. The data latches of stages 1 and 2 still remain gated. The stall latches of stages 2 and 4 will open, and the stall latches of stages 1 and 3 will close. The deasserted stall signal will propagate through the open stall latch of stage 2 to stage 1 and to the clock gating function of stage 2. The clock to the data latches of stage 2 is thus enabled again.

Clock edge 8. When the next rising clock edge *e8* arrives, the data latches of stage 3 will close and store data item C. The data latches of stage 2 are no longer gated and will open up, thus no longer storing data item C. The data latches of stage 1 still remain gated. The stall latches of stages 1 and 3 will open, and the stall latches of stages 2 and 4 will close. The deasserted stall signal will propagate through the open stall latch of stage 1 to the environment and to the clock gating function of stage 1. The clock to the data latches of stage 1 is thus enabled again.

Clock edge 9. When the next falling clock edge *e9* arrives, the data latches of stage 2 will close and store data item D. The data latches of stage 4 will close and store data item C. The data latches of stage 1 are no longer gated and will open up, thus no longer storing data item D. The data latches of all stages have now been ungated and the pipeline has again reached normal steady state operation.

5 Interlocked synchronous pipelines (ISP)

Given an approach for forward interlock through valid based clock gating (Section 3.1) and a backward interlock based on elastic pipeline properties (Section 4), the next step is to merge these approaches into fully interlocked synchronous pipelines. Such pipelines have the computing on demand and localized progressive stall properties of asynchronous pipelines while still being driven by a synchronous clock.

Figure 8 illustrates an interlocked synchronous pipeline. Valid bits are propagated in the forward direction of the pipeline. These valid bits indicate when the associated data is valid and a computation should be performed. In this respect, the valid bit is the equivalent of the request signal used in asynchronous pipelines. Stall bits are propagated in the backward direction of the pipeline. These stall bits indicate when the pipeline must halt, for example due to access conflicts to a shared resource. The stall bit is the equivalent of the (inverse) acknowledge signal used in asynchronous pipelines. Note that in its purest sense, the interlocking referred to in a synchronous pipeline is the interlock of the valid and stall bit with respect to the clock, and not between the valid and stall bits themselves. That said, the effect of this interlocking is very similar to the effect of interlocking the request and acknowledge signals in an asynchronous pipeline.

Note that during a stall condition, the valid bit latch must be clock gated together with the data latches. This is necessary in order to correctly propagate the valid bit along with its associated data. While stall latches in turn could be clock gated based on the valid bits for reasons of power saving, this is not a necessary condition for correct operation. To simplify the discussion, the clock to the stall latches in Figure 8 is therefore left running free. Also note that the delay gates normally used

