Clocked Storage Elements in High-Performance and Low-Power Systems

Class notes for EEC280

"Despite the simple outward appearance of the clocking system, it is often a source of considerable trouble in actual systems" – Unger and Tan, 1986.

"A well planned clock system is a prerequisite to reliable long-term computer operation. Conversely, a badly designed clock system can plague a computer throught it's lifetime
- Kenet Wagner, IBM

Introduction

Clocking of a digital system is one of the single most important decisions. Unfortunately much too often it has been taken lightly at the beginning of a design and proven to be very costly afterwards. This very well summarized in the words of Kenet Wagner of IBM. Thus, it is not pretentious to dedicate an entire book to this subject. However, we are limiting this book to even narrower issue of clocked storage elements, widely known as Flip-Flops and Latches. The issues dealing with clock generation, frequency stability and control, and clock distribution are too numerous to be treated in depth in this book, thus they will be mentioned and covered only briefly. We hope there will be another book in this series dealing with those issues only as we will find it to be very useful and necessary for the complete coverage of the subject.

The importance of clocking is becoming even more emphasized as the clock speed has been rising rapidly doubling every three years.

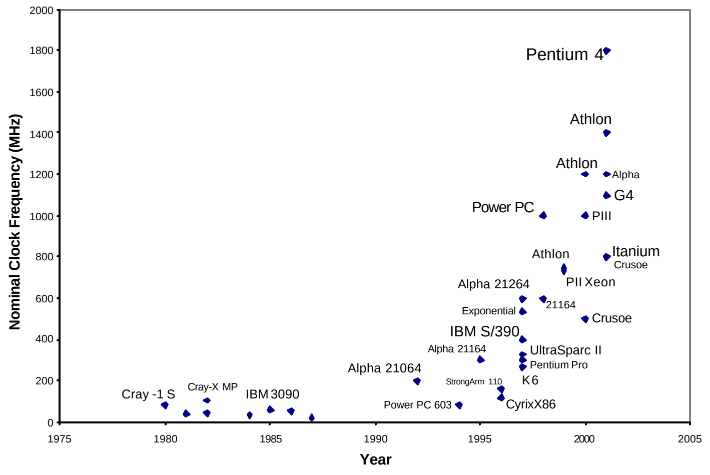


Fig. 1. Clock frequency versus year for various representative machines

At these frequencies ability to absorb tens of pico-seconds of clock skew or to make the clocked storage element faster for the same amount could result in 10% or more performance improvement since the performance is directly proportional to the clock frequency of a given system. Such performance improvements are very difficult to obtain through traditional techniques used on the architecture or micro-architecture level. Thus, setting the clock right and taking every available pico-second out of the critical path is becoming increasingly important. However, as the clock frequency reaches 5-10GHz range it our opinion that traditional clocking techniques will be reaching their limit. New, ideas and new ways of designing digital systems will be required. We do not pretend to know the answers, but some hints will be given throughout of this book to what we feel may be a good path to follow in the future designs.

System	**Intro Date**	**Technology**	**Class**	**Nominal Clock Period (nS)**	**Nominal Clock Freq. (MHz)**
Cray-X-MP	1982	MSI ECL	Vector Processor	9.5	105.3
Cray-1S,-1M	1980	MSI ECL	Vector Processor	12.5	80.0
CDC Cyber 180/990	1985	ECL	Mainframe	16.0	62.5
IBM 3090	1986	ECL	Mainframe	18.5	54.1
Amdahl 58	1982	LSI ECL	Mainframe	23.0	43.5
IBM 308X	1981	LSI TTL	Mainframe	24.5, 26.0	40.8,38.5
Univac 1100/90	1984	LSI ECL	Mainframe	30.0	33.3
MIPS-X	1987	VLSI CMOS	Microprocessor	50.0	20.0
HP-900	1982	VLSI CMOS	Micro-mainframe	55.6	18.0
Motorola 68020	1985	VLSI CMOS	Microprocessor	60.0	16.7
Bellmac-32A	1982	VLSI CMOS	Microprocessor	125.0	8.0

***from IEEE Design & Test of Computers**

Fig. 2. Clock frequency of some known historic computers and super-computers such as Cray and Cyber.

Historically computers were large in size filling up several electronic cabinets, which were laid out occupying an entire floor of a large air-conditioned room. They were built from discrete components using few of the LSI chips in the later models. Those systems were clocked at frequencies of about a MHz or few tens of MHz. Given the low scale of integration it was possible to "tune" the clock i.e. to either adjust the length of wires distributing clock signals or "tune" the various delay elements on the cabinets or the circuit boards so that the arrival of the clock signal to every circuit board can be adjusted to approximately the same point in time. With advent of VLSI technology the ability to "tune" the clock has virtually disappeared. The clock signals are generated and distributed internally in the VLSI chip. Therefore much burden has fallen onto the clocked timing element to absorb clock signal variations at various points of the VLSI chip, also known as the "clock skew".

