Timing Constraints for High Speed Counterflow-Clocked Pipelining

Jae-tack Yoo, Ganesh Gopalakrishnan and Kent F. Smith

UUCS-95-019

Department of Computer Science
MEB 3190, University of Utah
Salt Lake City, UT. 84112

October 30, 1995

Abstract

With the escalation of clock frequencies and the increasing ratio of wire- to gate-delays, clock skew is a major problem to be overcome in tomorrow's high-speed VLSI chips. Also, with an increasing number of stages switching simultaneously comes the problem of higher peak power consumption. In our past work, we have proposed a novel scheme called Counterflow-Clocked(C^2) Pipelining to combat these problem, and discussed methods for composing C^2 pipelined stages. In this paper, we analyze, in great detail, the timing constraints to be obeyed in designing basic C^2 pipelined stages as well as in composing C^2 pipelined stages. C^2 pipelining is well suited for systems that exhibit mostly uni-directional data flows as well as possess mostly nearest-neighbor connections. We illustrate C^2 pipelining on such a design with several design examples. C^2 pipelining eases the distribution of high speed clocks, shortens the clock period by eliminating global clock signals, allows natural use of level-sensitive dynamic latches, and generates less internal switching noise due to the uniformly distributed latch operation. By applying C^2 pipelining and its composition methods to build a system, VLSI designers can substitute the global clock skew problem with many local one-sided delay constraints.

I. Introduction to a high speed system

With the escalation of clock frequencies and the increasing ratio of wire- to gate-delays, clock skew is a major problem to be overcome in today's high-speed VLSI chips. Clock skew should ideally be less than 5-10% of the system clock cycle time [1]; this is a difficult figure to attain in many modern chips [2] and will become more so with the impending GHz rate of clocking [3]. The effect of shrinking VLSI feature sizes will increase this disparity [4] in the future, especially in the light of the fact that in submicron CMOS, interconnection delays are going to be larger than gate-propagation delays [5]. Consequently, an increased percentage of the clock period will be devoted to clock skew margins [6, 7]. The faster the clock and the bigger the die size, the worse the clock skew effects will be.

A major concern when building high performance VLSI systems is to build an effective clock distribution network. Many clock distribution methods for large high-speed VLSI chips have been developed [1] to achieve *rigid synchronization* (tight skew control) over the chip. Clock distribution networks of high-speed systems are normally comprised of binary trees of clock buffers [2, 8], which are expensive to produce in terms of area and design time. Network implementations such as H-tree methods [7] have been commonly exploited to reduce the clock skew. The effort to limit skews has an unfortunate side-effect: it causes the latches to switch almost simultaneously, causing ground-bounce and power-supply-droop, both of which can lead to chip malfunction. This often necessitates on-chip and off-chip decoupling capacitors [1], both of which add to the design cost.

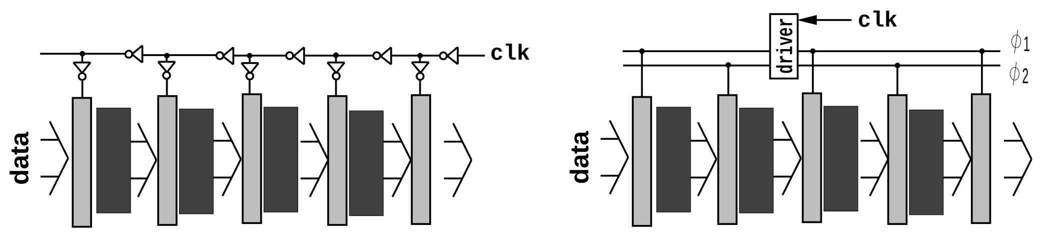
Rigidly clocked synchronous systems are often those that support a variety of data movements between their computational blocks. These systems have embedded bus structures that permit communication between physically distant modules. In these cases, the assumption that all the modules are rigidly synchronized to a global clock makes design easier, and hence is almost always made. *However, for systems that have a VLSI realization with mostly uni-directional data flows as well as possessing mostly nearest-neighbor connections, the assumption of rigidly synchronized clocking is not necessary, and can result in lost performance when enforced.* Examples of such chips are digital signal processing (DSP) chips, floating point units (FPU), graphics engines, asynchronous transfer mode (ATM) switches, etc. As we will show, higher performance and simpler clock distribution will result in these systems if we stick to *local* clocking constraints, much the same way the data dependencies in these systems are local. This is the main idea behind clock distribution in C^2 pipelined realizations.

Another major concern when building high performance VLSI systems is to employ high performance pipelined structures in conjunction with high speed clocks. Pipelining is a technique for reducing the clock period as well as increasing the amount of parallel circuit activity by splitting deep logic structures into shallower structures that are separated by pipelined latches. Although design methods for conventionally pipelined systems are well known [9], serious problems due to rigid clock synchronization may arise in very high speed pipelined designs. Strictly speaking, however, *pipelining* and *clocking* are orthogonal concepts. One can build *asynchronous* pipelines known as micropipelines [10] that do not employ clocks. However, the time penalty paid for generating the completion signals, as well as for handshaking [11] has prevented micropipelines from finding *widespread* use in high-performance VLSI systems. One can also implement *wavepipelining* [12] where the "latches" can be realized by the inherent combinational delays of logic structures. Despite their inherent performance advantages, wavepipelined systems require considerably more design effort to balance combinational delays, and consequently have received only limited usage. C^2 pipelining is a synchronous design scheme that (as pointed out before) comes with clock-distribution methods *as well as* pipeline design- and composition-methods.

A feature of C^2 pipelining is that the clock signals travel opposite to the direction of data movement. Back-propagating clock signals have been considered previously [2, 13], but never widely used in actual circuits. These previous back-propagating circuits were rigidly clocked, and hence offered no real advantages over H-tree distributed clocks; in fact, they actually increased the clock period. Another clocking method is buffered clocking, mentioned in El-Amawy [14], and originally described as *pipelined clocking* by Fisher *et al.* [7] (who does not assign any particular direction to pipelined clocks). This method also suffers from an increased clock period.

In C^2 pipelined systems, every pipeline stage employs clock buffers, as shown in Figure 1 (a), detailed explanation of which will be given in succeeding sections. These inverter buffers not only deliberately skew the clock (the exact one-sided constraints will be presented later) but also restore the clock-edge. This scheme achieves temporally distributed clocking. Clock amplification is also carried out in a distributed fashion. Conventional two-phase clocked pipelining is also illustrated in the figure for comparisons. The C^2 pipelining idea was first introduced in [15] where we presented many actual uses in the context of a subband vector quantizer (SB/VQ) chip. In this paper, we will focus on analyzing the timing constraints of C^2 pipelining. In Section V we will review the results of a C^2 pipelining network for the SB filtering chip.