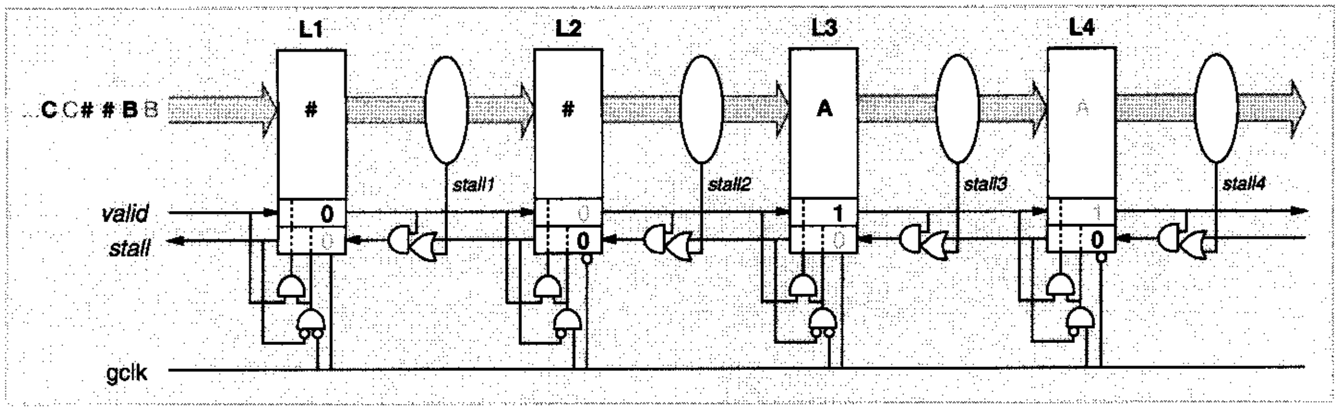


Figure 8. Two-phase clocked interlocked synchronous pipeline implementing forward and backward interlock.

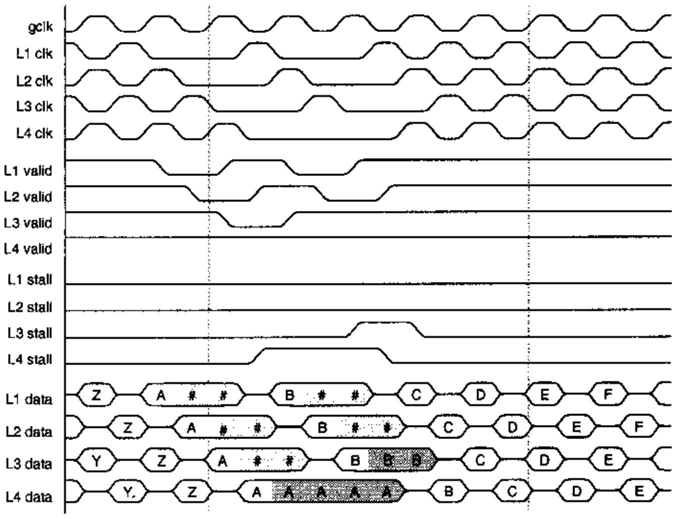


Figure 9. Waveform trace of two-phase clocked ISP.

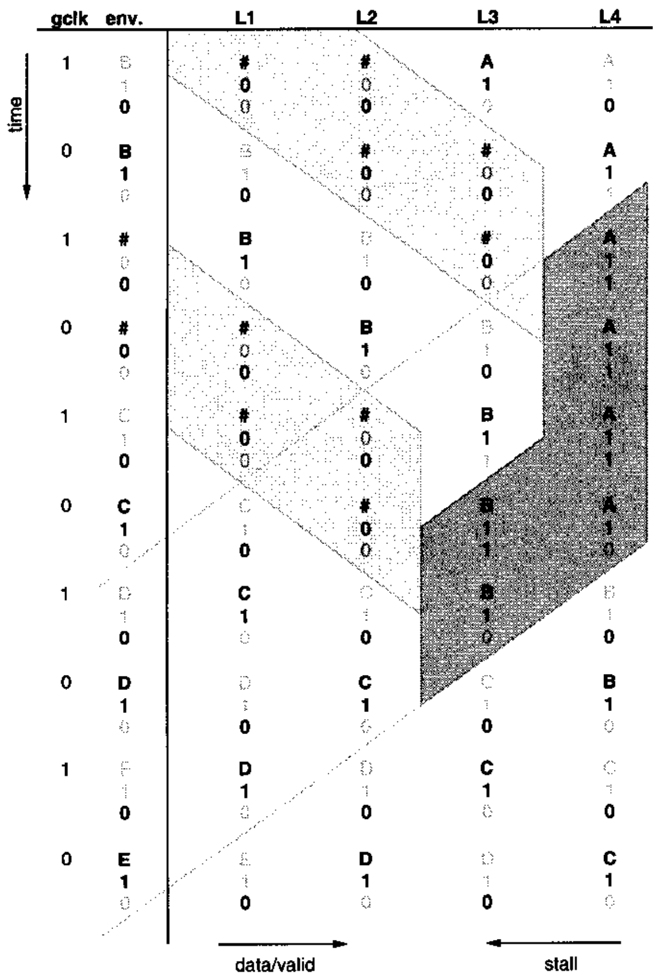


Figure 10. Detailed sub-trace of two-phase clocked ISP.