Clocking in Synchronous Systems

The notion of clock and clocking is essential for the concept of synchronous design of digital systems. The synchronous system assumes a presence of the storage elements and combinational logic, which together comprise a Finite-State Machine (FSM). The changes in the FSM are in general result of two events: input signal changes and clock as illustrated in Fig.3.

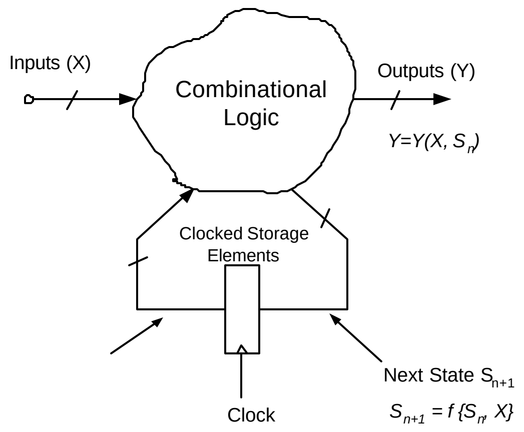


Fig. 3. The concept of Finite-State-Machine

The next state S_{n+1} is a function of the present state and the logic value of the input signals: $S_{n+1} = S_{n+1}(S_n, X_n)$. Therefore the remaining question is when in time will the next state of the storage S_{n+1}, elements be assumed ? This depends on the nature of the clocked storage elements used and the clock signal, which is introduced for that purpose. Therefore the presence of the clock signal introduces the reference point in time when the FSM changes from the present S_n to the next state S_{n+1}. This process is illustrated in Fig.4.

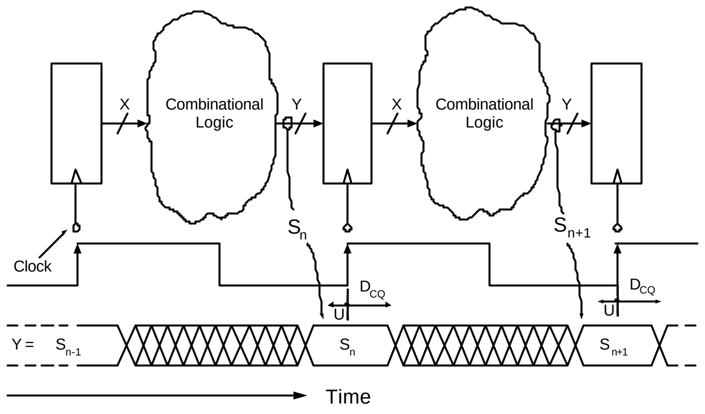


Fig. 4. State changes in the Finite-State-Machine

In Fig.4. we have implicitly assumed that the moment when the state changes from S_n to S_{n+1} is determined by the change of the clock signal from logic "0" to logic "1". However, this is in fact determined by the nature of the clocked storage elements used and will be discussed in details further in this book. For the purpose of this discussion let us just observe that without the presence of the clock signal this change from S_n to S_{n+1} will not be precisely determined. There are digital systems where this change is not caused by the presence, and more precisely, by the change of the clock signal but by the change of the data signal, for example. Such systems are known as "asynchronous systems" because they do not require the presence of the clock signal in order to cause an orderly transition from S_n to S_{n+1}. Much research has been done in the last several decades in defining a workable "asynchronous system". However, a practical design is yet to be produced. Recently one of the microprocessors was design to operate in the asynchronous manner and it has been claimed that some small advantages in power consumption were obtained [ARM processor]. In spite of that, the practicality as well as advantage of "asynchronous design" is yet to be proven.

Throughout of this book however, we will be staying with the discussion of the synchronous systems.

If we choose to unroll in time the state diagram of the FSM we can obtain the illustration of the pipelined design. In many cases when dealing with the synchronous design the delay thought the logic block is excessive and the signal change can not propagate to the inputs of the clocked storage elements in time to affect the change to the next state . In such a case, the machine has not met the "critical path requirement", i.e. it will fail in its functionality because the changes initiated by the input signals will have no effect. This is the case because the time allowed until the change to the next state S_{n+1} is to be achieved is too short and the change on the input signals simply did not have sufficient time to propagate. In technical jargon this is known as the "critical path violation". In such cases, an additional state (or states) is inserted to assure that every transition proceeds orderly and in time. A diagram of a pipelined system is shown in Fig. 5.