Another feature of C^2 pipelined systems is that it enables one to use simple and efficient dynamic latches, which offer extremely low latch delays and areas, and avoids special latch designs [1, 2]. The C^2 pipelining method also staggers the switching activities of the latches, thus reducing the peak power consumption. This, in turn, reduces internal switching noise and also simplifies power-line routing, making it easier to distribute high speed clocks.



(a) Circuit C2 **(b) Circuit Conventional**

Figure 1: Circuit C2 and circuit Conventional

The pipeline interconnection methods to be described actually make the idea of C^2 pipelining more useful than pipelines with only nearest neighbor connections. In [15], we introduced such methods for 1) *data forwarding*, in which data skips a few pipeline stages in the direction of the dataflow, 2) *data backwarding*, in which data skips a few pipeline stages backwards (commonly used for iterative computations), 3) *sequential connection* of different pipelines, 4) *pipeline fork and join methods* to combine pipeline functionality in parallel, and 5) *synchronization methods* to synchronize incoming data and outgoing data to a clock signal. Timing constraints involved in these methods will also be discussed in detail in this paper in Section III.

In Section II, basic C^2 pipelining architectures are described and analyzed. Basic composition methods of data forwarding and data backwarding are analyzed in Section III. Section IV shows extended composition methods of sequential connection, pipeline fork and join and synchronization. These methods are explained using the analysis results shown in Section III. Section V gives a practical assessment of C^2 pipelining with a design and layout example. Conclusions are given in the final section.

II. Basic C^2 Pipelining Architectures

This section shows basic C^2 pipelining architectures with an analysis of timing constraints.

A. Principles of C^2 Pipelining Architecture

Figure 1 shows the difference between a C^2 pipelining and a conventional clock distribution for a pipeline. *Circuit C2* on Figure 1 (a) employs a chain of inverters to provide local clock signals. Local buffers attached to the chain provide appropriate output power to control local latches. Figure 1 (b) shows the conventional method in which a non-overlapping two-phase clock generator is located at the center of the clock distribution network. This clock generator is designed to cope with clock loads of the entire clock network.

C^2 pipelining can be realized in several ways as shown in Figure 2. Figure 2 (a) shows the basic architecture with back-propagating and inverting delays in a clock distribution line. Figure 2 (b) shows a version with computational components in the data path. Figure 2 (c) shows an alternative with noninverting buffers used in place of inverting buffers. Although they are illustrated differently, Figure 2 (b) can represent all three cases for the purpose of timing constraint analysis.

Figure 3 (a) shows a portion of the C^2 pipeline of Figure 2 (b), and Figure 3 (b) illustrates clock waveforms for its level sensitive dynamic latches. These latches are transparent during high clock signal and opaque during low clock signal. Latch i is controlled by a clock which is inverted and delayed by a clock line delay (d_c) from the clk_{i+1} for latch *i+1*. Similarly, latch *i-1* receives an inverted and delayed clock signal from clk_i.

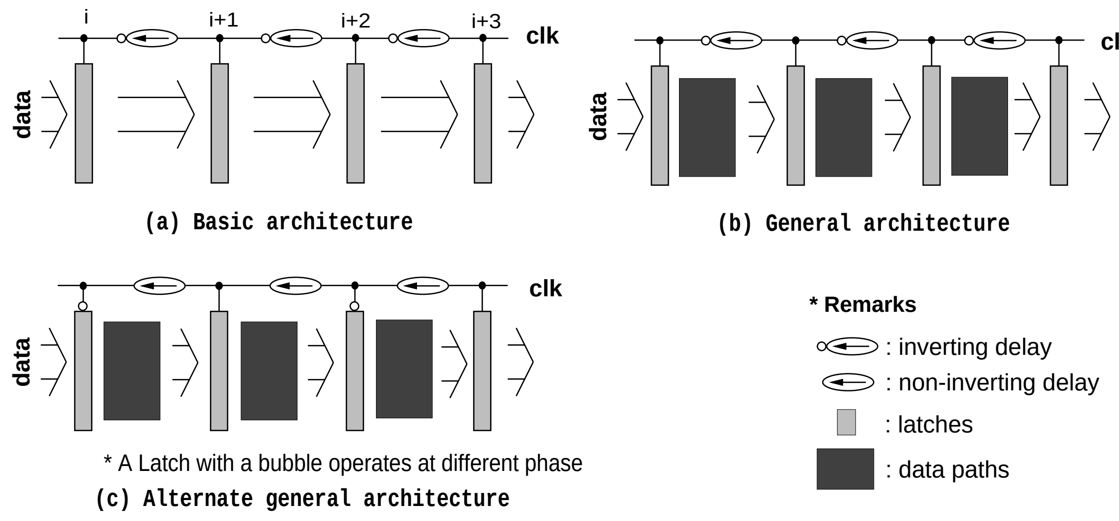


Figure 2: C^2 pipelining architectures

Clock timing analysis pertaining to a particular latch i with respect to its neighboring latches will now be discussed. First, the pipelining involves "go-throughs" during clock period I and III shown in Figure 3 (b) (due to the fact that C^2 pipelining implements overlapping clocks.) For instance, during period III, stage *i-1* output can "go-through" to stage *i+1* because the *i-1* latch is in hold while i and *i+1* are transparent. Go-through should be avoided in a *rigidly clocked synchronous* system with a *non-overlapping* clock. However, this go-through does not make stage *i+1* produce a wrong output in a C^2-pipelined system.

A possible scenario involving a go-through is the following:

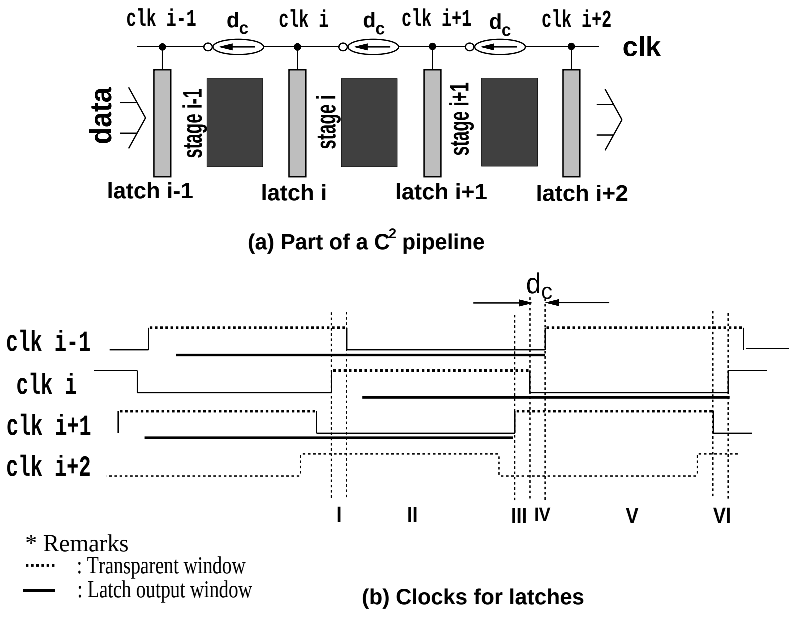


Figure 3: A part of a C^2 pipeline

- *stage-latch i-1* stabilizes its output by period II; however *stage i-1* delays this output which reaches the input of latch i only during period III (note the distinction between stage and stage-latch);
- the output of *stage i* (not stage-latch) can be generated early during period III and be sent to stage-latch *i+1* which is also transparent.