to balance the data, valid, and stall latch clocks are not shown in the figure. The main contribution of the valid bits, besides providing fine grained power down, is that they indicate holes in the pipeline. When propagating a stall condition backward along the pipeline, the stall signal does not need to be propagated further as long as there is no valid data that needs to be stalled. The valid bit, when not asserted, can therefore be used to override the stall bit. Shorter stall conditions may therefore not have to propagate to the beginning of the pipeline, thus not having to stall the environment unless the pipeline completely fills up.

When considering pipelines where we want to fill in holes in the pipeline during stall conditions, and where multiple stages in the pipeline can generate their own stall condition, it is easy to realize the benefits of localized stalls. When filling in the holes in a pipeline extra logic is required on the stall signal wire to detect these holes. To fill in holes, the stall signal must be AND:ed with the local valid signal of each stage as it progresses backward in the pipeline. Furthermore, additional logic is needed if each stage can generate its own stall condition, in which case the global and local stall conditions must be OR:ed together at each stage. In addition to the long wire delays when stalling a pipeline globally, there is now added gate delays that grow linearly with number of stages in the pipeline. When stalling multi-stage pipelines globally, the extra wire and logic delay of the stall signal will eventually start impacting the cycle time. In an interlocked pipeline on the other hand, the logic needed to detect holes and support generation of stalls in each stage is kept local to the stage, adding only one gate delay to the stall signal.

5.1 Interlocked pipeline operation

Consider the interlocked pipeline in Figure 8. Consider the waveform trace illustrated in Figure 9. The figure illustrates the clocks, valid, stall, and data signals for each stage. The characters in the data waveforms illustrate distinct data items as they move through the pipeline. A box containing a character indicates that the data latches of the stage are closed and that the corresponding data item is currently stored in that stage. A line indicates that the data latches are open and not storing data.

The sub-trace between the dotted lines of Figure 9 is illustrated in more detail in Figure 10. In the sub-trace, the data stream A,#,B,#,C,D,E, where "#" represents invalid data (a hole), is applied to the pipeline. Note that invalid data is not propagated through the pipeline due to the valid bit based clock gating, but rather is created in place when the corresponding valid bit turns zero. The bold text in the trace indicates when data (or valid/stall) is stored in the corresponding latch, i.e., the latch is opaque. This includes clock gated conditions. Grey non-bold text indicates that data is propagated through the latch, i.e., the latch is transparent. The two light-grey polygons to the left in the trace illustrate how the clock gated conditions due to data being invalid propagates forward in the pipeline. The rightmost, dark-grey, polygon in the trace illustrates how the clock gated condition due to the stall propagates backward in the pipeline. These shaded polygons correspond to the similarly shaded regions in Figure 9.

Assume the clock is high and the pipeline is in steady state operation with two data items (or holes) continuously present in the pipeline. When data item A reaches stage 4 a stall will be generated for two consecutive clock cycles. The stall condition is illustrated by the dark-grey polygon of the trace in Figure 10. In an elastic pipeline, the stall condition would propagate backward in the pipeline as outlined by the dotted lines. Each stage in the pipeline would have stalled for two cycles. However, in an interlocked pipeline the valid bits indicate holes in the pipeline. When a hole is encountered, the valid bit overrides the stall condition by zeroing out the stall signal. This is illustrated by the stall window (dark-grey polygon) in the trace getting truncated when it encounters the invalid windows (light-grey polygons). The override in turn cancels out the invalid condition when the hole gets filled with valid data. This is illustrated by the invalid windows getting truncated when they encounter the stall window in the trace.