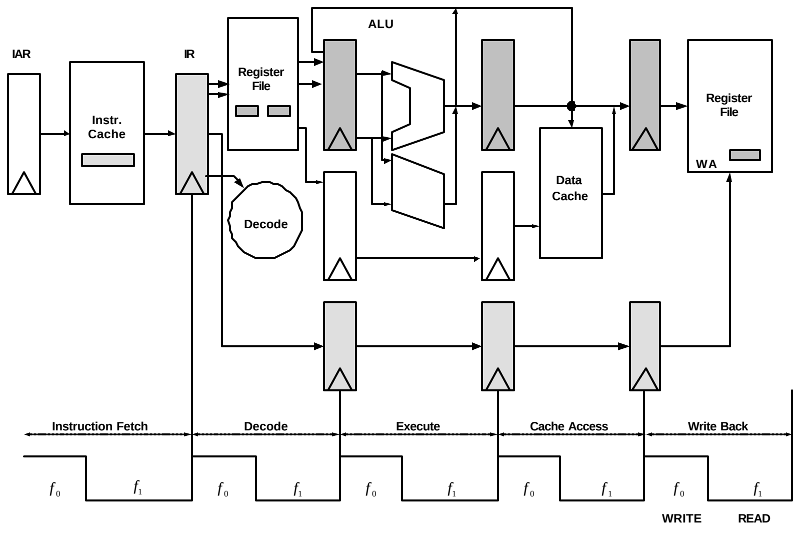


Fig. 5. Diagram of a pipelined system.

Several clock cycles may be needed in order to move through various stages of a computer system in time. In general, execution of an instruction may require several "*machine cycles*". This is especially true if *micro-code* is used to control the machine. In the past *micro-coding* was a popular concept and it was extensively used in Complex Instruction Set Machines (CISC). In those cases a process of executing an instruction required several *machine cycles*. During each machine cycle a micro-instruction was executed. It normally took several micro-instructions to execute an instruction. Each machine cycle required one or several register transfers or pass through several pipeline stages. That in turn required one or more clock cycles, or multiple phases of the clock. Thus, clocking was quite complex and encompassed several levels of hierarchy. This is illustrated in Fig. 6.

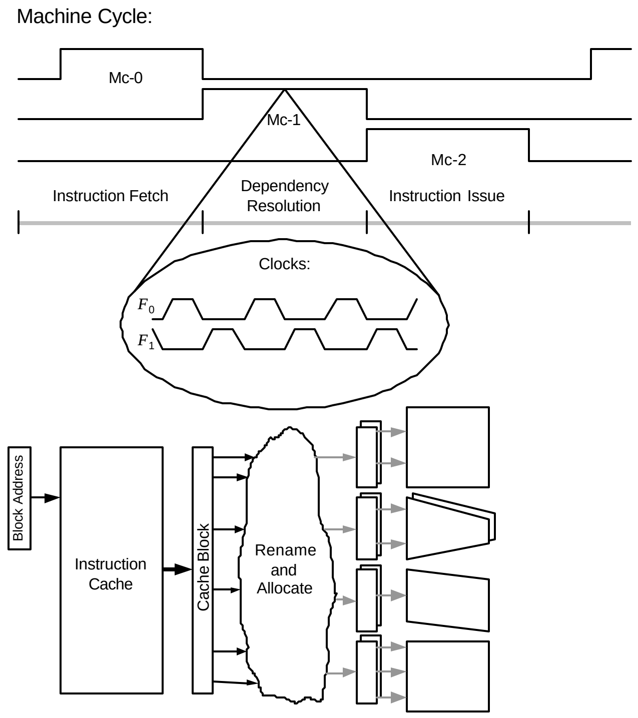


Fig. 6. Machine execution phases with respect to the clock cycles

Obviously, in micro-coded machines there existed a large disparity between the speed of the clock and the speed of logic. It could take several clock cycles or even several tens or hundreds of clock cycles in order to execute one instruction. The more complex instruction required many more clock cycles. A number of "*logic levels*" in the "*critical path*" (the number of gates in the longest path through the logic) was in order of several tens and 40-50 logic levels were not uncommon. Thus, the time associated with the clock and clocking was not as critical as it is today.

As the level of integration increased, followed by a speed increase of today machines, the number of logic levels in the *critical path* diminished rapidly. In today's high-speed processors that are either characterized by the Reduced Instruction Set Computer (RISC) architecture, or are using ROPs (RISC operations) in their micro-architecture, the concept of micro-coding has almost disappeared and so did the concept of machine cycle. The instructions are executed in one-cycle, which is driven by a single-phase clock. In other words one instruction is executed at every clock cycle. The levels of hierarchy that existed between the clock cycle and instruction execution have disappeared. In addition the pipeline depth keep decreasing in order to accommodate the trend in ever-rising speed. Today 10 levels of logic in the critical path are more common and this number is still decreasing as illustrated in Fig. 7. Thus any overhead associated with the clock system and clocking mechanism in directly and adversely affecting the machine performance and is therefore critically important.