In this scenario, the output generated by latch *i-1* gets processed by stages *i-1* and i and is applied to the input of stage *i+1*—all during period III. This go-through is not harmful because it causes stage *i+1* output to tend towards the same value as it will evaluate to in the absence of go-through (much like *chaining* [16]). The go-through possible in period I can also be analyzed in the same way. In fact, go-throughs can actually help shorten the clock period by allowing a stage to absorb a fraction of the long-path delays associated with the stage preceding it. This can potentially be an advantage if the stage delays are not exactly balanced. The other periods involved (II, IV, V and VI) do not allow go-throughs to happen.

Figure 4 illustrates the overall latch operations for a C^2 pipeline. This figure shows staggered latch operations, where each latch alternates between transparent and opaque states. The vertical bold lines emanating from one period of the *latch i* operation marks a *sending window*, involving a transparent state and the succeeding opaque state of a latch, and a matching *receiving window* of the following latch. The latter *latch i+1* is in the transparent state between the two bold lines. This shows that the latches are operating as described in previous paragraphs.

The novelty of C^2 pipelining results from the use of intentionally inserted delays on clock lines. These delays not only provide pipeline speed-up described above, but also partition the clock line into many small pieces enabling one to avoid global clock skew problems. This leads to a locality property of timing constraint to the whole pipeline: *i. e.* the whole pipeline works properly by assuring local delay constraints for all stages.

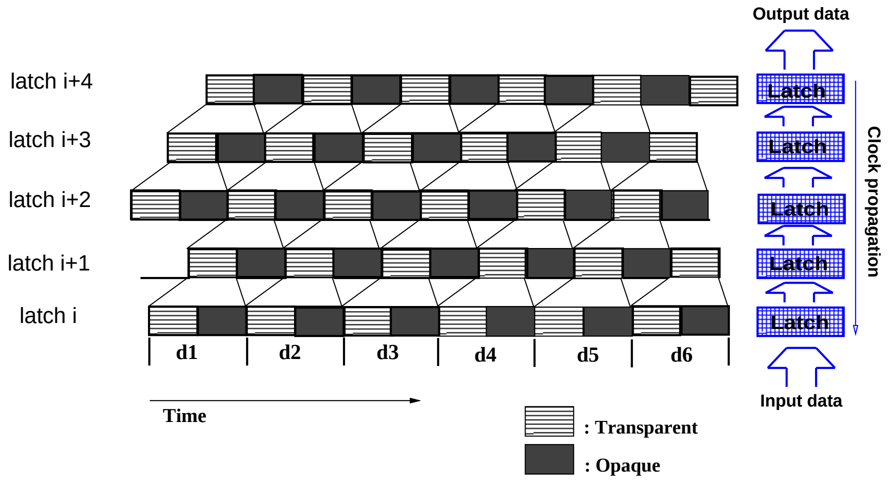


Figure 4: Latch operations for a C^2 pipeline

B. Timing Constraints for the Basic Architecture

In very high speed designs, the delays associated with segments of wires cannot be ignored. These are taken into consideration in the following calculations. Figure 5 shows stage i with its associated delays for clock wires, a clock buffer and a data path. Specifically, let the shortest wire delay for the first latch be d_{fsw}, the longest wire delay for the first latch be d_{flw}, the shortest wire delay for the second latch be d_{ssw}, the longest wire delay for the second latch be d_{slw}, the inserted inverting buffer delay for the clock be d_c, the shortest data path delay for the stage be d_{ds} and the longest data path delay for the stage be d_{dl}.

Figure 6 shows the detailed timing diagram for the stage shown in Figure 5, including latch set-up time (S) and hold time (H). This figure emphasizes (slowest and fastest) data validation timing for latch *i+1*, and (slowest and fastest) clocks for the latch *i+1*.

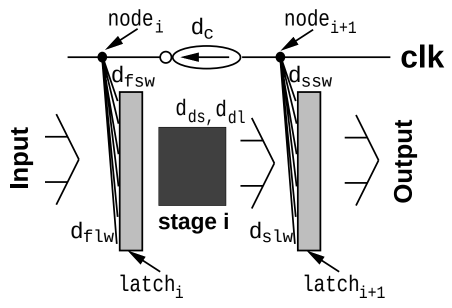


Figure 5: Delays associated with stage i

The local timing constraint can be derived as follows (Refer to Figure 6):

- The earliest available output time (t_a) of stage i is $d_c + d_{fsw} + d_{ds}$ after the falling edge of clock *clk* at node $i+1$.

- the latest activation time of latch $i+1$, t_b, is d_{slw} after the falling edge of the clock *clk* at node *i+1*.

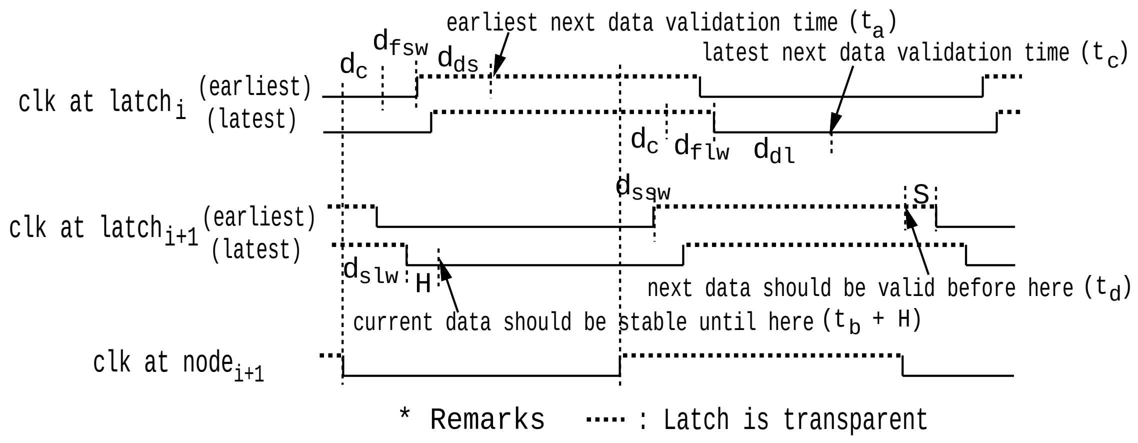

earliest next data validation time (t_a)
latest next data validation time (t_c)
d_c
d_{fsw}
d_{ds}
clk at latch$_i$
(earliest)
(latest)
d_c
d_{flw}
d_{dl}
d_{ssw}
S
clk at latch$_{i+1}$
(earliest)
(latest)
d_{slw}
H
next data should be valid before here (t_d)
current data should be stable until here (t_b + H)
clk at node$_{i+1}$
* Remarks ····· : Latch is transparent

Figure 6: Detail timing diagram for Figure 5

- t_a should be greater than t_b + H (H being the latch hold-time) to avoid violating the hold-time requirement for latch $i+1$ (avoid changing the data being latched by latch $i+1$ during the hold-time period).