The data stream in Figure 10 contains two holes, one after data item A and one after data item B. In an interlocked pipeline, rather than stalling all stages for two cycles, stage 4 will stall for two cycles, while stage 3 stalls for only one cycle, and stages 2 and 1 do not stall at all. The reason for this is that the stall condition is shortened by one cycle at stage 3 where the invalid data following A overrides the asserted stall signal in order to fill in the hole in the pipeline. The first cycle of the two cycle long stall window is therefore zeroed out and is not propagated backward in the pipeline. Rather than being stalled in stage 2 for two cycles, data item B is therefore instead propagated to stage 3 and stalled for only one cycle, thereby filling up the hole in the pipeline. Similarly, as the remaining second cycle of the stall window reaches stage 2 the invalid data following data item B zeroes out the stall window completely. Thanks to the holes in the pipeline the stall condition never reaches the start of the pipeline and the environment does not need to stall. Data items C and D in the data stream are therefore not stalled but rather propagate through the pipeline in a normal fashion.

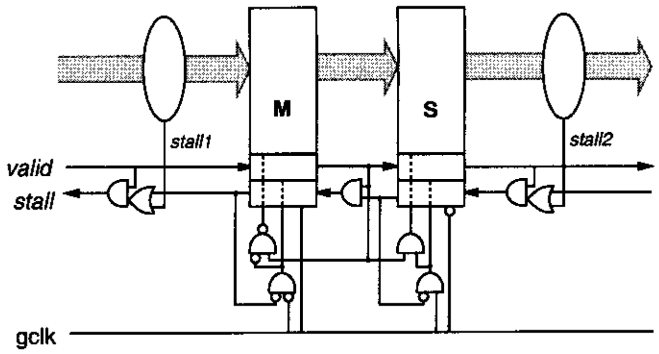


Figure 11. Master-slave based interlocked synchronous pipeline.

5.2 Application to one-phase clocked pipelines

While the elastic and interlocked pipelines have been presented in the context of two-phase clocked pipelines, the techniques work equally well for one-phase clocked pipelines based on master-slave latches. Consider the interlocked pipeline in Figure 8. The pipeline is based on a two-phase clock and has combinational logic between each array of latches. If the combinational logic was removed between latches L1 and L2, and between latches L3 and L4, the pipeline would implement the behavior of a one-phase clocked master-slave pipeline. Latches L1 and L2 would then form one master-slave pair and latches L3 and L4 the other. Note that the OR gate to merge the local and global stall signals is not needed between master and slave latches. The AND gate between the master and slave latches, however, is still needed in order to detect the presence of a hole in the master latch.

In a one-phase pipeline the decision to clock gate is made at the end of each clock cycle as opposed to at the end of each half-cycle as in a two-phase pipeline. Compared to a two-phase pipeline, the clock at the master latch in a one-phase pipeline is therefore one half-cycle out of phase with respect to the valid signal. The output inverter of the AND gating function of the master data latch clock must therefore be moved to the clock input of the AND gate such that the signals are in phase. However, this makes the gating function susceptible to glitches on the valid signal during the first half of the clock cycle. The valid signal to the gating function of the master clock is therefore first filtered through a master latch. The result is that the master and slave clocks both use the same phase of the valid signal. Figure 11 illustrates the interlock logic for a single stage in a one-phase clocked master-slave based pipeline.

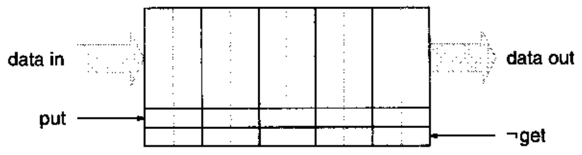


Figure 12. Elastic synchronous queue.

6 Storage properties of elastic pipelines

It is interesting to note that the storage capacity and latency of the presented elastic and interlocked synchronous pipelines varies dynamically with the input/output rate of data items. The elastic property can be used advantageously to store more data than what is normally possible in synchronous pipelines and queues.