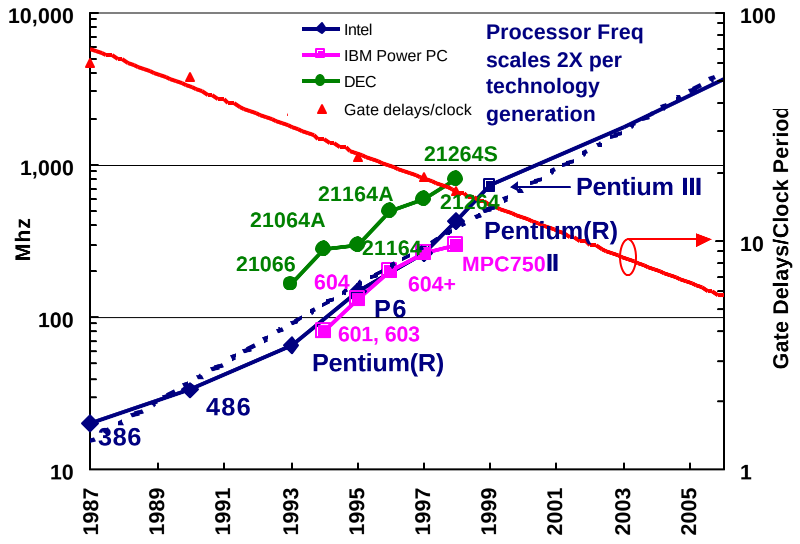


Fig. 7. Increase in the clock frequency and decrease in the number of logic levels in the pipeline (courtesy of Intel Corp.)

With this introduction we should be able to understand the function of the clock signal before we proceed with some of the definitions. The function of the clock signal can be compared to the function of the metronome in music. Similarly, in digital system the clock designates the exact moment when the state is changing as well as when the next state is to be captured. Also, all the logic operations have to finish before the tick of the clock because their final values are being captured at the tick of the clock. Therefore, the clock provides the time reference point, which determines the movement of data in the digital system.

Clock generation and synchronization

In this section, a short overview will be presented on how the system clock is generated, brought on-chip and synchronized with the on-chip clock. In addition, two typical clock aligner topologies are discussed, and main noise sources outlined leading to some well-known design tradeoffs.

System clock

Clock generation begins on a system board, where the global system clock reference is generated from a "crystal" oscillator. This is a circuit that uses a piezoelectric quartz crystal or some ceramic materials, as a mechanical representation of an electrical LRC series resonant circuit. Piezoelectric effect in a material occurs with the exchange in energy between the mechanical compression and applied electric field. In quartz crystal, the physical dimensions of the lattice can very precisely determine the oscillation frequency. Very good property of such resonators is extremely high Q-factor, typically 1000-10000. By attaching a non-linear element (such as NFET) to the resonator, the series resistance of the resonator is cancelled by the negative resistance of the non-linear element and "loss-less" oscillations are maintained. Due to the high quality Q factor, the variation of the resonant frequency of the oscillator is only a few parts-per-million (ppm).

System clock is set to directly correspond with the speed of data busses on the system board, i.e. from 66MHz, 100MHz, 133MHz in PC boards, to a few hundred MHz in specialized systems. However, the on-chip clocks operate at frequencies that are in GHz range. Even if the on-board clock signal of the same frequency as on-chip clock, could be generated, it would be very hard to bring it on-chip, because of large parasitic capacitances and inductances in the package and bondwires/balls that connect to the die. From these reasons, the low frequency system clock is first brought on-chip and then frequency multiplication is performed to achieve desired on-chip clock rate.