Thus, the condition for the local timing constraint is

$$d_c + d_{fsw} + d_{ds} > d_{slw} + H$$

which results in the smallest inserted-delay value, d_c, of the clock line buffer(s) being:

$$d_c > d_{slw} + H - d_{fsw} - d_{ds} \qquad (1).$$

To calculate the minimum allowable clock-phase duration, P, assume a 50% duty-cycle clocks which results in:

- the latest data validation time at the input of latch *i+1* is when the incoming data to stage i was validated late. This time instant (t_c) will be $d_c + d_{flw} + d_{dl}$ after the rising edge of the clock *clk* as shown in Figure 6.
- the earliest latch *i+1* opening time (latch opened by the rise of Clk) is d_{ssw} after the rising edge. Therefore, the earliest latch *i+1* closing time is $d_{ssw} + P$.
- t_c should be before $t_d = d_{ssw} + P - S$ to satisfy the latch *i+1* setup time.

This will result in the following inequality:

$$d_c + d_{flw} + d_{dl} < P - S + d_{ssw}$$

which results in the clock phase duration lower-bound

$$P > d_c + d_{flw} + d_{dl} + S - d_{ssw} \qquad (2).$$

The inequality in (2) can always be satisfied because the clock period is externally controllable as in conventional synchronous clocking. *The inequality in (1) is the condition that is most important.*

Hence, C^2 pipelining results in *one-sided timing constraints.* Also, notice that (2) is *independent of clock-skew*, which confirms the observation that C^2 pipelining is an attractive method for GHz clocked circuits where skew is expected to become a major problem using conventional rigid-clocking methods.

III. Basic composition methods

During the composition of C^2 pipeline blocks, there will arise situations in which the data needs to (1) move downstream (with respect to the data movement) to be consumed by a functional block with typically several inputs (Figure 7 (a)), and/or (2) move upstream to be consumed by a functional block with several inputs (typically in iteration structures) (Figure 7 (b)). As we expect such "stage skipping" connections to be infrequent as well as skip only a small number of stages, we do not provide any special circuits to resynchronize the data; instead, we obtain timing constraints to be obeyed. Skips over longer distances have to proceed as several short skips in sequence with corresponding adjustments in the data timing. Timing constraints required for data forwarding and backwarding are now analyzed in the sections to follow.

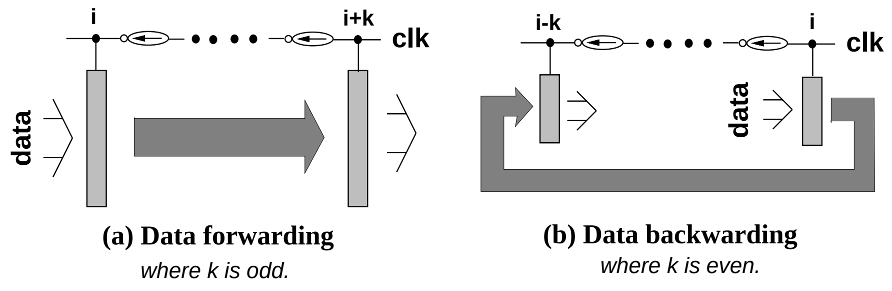


Figure 7: Data forwarding and data backwarding

A. Data Forwarding

Figure 8 (a) shows data forwarding ignoring wire delays. The simplest form of data forwarding is feeding data from latch i to latch $i+1$ with blank data path (waveforms (1) and (2) of Figure 8 (a) respectively). Notice that in this case latch $i+1$ is controlled by a clock that is inverted and leading with respect to the latch i clock. This case of forwarding represents an empty C^2 pipeline stage with zero data-path delays and was analyzed in the previous section.

When the destination is a latch with clk_{i+2j+1} as shown in waveform (3), the clock for this latch is inverted and leading by $(2j+1)*d_c$ where d_c is a clock buffer delay. This leading duration can be extended up to $P-S$ as shown by waveform (4) (*i.e.*, the latest data sent by latch i must fall before the set-up time window, marked S, of a latch with clk_{i+2k+1} on waveform (4)). Note that data forwarding by whole cycles is possible. However, such extended forwarding needs to be avoided since the amount of delay in a long chain of inverters can significantly vary with temperature, operating voltage and fabrication process parameters, and hence may not reliably track the cycle time.

Waveforms (1) and (5) give an example of an incorrect data go-through situation occurring from latch i to latch $i+2l+1$. This resulted from a violation of the above-stated forwarding limit. In this example, imagine that latch i presents incorrect data at the beginning of its transparent state and correct data only at the end of its transparent state. By the time correct data is presented, however, latch $i+2l+1$ could become opaque, as can be seen in waveform (5).

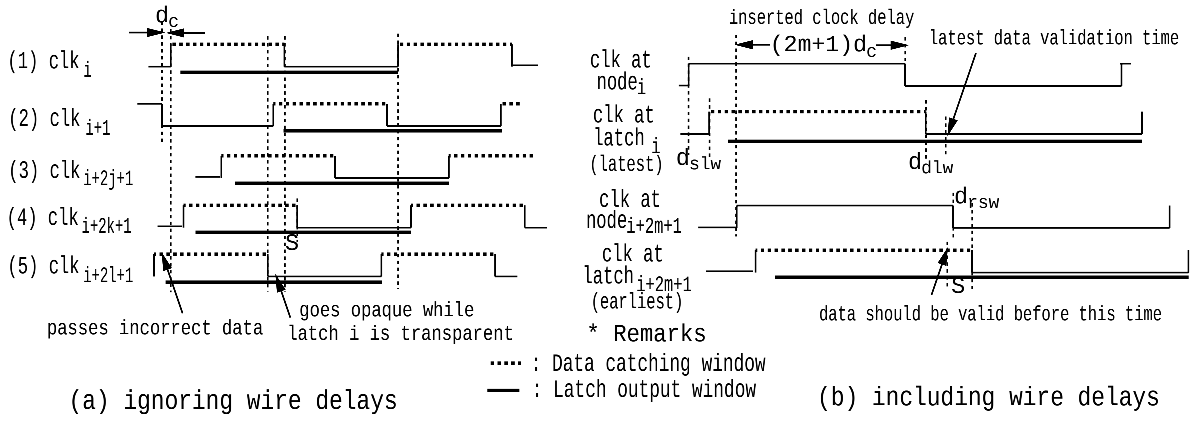


Figure 8: Timing diagram for data forwarding

The timing constraint for data forwarding, ignoring wiring delays, is :

$(2m + 1) * d_c < P - S$ where m is a positive integer.