Consider the master-slave based elastic queue structure in Figure 12. When a data item is inserted in the queue an associated put signal is asserted. This put signal corresponds to the valid signal discussed in previous sections. Each cycle that no data item is taken out of the queue, a ¬get signal is asserted. An asserted ¬get signal indicates a stall condition. Just as an interlocked synchronous pipeline, the elastic queue makes use of the put signal to override the ¬get signal such that holes in the queue are filled with valid data. Note that the put signal is not necessary to achieve the improved storage capacity of an elastic queue, but is beneficial for purposes of filling in holes and saving power through clock gating.

Figure 13 illustrates the storage capacity of the elastic queue under different input/output rates. If the input rate to an N stage queue is x and the output rate is 0, then the queue will store 2*N data items and have a latency of 2*N cycles. If the input rate is x and the output rate is x/2, then the queue will store a maximum of 3/2*N data items and have a latency of 3/2*N cycles. If the input rate is x and the output rate is x, then the queue will store N data items and have a latency of N cycles. The queue dynamically adapts the storage capacity to match the output rate, providing more storage as the output rate slows down. If the receiver slows down and more storage is needed, the queue size "grows", automatically providing the needed storage. As the receiver speeds up, the queue size shrinks, and latency through the queue is reduced. A normal synchronous queue would have to stall the sender when 100% capacity is reached, while an elastic synchronous queue can achieve a storage of up to 200% before the sender needs to be stalled. Note that above 100% storage capacity, the latency through the elastic queue is directly proportional to the occupancy of the queue. The proposed elastic queue structure provides twice the maximum storage of a normal queue at a cost of one extra latch per stage (not including optional latch for put signal). An elastic queue can thus provide 2*M bits of storage in an M+1 bit queue stage.

The extra storage capacity of elastic pipelines has the potential to allow queues in a system to be sized considerably smaller than normal. Utilizing elastic queues could therefore save significant area and power while still providing the same average performance. The elastic storage properties can also be used advantageously in ordinary pipelines where the extra storage capacity of each stage may reduce the need to insert explicit buffer stages in the pipeline.

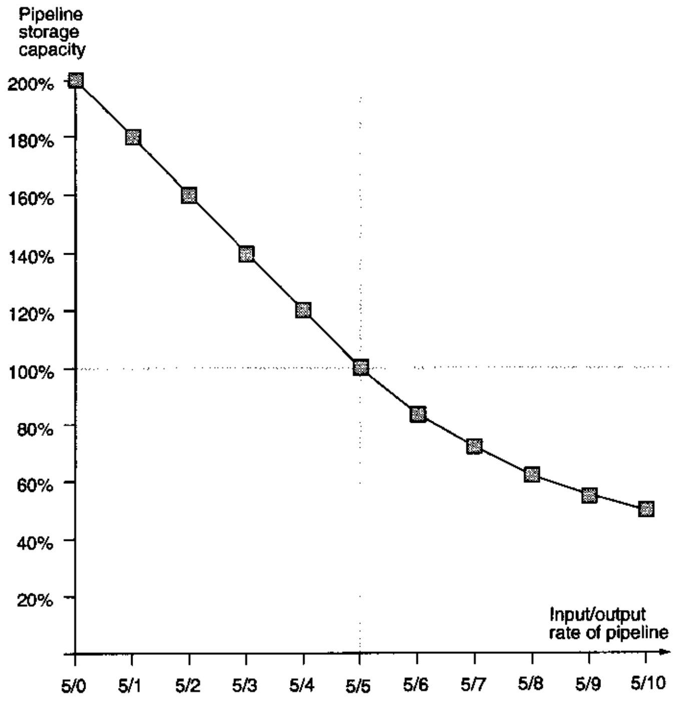


Figure 13. Storage capacity of an elastic synchronous queue.