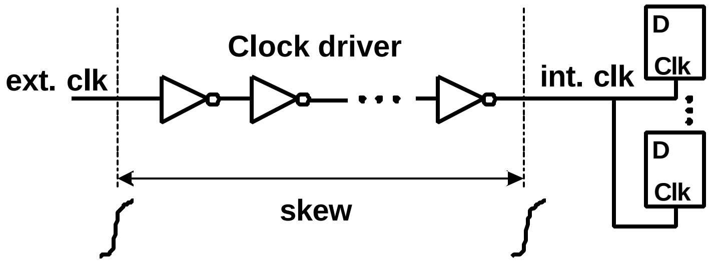


Fig. 8. On-chip clock skew

With the increase in on-chip clock frequency, it became necessary to eliminate the delay between external and internal clock (clock skew) caused by the on-chip clock driver delay, as shown in Fig. 1 with inverter chain representing the equivalent of the clock driver tree, and flip-flops/latches, the total clock load. Several nF of clock load are routinely encountered in modern microprocessor designs, [1]. This requires 5 or more FO4 delays through the clock driver, easily attributing to over 50% of the processor cycle time and causing large setup/hold times for the input/output signals. Moreover, due to process and environmental variations, the delay of the clock driver may vary, causing unknown phase relationship of the external and internal clock. This problem can be solved using the phase-locked-loop (PLL). The main task of the PLL is to align the external reference clock with the on-chip internal clock at the end of the clock driver, thus effectively removing the driver delay (skew).

On-chip clock generators/aligners

There are two main types of PLLs. In first type, the PLL has its own voltage controlled oscillator (VCO) that generates the internal clock which is then aligned to the external reference clock by the virtue of negative feedback, as shown in Fig. 2. The phase difference between the external reference clock and the internal distributed clock is detected with the phase detector, PD, and low-pass filtered, LP, to create the control voltage for the VCO, steering the oscillation frequency in such direction as to align the external and internal clocks achieving ideally a zero phase difference, at which a so called lock is achieved, [2]. This type of PLL was introduced first and hence, historically kept the name PLL.

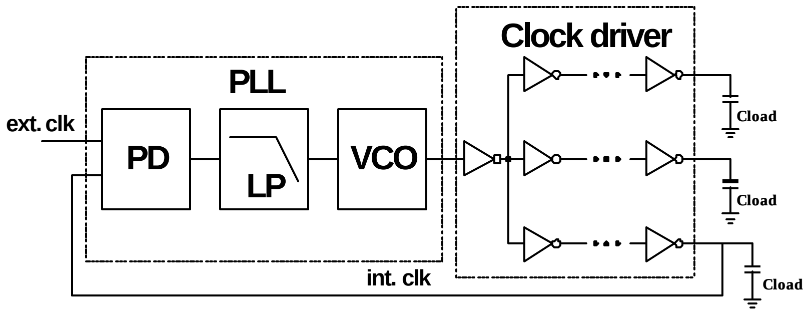


Fig. 9. The phase-locked loop block diagram and operation

The other type of the PLL is delay-line based or delay-locked loop (DLL). As shown in Fig. 3, the VCO in the PLL is replaced by the voltage controlled delay-line (VCDL) which delays the external clock, feeding the clock driver, until the internal clock becomes aligned with the external clock, at which point the control voltage of the VCDL will become unchanged and the loop will stay in lock. The key point to realize is that in both PLL and DLL, the alignment is possible because both external and internal clocks are periodic, hence delaying them by an integer number of cycles with respect to each other results in delay cancellation. Otherwise, it would not be physically possible to subtract the delay (skew). It is only possible to add some more delay until the total delay becomes integer number of clock cycles.

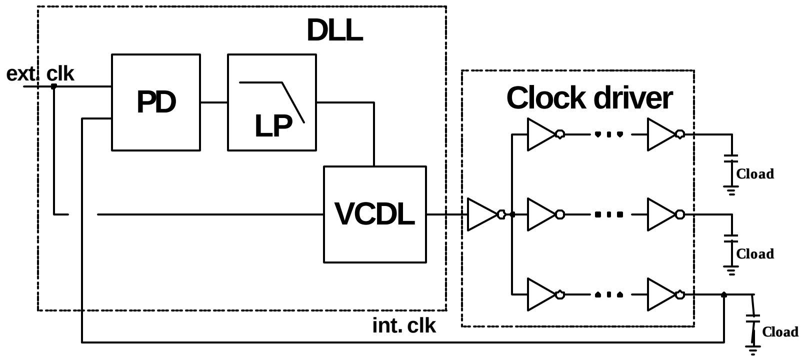


Fig. 10. The delay-locked loop block diagram and operation

In addition to clock alignment, PLLs can perform frequency multiplication, which is very useful in microprocessor systems, as explained above. Fig. 3 shows general block diagram where the VCO operates at f_{vco}=fext×B×C/A, and the frequency of the internal clock is f_{int}= f_{vco}/B. Typically, the value of B is two, to guarantee 50% duty cycle of the internal clock, the value of A is one, while the value of C is set to the ratio between the desired internal clock frequency and the external (system) clock frequency [1], always conveniently set to be an integer value, preferably base two. There are, however, cases where multiple values of A, B and C are used in the power-up sequence to avoid excessive supply noise on large chips, like Alpha 21264, [3].