The timing constraint with wiring delays taken into account can be derived as follows (see Figure 8 (b) also). (Note: this derivation may be skipped during initial reading.) Specifically, let the shortest wire delay for the sending latch be d_{ssw}, the longest wire delay for the sending latch be d_{slw}, the shortest wire delay for data sending be d_{dsw}, the longest wire delay for data sending be d_{dlw}, the shortest wire delay for the receiving latch be d_{rsw} and the longest wire delay for the receiving latch be d_{rlw}. When our signal-and-data observation point is at latch *i+2m+1*, the worst case scenario is: 1) the latest data validation time to the input of the latch is $d_{slw} + d_{dlw}$ after falling edge of the *clk* at node *i*, and 2) the earliest clocking time for the latch is d_{rsw} after falling edge of the *clk* at node *i+2m+1*. Timing of 1) should come before timing of 2) to meet latch *i+2m+1* set-up time condition. This results in:

$(2m + 1) * d_c - P + d_{slw} + P - d_{dlw} < P + d_{rsw} - S.$

Thus, the maximum forwarding limit is:

$(2m + 1) * d_c < P - S - (d_{slw} + d_{dlw}) + d_{rsw}$ (3).

For example, taking $d_c = 1ns$ for an inverter delay and $P = 50ns$ for phase duration for a chip in 2-μ CMOS, the maximum number of stage skips can be about 50 since other delay values of S, d_{slw}, d_{dlw} and d_{rsw} are negligible. Comparing this limit with the previous case (with wire delays ignored), we see that data forwarding limit is decreased by $(d_{slw} + d_{dlw}) - d_{rsw}$, the value of which is mostly positive. If long forwarding is needed, both of $(2m + 1) * d_c$ value on clock line and d_{dlw} value on connection wire between the two latches will restrict the number of stages to be forwarded.

Figure 9 shows overall latch operations for a data forwarding to send data directly from *latch i* to *latch i+3*. Vertical bold lines on *latch i* operation shows a sending window and a transparent state between the bold lines on *latch i+3* operation shows a receiving window. If data from *latch i* should be directly sent to *latch i+n* in the figure, it results in hazardous latch operation as discussed above.

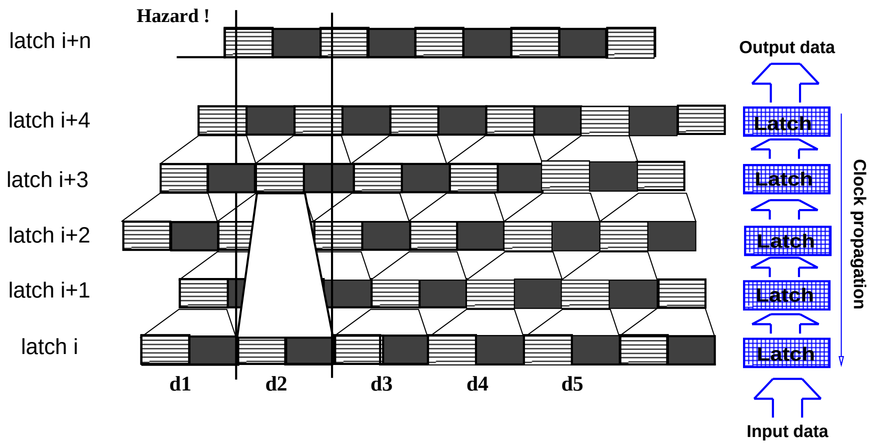


Figure 9: Latch operations for data forwarding

Figure 10 shows an example circuit of a line memory unit design in C^2 pipelining to provide delayed data [15] which is often necessary during image data processing. This design has the advantage of staggering transistor switching activity among each line memory block due to the deliberately skewed nature of C^2-pipelined clock. The peak power consumption can be much lower which increases noise margin and lower power rail capacity required. *Data forwarding 1* is directly achieved since the difference between data bundle clock C_i and destination clock C_{i+5} is five which is *odd*. However, *data forwarding 2* with four clock difference, which is not odd, is achieved in two steps dividing the four into one and three: data forwarding from clock C_{i+1} to C_{i+2} and data forwarding from clock C_{i+2} to C_{i+5} with the cost of extra latches for clock C_{i+2}.

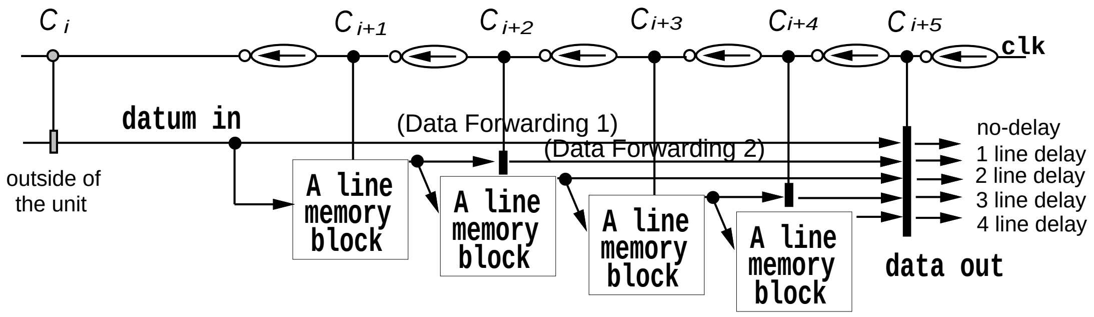


Figure 10: Line memory unit design in C^2 pipelining

B. Data Backwarding

Figure 11 (a) shows data backwarding ignoring wire delays. The simplest form of data backwarding is feeding data from a latch with clk_i to a latch with clk_{i-2} with blank data path (waveform (1) and (2) of Figure 11 (a) respectively). Waveform (2) on the figure shows that latch *i-2* is controlled by a non-inverting and delayed clock. The clock delay on the clock line provides timing margin to send data from latch i to latch *i-2* because the latch closing timing on waveform (2) is delayed by the clock delay amount from the latch closing timing on waveform (1). When the destination is a latch *i-2j* as can be seen in waveform (3), the clock for the latch is delayed by $2j * d_c$ and not inverted. The delay duration can be extended up to $P - H$ as shown by waveform (4) to meet latch *i-2k* hold time condition. Note that data backwarding by whole cycles is possible. However, such extended backwarding is to be avoided with the same reason for the case of extended data forwarding.