7 Mapping between synchronous and asynchronous pipelines

The presented interlocking technique of the interlocked synchronous pipeline has a behavior very similar to interlocked asynchronous pipelines. The handshake protocols used to implement interlocking in asynchronous pipelines and the presented synchronous pipelines perform the same functionality. While the type of signals used for the handshake may differ, the synchronous interlocking makes use of signal levels while asynchronous interlocking typically makes use of signal events or pulses, the purpose of the handshakes is still the same, namely to base the local clocking decision of a pipeline stage upon the state of its neighboring upstream and downstream stages. At the conceptual level, the request-acknowledge handshake used in asynchronous pipelines therefore directly corresponds to the valid-stall handshake used in the interlocked synchronous pipelines. This correspondence has the potential to facilitate a direct mapping between synchronous and asynchronous implementations of pipelined designs.

The flow of data through an asynchronous pipeline is determined by the sequencing of request signals. As data and its associated request signal propagates through an asynchronous pipeline local control decisions are made in each stage. These control decisions are used to "steer" the request signal to the next desired destination of the data. When receiving a request signal, the destination latches the associated data and in turn decides where the data should be sent next. The flow of request signals in an asynchronous pipeline can be directly replicated in an interlocked synchronous pipeline through the use of valid bits. In a similar fashion, the flow of acknowledge signals in an asynchronous pipeline can be replicated through the use of stall bits. The activation, or "clocking", of a pipeline stage in an asynchronous pipeline can therefore be directly mapped into the activation of a synchronous pipeline stage and vice versa.

Most pipeline structures such as branches, forks, and joins should be possible to directly map between interlocked synchronous and asynchronous pipelines. Arbitration structures, however, need to be more carefully examined due to the unbounded delay of such structures in asynchronous circuits. While mapping of asynchronous arbitration into synchronous arbitration is possible, the reverse might not be when synchronous and asynchronous pipeline segments are mixed in the same design. Another circuit structure that needs to be examined more closely is precharged logic. When used in datapaths, such logic is typically precharged by using the acknowledge signals from downstream stages. In an interlocked synchronous pipeline the precharge would have to be mapped to a combination of the clock and stall bits.

In order to automate the mapping between synchronous and asynchronous pipelines, tools that support extraction of matching datapath delays and replacement between master-slave and simple transparent latches must be developed. Once such tools are in place, the ability to map between synchronous and asynchronous pipelines would enable design exploration across synchronous and asynchronous domains to a degree not previously possible.

8 Verification and testing

The pipelines presented in this report have been formally verified to ensure functional and timing correctness of the implementation. Strategies for supporting testability have also been developed.

8.1 Verification

Prototypes of the proposed pipeline structures have been implemented at the transistor level in a 0.13um SOI process. The pipelines were simulated using SPICE and shown to operate correctly under a five corner model. To ensure correct behavior of signal changes over all possible timings the pipelines were also formally verified using timed state space exploration.

The timed circuit tool ATACS [3] was used as a synthesis and verification engine to formally verify the clock gating circuits of the pipelines. For verification, a behavioral specification of the pipeline with assumed timing bounds and required safety properties such as timed causality and glitch-free operation of the clocks is first constructed. Compliance with the specified safety properties is then verified through timed state space exploration. The specification is then synthesized from the enumerated timed state space. Actual timing bounds of the resulting gate netlist are then obtained from gate libraries. If the actual timing bounds are outside of the assumed timing bounds, verification is repeated with the actual timing bounds. Otherwise the circuit is correct by construction.

The elastic pipeline in Figure 5 and the interlocked pipeline in Figure 8 were both verified using this technique. The state machine descriptions of the interlocking mechanisms were synthesized and mapped to gate level implementations. The synthesized implementations were found to directly correspond to the interlocking logic illustrated in Figures 5 and 8 respectively. The resulting implementations were shown to be hazard free, and thus, correct by construction. The verification shows that the pipelines operate correctly under synchronous timing assumptions.