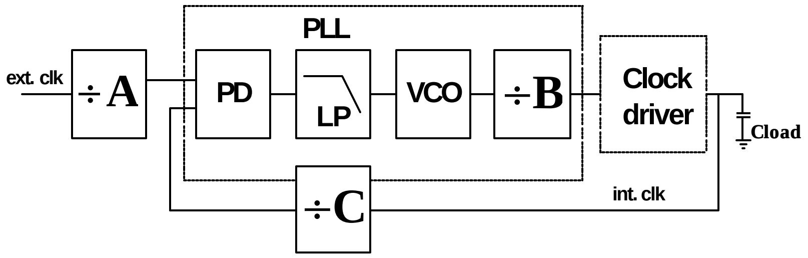


Fig. 11. PLL frequency multiplication

VCO is built either as a ring oscillator topology or LC tank oscillator, with later becoming possible with the use of on-chip spiral inductors. VCDL can be built of the same delay elements as the ring oscillator VCO. Most often used delay elements are differential pairs which provide good power supply rejection, and recently popular, inverters, with power supply regulator that performs power supply filtering and effectively shields the inverters of any power supply noise, [3], [6]. For details on other building blocks of the PLL and DLL we refer the reader to [2], [4], [5] for further exploration. The following section briefly describes some of the most important noise sources and tradeoffs involved in PLL and DLL design as well as comparative analysis of PLL vs. DLL performance.

Main noise sources and optimal loop bandwidth

For the purposes of high-level analysis, we divide the noise sources into three main categories: 1) noise of the reference clock, 2) noise induced in the VCO (VCDL) and 3) noise induced on the clock during distribution from the PLL (DLL) to the latch/flip-flop, here defined as clock driver noise. Since these noise sources are introduced into the loop at different locations, the transfer functions to the output are different for each of them. For example, input reference noise is low pass filtered at the output of the PLL, with filter bandwidth set by the bandwidth of the PLL. On the other hand, input reference noise passes directly to the output of the DLL, through the VCDL, without any filtering. Noise induced in the VCO is fed-back to the VCO input (in ring-oscillator implementation) and "accumulated", [4]. Any noise induced in VCO or VCDL is tracked and rejected by the loop, up to the loop bandwidth. Therefore, the transfer function of noise from VCO (VCDL) to the output is high-pass, contrary to the one from the input reference to the output. This immediately points to the possible tradeoff between the amount of input reference noise and VCO noise, at the output of the PLL. Indeed, optimal bandwidth exists at which these two noise sources are balanced and minimum total noise is achieved, [7]. In summary, DLLs have better performance in cases where reference clock is clean and most dominant noise occurs from the noise induced in the VCDL line. PLLs are, however, better in cases where the input reference noise is dominant, and typically worse in cases of dominant noise induced in the VCO, due to the noise "accumulation" effect, given that compared VCOs and VCDLs are implemented using the same type of delay element.

The analysis above is somewhat blurred in modern systems, due to the noise induced in the clock driver. While VCOs and VCDLs are typically implemented using 3-6 delay stages, due to the increasing amount of clock load, clock driver depth increases from generation to generation, and is over 5 stages in modern processors. Given that sensitivity of the delay elements in VCO or VCDL is typically order of magnitude better than that of the inverter, which has 1% delay variation for 1% power supply variation, it can be easily seen that the overall noise of the distributed on-chip clock is usually dominated by the noise induced in clock driver tree.

Regarding the design of the PLLs and DLLs, PLLs are typically harder to design, due to the stability issues (PLL is a second order system due to the integrating function of the VCO), but offer more flexibility than DLLs, i.e. wider locking range, frequency multiplication, etc. DLLs are simpler to design, given that they are first-order systems (unconditionally stable), but offer limited lock range. However, it is true that more complicated DLLs that offer similar flexibility to PLLs are also very complex systems, [8].

PLLs are mostly used in modern processors to multiply the frequency of the external system clock and reject any existing high frequency reference clock noise. DLLs have recently found application as de-skewing elements in high-performance processors, synchronizing different clock domains on a die to the global clock reference from the PLL, [9], [10]. It should be noted, however, that these approaches only deal with the DC portion of the noise on the clock (skew), while AC portion of the noise (jitter) is not eliminated as discussed above. The jitter induced in the clock driver by power supply variations still presents dominant source of noise in the on-chip clock distribution and needs to be budgeted for in any clocking methodology.

Clock System Design

Clock system is usually divided into two distinct categories: *Clock Generation* and *Clock Distribution*. However, this classification should be extended by adding *Clocked Storage Elements* as an additional category because the nature of clocked storage elements is intimately connected to the clock system generation and distribution and it is the nature of clocked storage elements that dictates requirements imposed on the clock system. This relationship is the best illustrated by the choice of the clocking scheme as show in Fig. 8. The clock system could consist of a single-phase clock, two-phase, or multiple phase clocks.