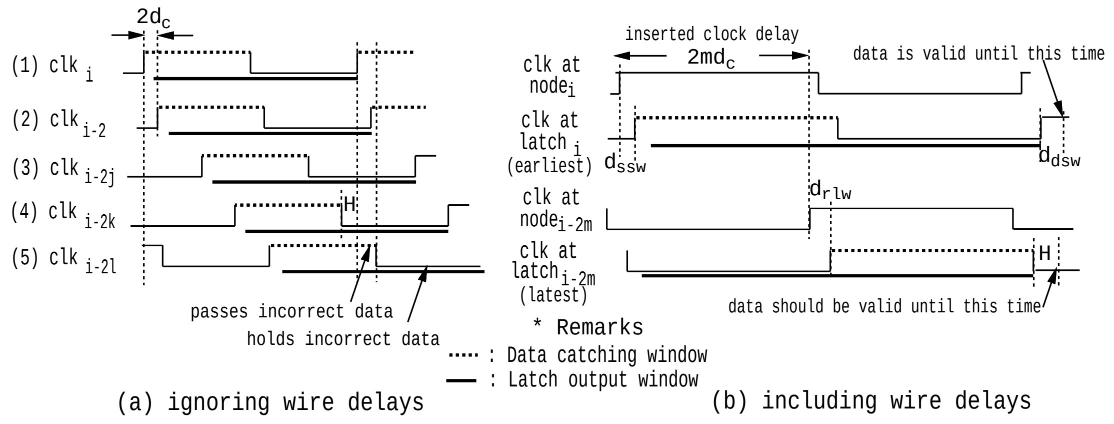


Figure 11: Timing diagram for data backwarding

Waveforms (1) and (5) give an example of an incorrect data go-through situation occurring from latch *i* to latch *i-2l*, resulting from a violation of the above-stated backwarding limit. In this example, imagine that latch *i* presents correct data, to be passed, during its opaque state and incorrect data at the beginning of its succeeding transparent state. By the time incorrect data is presented, latch *i-2l* could be transparent still as shown in waveform (5).

The timing constraint for data backwarding, ignoring wiring delays, is :

$$2m * d_c < P - H.$$

The timing constraint with wiring delays taken in to account can be derived as follows (see Figure 11 (b) also). (Note: this derivation may be skipped during initial reading.) The same wire delay conventions used for data forwarding are used. When our signal-and-data observation point is at latch *i-2m*, the worst case scenario is: 1) the earliest data validation time to the input of the latch is $d_{ssw} + d_{dsw}$ after the falling edge *clk* at node *i*, and 2) latest clocking time for the latch is d_{rlw} after the falling edge of the clock at node *i-2m*. Timing of 1) should come after the timing of 2) to keep latch *i-2m* hold time condition. This results in:

$$-2m * d_c + d_{ssw} + 2 * P + d_{dsw} > d_{rlw} + P + H.$$

Thus, the maximum backwarding limit is:

$$2m * d_c < P - H + (d_{ssw} + d_{dsw}) - d_{rlw} \qquad (4).$$

Comparing this limit with the previous case (with wire delays ignored), we see that the data backwarding limit is increased by $(d_{ssw} + d_{dsw}) - d_{rlw}$, the value of which is mostly positive. Both of $2m * d_c$ value on clock line and d_{rlw} value on connection wire between the two latches restrict the number of stages to be backwarded.

Figure 12 visualizes overall latch operations for a data backwarding to send data directly from *latch i* to *latch i-4*. Vertical bold lines on *latch i* operation shows a sending window and a transparent state between the bold lines on *latch i-4* operation shows a receiving window. If data from *latch i* should be directly sent to *latch i-n* on the figure, it results hazardous latch operation as discussed above.

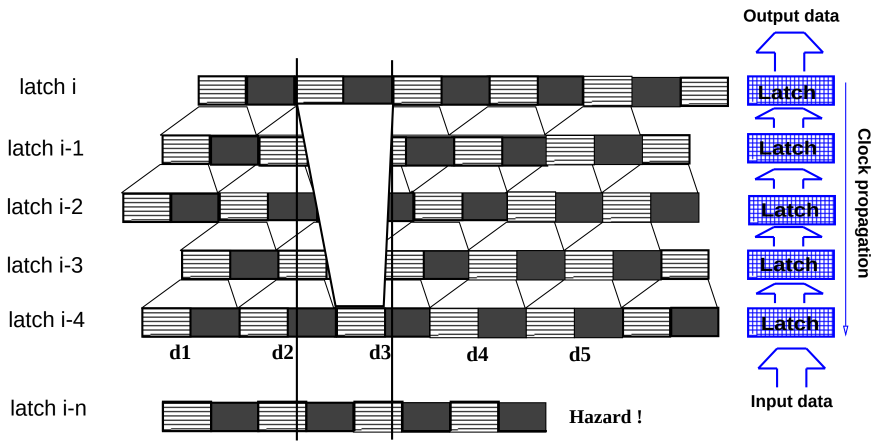


Figure 12: Latch operations for data backwarding

A concrete example of data backwarding is demonstrated in Figure 13 which shows a multiplication and accumulation unit which frequently used in digital signal processing. The data backwarding is need to iterative calculation since output data should be fed back to input. Figure 13 (a) shows data backwarding with direct implementation of data backwarding method with two clock difference which is *even*. This needs extra latches at *L2* position to latch the backwarding data. Dynamic latches of *L3* and *L4* provides necessary data register function. The cost here is the use of the extra latches to adjust latching timing. Figure 13 (b) shows a modified implementation to remove the extra latches. Data backwarding delay (d_b) was naturally implemented by a 2-MUX, to choose reset or accumulation, and an inverter to restore its output to positive logic value. This delay is enough to meet timing constraint when $d_b > 2d_c$, ignoring wiring delays for simplicity purpose of understanding. This means that backwarding data wave to the adders will arrive later than the data wave from *L2*. This modified version is free of extra latches.

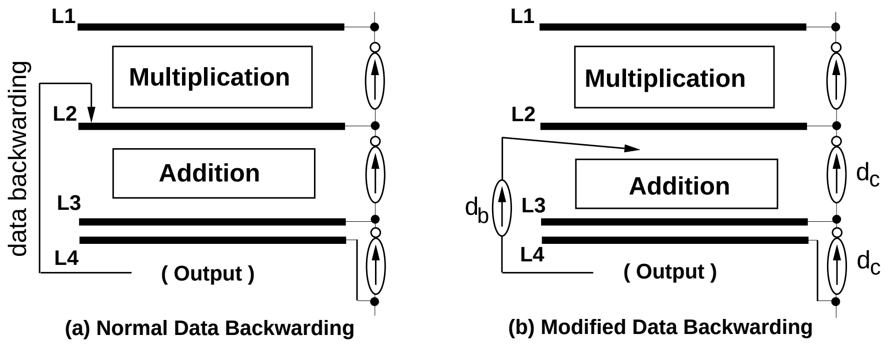


Figure 13: A Multiplication-and-accumulation unit example

IV. Extended Composition Methods

Although both data forwarding and data backwarding provide basic means to build a system, it is better to have extended composition methods such as a pipeline fork and join, sequential connections, and synchronization interface to build a C^2-pipelined system. Figure 14 (a) shows pipeline fork and join to connect two or more pipelines in parallel when the functionality of the pipelines are to be combined in parallel. Figure 14 (b) shows sequential pipeline connection to connect pipelines directly sequentially Figure 14 (c) shows pipeline synchronization, illustrating the incoming data and outgoing data of a C^2 pipelined block needed to be synchronized to a particular clock. This is needed when a block or a pipeline should be synchronized to a particular local or global clock signal. Use of this synchronization method provides a way to build a system with *hybrid clocking* with counterflow-clocking and conventional clocking. This can be valuable in high performance processors that have dedicated DSP hardware for example. These three extended composition methods, in addition to the two basic composition methods, provide a VLSI designer means to build systems of non-trivial size using C^2 pipelining. This section analyzes those extended methods.