8.2 Testability

Support for testability is a requirement when designing commercial chips. Consider a one phase clocked master-slave based elastic pipeline. A master-slave latch supporting LSSD with sequential scan chains usually contains three latches, a master latch, a slave latch, and a scan latch. During scan-in and scan-out of test patterns, the master latch is closed and the scan latch instead acts as a master latch. This type of LSSD latch supports scan-in and scan-out of one data item, typically stored in the master latches. When an elastic pipeline is stalled, however, both master and slave latches contain data. This poses a challenge in how scan-in and scan-out of data can be performed.

During scan-in it may be desirable to support the ability to create a situation where the pipeline is stalled to start with. Similarly, during scan-out support for scanning both master and slave latches is needed. However, a three latch LSSD master-slave latch only supports scan-in and scan-out of one data item. The most straight forward way to implement scan-out ability of both master and slave latches is to run identical test vectors twice, the first time scanning out data in the master latches, and the second time scanning out data in the slave latches. This solution is cost-effective in terms of hardware, but increases testing time. Another solution is to add extra scan latches that allow two data values to be scanned out per master-slave latch. Achieving scan-in to both master and slave latches is a more difficult matter. There are several solutions to this problem including adding extra scan latches, or simply to scan-in test patterns that will create the desired stall during runtime. Which approach is most cost-effective is a tradeoff between hardware complexity versus test runtime and the complexity of creating test patterns. While it is not yet clear which of these methods provides the most cost-effective solution, it is clear that the pipelines presented in this report can be made fully testable using sequential LSSD techniques.

9 Conclusions

This report has presented a novel technique, interlocked synchronous pipelines, that achieves interlocking between stages in a synchronous pipeline. Interlocked synchronous pipelines use valid bits propagating in synchronous lock-step with the data in the forward direction of the pipeline, and stall bits propagating in synchronous lock-step in the backward direction of the pipeline, to achieve interlocking between pipeline stages. The stages of an interlocked pipeline computes only on demand, on a stage by stage basis. This is achieved through clock gating based on the valid bits. The stages of an interlocked pipeline stall on a local basis, stage by stage. This is achieved through clock gating in the backward direction of the pipeline based on the stall bits. Progressive stalling has the potential to reduce variance in switching currents and to increase slack on the stall signals. The elastic nature of interlocked synchronous pipelines provides improved storage properties over normal synchronous pipelines. Elastic queues can therefore be sized smaller, thereby saving area and power, while still providing the same average performance. The presented techniques works equally well for two phase clocked pipelines based on transparent latches as well as one phase clocked pipelines based on master-slave latches.

The interlocked synchronous pipelines presented in this report demonstrate a first step towards a middle ground between asynchronous and synchronous pipelines. The interlocked synchronous pipelines can, in a synchronous context, achieve several beneficial properties previously only found in asynchronous pipelines.

Acknowledgements. The authors would like to thank Wendy Belluomini, Phil Emma, Bruce Fleischer, David Kung, Phil Restle, and Leon Stok for many helpful discussions.

References

[1] GOWAN, M., BIRO, L., AND JACKSON, D. Power considerations in the design of the Alpha 21264 microprocessor. In *Proc. ACM/IEEE Design Automation Conference* (1998), pp. 726–731.

[2] MCLELLAN, E. J. Reducing stall delay in pipelined computer system using queue between pipeline stages. *Digital Equipment Corporation, U.S. patent 5325495* (1994).

[3] MYERS, C. J., BELLUOMINI, W., KILLPACK, K., MERCER, E., PESKIN, E., AND ZHENG, H. Timed circuits: A new paradigm for high-speed design. In *Proc. of Asia and South Pacific Design Automation Conference* (Feb. 2001), pp. 335–340.

[4] SCHUSTER, S., REOHR, W., COOK, P., HEIDEL, D., IMMEDIATO, M., AND JENKINS, K. Asynchronous interlocked pipelined CMOS circuits operating at 3.3-4.5 GHz. In *International Solid State Circuits Conference* (2000), pp. 292–293.

[5] SUTHERLAND, I. E. Micropipelines. *Communications of the ACM 32*, 6 (June 1989), 720–738.