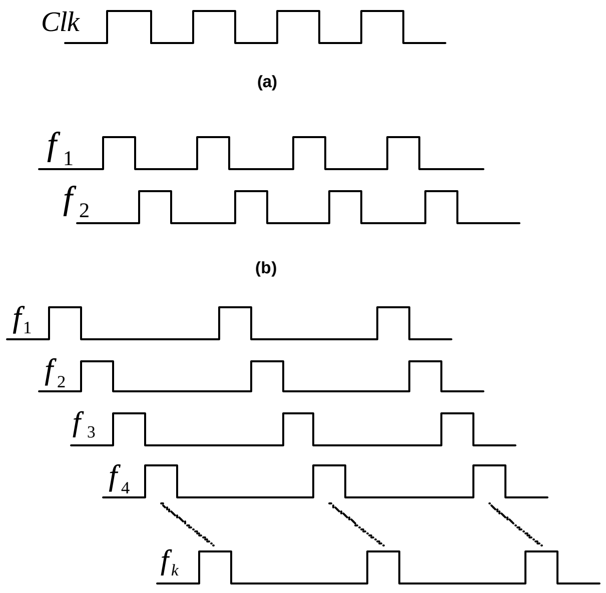


Fig. 12. System Clocking Schemes: (a.) single-phase, (b.) two -phase, (c.) multiple-phase clock

In the older system it was more common to see multiple-phase clocks. As the frequency of operation keep increases it became increasingly difficult to control various phases of the clock and their relationship to each other.

The two-phase clock is a robust scheme and is compatible with the design for testability, a desired feature of a complex computer system. Such a scheme, which incorporates a test mode, has been used in generations of IBM mainframe computers as a part of Level Sensitive Scan Design (LSSD) design methodology. The two non-overlapping phases of the clock assure a robust clocking system, tolerant to the manufacturing and process parameter changes.

However, as the quest for more speed continued, combined with the increased level of integration even the relation between two phases of the clock became difficult to control on the chip. That lead to the wide spread adoption of a single-phase clock today. The two-phase docking is still used in some of the systems, however, it is a single-phase clock that is distributed thought the system and the two necessary phases are generated locally. This technique achieves two goals: (a.) necessary amplification of the clock signals and ability to drive a large row of storage elements (register for example), (b.) generation of two clock phases and compatibility with scan test methodology. A scheme used for local two phase-clock generation from a single-phase clock distributed on the chip is shown in Fig. 9. Such a scheme is also capable of supporting the TEST (SCANN) mode.

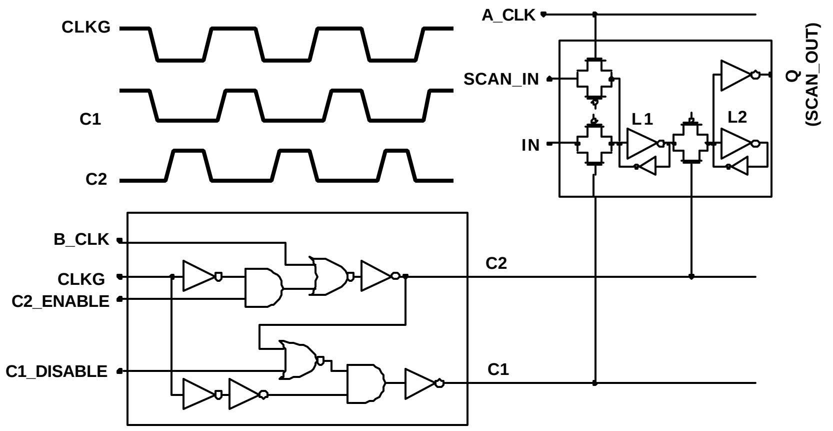


Fig. 13. Local generation of two -phase clocks as used in IBM PowerPC.

Clock Generation and Clock Distribution

Usually clock signal is generated using quartz crystal controlled oscillator to provide accurate and stable frequency. Given the size limitation of the quartz crystal, the frequency of such generated clock signal cannot be very high and frequencies in excess of 30-50 MHz are rarely generated using quartz crystal. The clock signal is conditioned and amplified to reach desirable driving strength before it is being applied to the outside pins of a VLSI chip from which it is driving an internal PLL or DLL. Before reaching the boundaries of the VLSI chip adjustments to its shape and form are possible. In older computer systems that consisted of several electronic cabinets distributed over the “computer floor”, and containing number of printed circuit boards, adjustment to the clock signal were made at each level. Thus, the clock signals were distributed over longer distances and over several levels including the cabinet, printed circuit boards and modules internal to them. Those separate entities entered by the clock signal were referred as “logic islands”, the term introduced by Amdahl. The concept of “logic islands” is illustrated in Fig. 10.