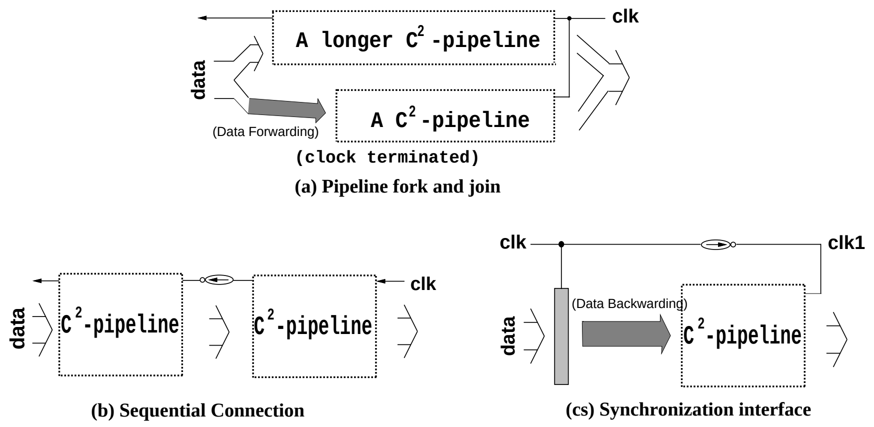


Figure 14: Connection methods

A. Pipeline Fork and Join

To connect pipelines in parallel, it is ideal to have same length (number of stages) of pipelines sharing a clock distribution line. When the lengths are not the same, clock timing adjustment should be made while sending or receiving data. Figure 14 (a) illustrates one of those two situations, showing the timing adjustment to feed data. Since the outgoing clock from a shorter pipeline is at the downstream of the data flow from the data to be fed, the data feeding should use the data forwarding method to adjust clock timing associated with the data. The outgoing clock needs to be terminated and another outgoing clock from the longer pipeline should be fed to preceding pipelines since the other one has correct timing for incoming data. Then, all the local timing constraints associated with input and output of the parallel-connected pipelines are satisfied.

B. Sequential Connection

The sequential connection method uses the same method as to connect two adjacent pipeline stages. The output from the preceding pipeline on Figure 14 (b) is fed to its succeeding pipeline in the figure using one inverting clock delay and wiring between latches to satisfy local wiring constraints.

C. Synchronization Interfaces

Since the synchronization method uses a particular clock signal for its input and its output, timing to the input or output of a C^2 pipeline should be adjusted. Figure 14 (c) illustrates such a method employing data backwarding for input data. Since the input data and the C^2 pipeline share the particular clock *clk*, the outgoing clock signal from the pipeline is delayed by the amount of its clock line delay. This necessitates data backwarding to adjust clock timing from the input data and the pipeline input. The output data of the pipeline is okay to be consumed by a receiver using the *clk* signal.

With this scheme, a physical layout issue pertaining to wiring arises. The fact that the output data normally should move from the end of the pipeline to its front necessitates having a long connection wire when both ends are physically separated far. In this case the timing constraint is satisfied with the scheme that the clock *clk1* is delayed and inverted from the clock *clk*. Then, timing constraints to the clock *clk* is satisfies for the input and the output. This interface provides a way to implement a functional block in C^2 pipelining *within a conventionally synchronous-clocked system.* Thus C^2 pipelining can also be used selectively in large VLSI chips.

V. A PRACTICAL ASSESSMENT OF C^2 PIPELINING

This section describes a specific C^2 pipelining design and with its associated layout issues. Figure 15 shows the design and layout of a subband filtering chip (SB chip) for processing HDTV input image data to four subband images [15, 17] The chip size was found to be $17.6^{mm} \times 15.8^{mm}$ in 2-μ CMOS technology with 483,000 transistors on it. The chip target clock speed, which is preliminary (using 2-μ CMOS technology), is 4.5 MHz. The speed is reduced by a factor of four from 18 MHz, which needed to process 72 MHz input pixel data, using 0.8-μ CMOS technology. The chip, based on Winzker *et al.*'s subband filtering chip set design [18], implemented (1) a 2D FIFO unit to provide two dimensional polyphase data for the filter stages [19], (2) two line memory units to provide line-delayed pixel data (each unit provides 5 sets of data: zero line delay to four line delay as shown previously in Figure 10), and (3) a filterbank unit to separate the incoming image into high frequency component and low frequency component vertically first and then horizontally by finite impulse response (FIR) filtering. The figure hides pads and testability circuits of the chip to highlight data flow and clock distribution.

The units in Figure 15 are connected sequentially: the 2D-FIFO unit to the incoming pixel data, two line memory units (unit I at left and unit II at right) in the middle in parallel, and the filterbank unit to produce output data of subband images, as shown in the figure. The *unidirectional data flow* is well suited to C^2 pipelining. The clocks counterflows the direction of the data flow as shown in the figure. Each unit was designed using C^2 pipelining except the 2D-FIFO unit which is designed by using conventionally rigid synchronization.

Regarding wiring lengths between the units, there are several things to mention. (1) The clock *c1* line delay (to be included on clock delay, d_c) and the longest delay on data bus *d1* provides timing margin for its data forwarding between the two units, Using the inequality derived in (3) of Section III. (2) Similarly, the data forwarding involved in clock *c3* line and data bus *d2* has more timing margin, due to longer wire-lengths, than that for previous *c1* and *d1*. (3) Between the two clocks *c2* and *c4*, *c4* has a longer delay than *c2*, due to longer physical wire for *c3* which is about half of the chip width; Thus *c4* needs to be fed to the preceding unit and *c2* is terminated,

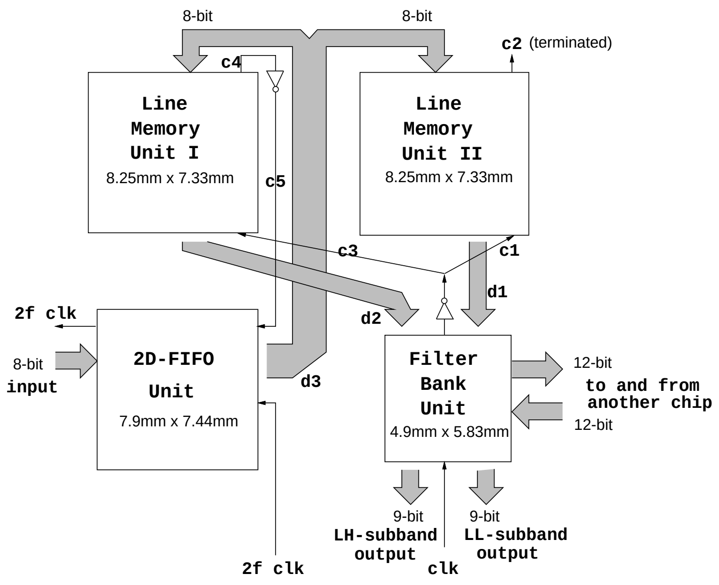


Figure 15: Subband filterbank chip layout

as previously described for pipeline fork and join connection method. (4) The data forwarding by the *d3* data bus and *c4* clock (to the line memory unit I) has timing margin realized through wire delays; The data forwarding by *d3* data bus and *c2* clock (to the line memory unit II) has more timing margin than that due to temporally advanced clock *c2*.

There are two other noteworthy things about the dsign: the double frequency clock (*2f clk*) feeding to a particular location of the chip and 12-bit connections to another chip. The *2f clk* needs timing adjustment to align the clock with the *c5* clock. The adjustment can be done by an adjustable delay element. The 12-bit connections involve data forwarding technique.

This section shows how a big chip for image processing can be designed and implemented in C^2 pipelining. Another big chip, a vector quantizer chip, which will process a subband image from the SB chip, has also been designed and implemented in C^2 pipelining [17].

VI. Conclusions

The development of C^2 pipelining was motivated from the fact that the development of an effective high speed clocking technique is essential for building high performance VLSI systems. It was observed that rigid synchronization over a chip or a system makes design easier. However, for systems that have VLSI realization with mostly uni-directional data flows as well as possessing mostly nearest-neighbor connections, the assumption of rigidly synchronized clocking is not necessary, and can result in lost performance (including waste of clock period due to clock skews and less noise margin due to simultaneous firing of latches) when enforced. C^2 pipelining was developed from an observation that many high speed systems show *mostly uni-directional data flows and mostly nearest-neighbor connections.* C^2 pipelining adopts back-propagating clock signals [2, 13], which are known to be safe but the use of which is usually avoided (due to extended clock period), in combination with pipelined clocking [7, 14]. A C^2-pipelined system can be built by using only "local delay constraints", which is a prominent feature to achieve very high speed clocking.

This paper introduced C^2 pipelining technology including 1) basic C^2 pipelining architectures, 2) the composition methods of data forwarding and data back warding, and 3) the extended composition methods of pipeline fork and join, sequential connection and synchronization interfaces.

Those basic architectures and composition methods provide a VLSI designer a means to build a system with many *one-sided local delay constraints* without concern about global clock distribution problems and skew control. A C^2 pipelining design and layout example for a subband filtering chip was reported in Section V. By building a system in C^2 pipelining, one can shorten clock period significantly when interconnection delays are larger than gate-propagation delays [5]. The trade-off is to use inverter chains and some extra latches for data forwarding or data backwarding (in case that receiving windows do not match with their corresponding sending windows) versus to build an elaborate clock distribution network to supply a global clock.

This paper concludes that by applying C^2 pipelining and its composition methods to build a system, clock periods can be much shorter than the one with rigidly clocked synchronization when interconnection delays are larger than gate propagation delays. In addition to this, power bump peaks can be reduced by staggered operation of latches. These two factors are essential for building large and very high speed clocked system.

References

[1] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley Publishing Company, Inc. 1990, pp. 353-355

[2] D. W. Dobberpuhl, *et al.*, "A 200-MHz 64-b dual-issue CMOS microprocessor," *IEEE J. of Solid-State Circuits*, Vol. 27, No. 11, November 1992

[3] S. Yinger, *et al.*, "HBT gate array for 5 GHz ASICs," *15th Annual GaAs IC Sym. Technical Digest*, October 1993

[4] Santanu Dutta and Wayne Wolf, "Asymptotic limits of video signal processing architectures," *ICCD 1994*, 1994

[5] D. Conner, "Submicron technologies require floorplanning," *EDA*, September 2, 1993

[6] J. Yuan, C. Svensson, "Pushing the limits of standard CMOS," *IEEE Spectrum*, February, 1991

[7] A. L. Fisher, H. T. Kung, "Synchronizing large VLSI processor arrays," *IEEE Trans. on Computers*, August 1985

[8] W. J. Bowhill *et al.*, "Circuit implementation of a 300-MHz 64-bit second-generation CMOS Alpha chip," *Digital Technical Journal*, Vol. 7, No.1, 1995, Digital Equipment Corporation.

[9] Karem A. Sakallah, *st. al*, "Synchronization of pipelines," *IEEE Trans. on Computer-Aided Design of ICs and Systems*, Vol. 12, No. 8, August 1993.

[10] I. E. Sutherland, "Micropipelines," *Comm. of the ACM*, June 1989, Vol. 32, No. 6

[11] A. J. Martin, "Tomorrow's digital hardware will be asynchronous and verified," California Institute of Technology, Tech. Report CS-TR-93-26, 1993

[12] D. Fan, *et al.*, "A CMOS parallel adder using wave pipelining," *Advanced Research in VLSI and Parallel Systems, Proc. of the 1992 Brown/MIT Conference*, 1992

[13] N. H. E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, 2nd Ed., Addison Wesley, 1993, pp. 239

[14] A. El-Amawy, "Clocking arbitrarily large computing structure under constant skew bound," *IEEE Trans. on Parallel and Distributed System*, March 1993

[15] Jae-tack Yoo *et al.*, "High speed counterflow-clocked pipelining illustrated on the design of HDTV subband vector quantizer chips," *Proc. of 16th Conference on Advanced Research in VLSI*, 1995

[16] B. M. Pangrle and D. D. Gajski, "Design tools for intelligent silicon compilation," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 6, November 1987

[17] Jae-tack Yoo, "Counterflow-Clocked Pipelining Illustrated on the Design of High Speed HDTV Subband Vector Quantizer Chips," *PhD Proposal*, Department of Computer Science, University of Utah, November, 1994.

[18] Marco Winzker, *et. al.*, "VLSI chip set for 2D HDTV subband filtering with on-chip line memories," *IEEE J. of Solid State Circuits*, Vol. 28, No. 12, December 1993

[19] P. P. Vaidyanathan, *Multirate systems and filter banks*, Prentice Hall, Englewood Cliffs, NJ. 1993.