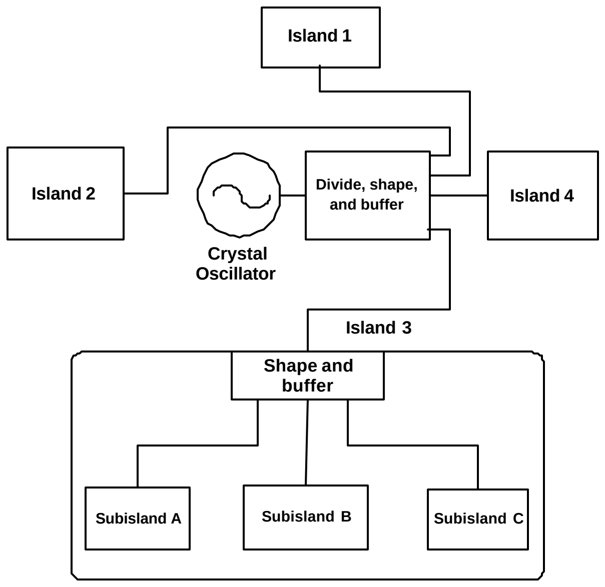


Fig. 14. The concept of Logic Islands [D&T of Computer; Wagner]

At the point of the clock entry to the board or cabinet (referred to as an "island"), further tuning and delay adjusting of the clock signal is possible, as shown in Fig. 11. Those elements are usually referred to as "tuning points". The positioning of "tuning points" in a system is illustrated in Fig.11. Various clock shaping, forming and tunable delay elements are employed, some of them are illustrated in Fig. 12. They make it possible to control the timing of the "leading" as well as "trailing edge" of the clock signal and to produce and "early" as well as "late" clock signal with reference to the nominal clock.

By adjusting the clock delay and subsequently shaping the edges of the clock signal it is possible to create "early", "nominal" and "late" clocks as shown in Fig. 13 c. Those clocks can be routed to various points on the board accordingly. It is obvious that older systems had much greater control of the clock signal than what is possible today because once the clock reaches the boundary of the LSI chip, tuning and shaping of the clock is not possible. This is because it is much more difficult to perform tuning on the chip due to the lack of external control and greater parameter variations on the chip. It is also difficult to build tuning elements such as inductors on the chip and to make adjustments from outside.

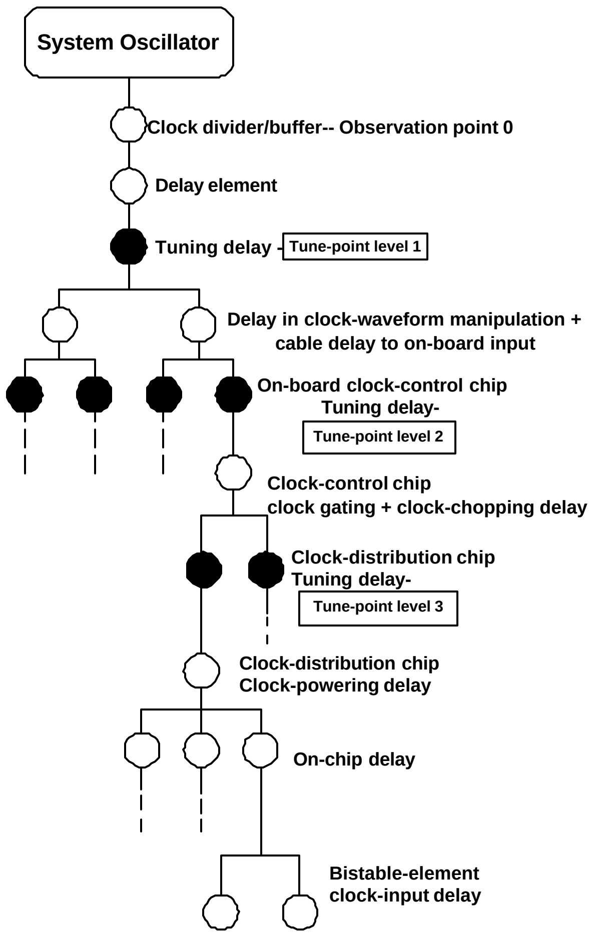

System Oscillator
Clock divider/buffer-- Observation point 0
Delay element
Tuning delay -
Tune-point level 1
Delay in clock-waveform manipulation + cable delay to on-board input
On-board clock-control chip Tuning delay-
Tune-point level 2
Clock-control chip clock gating + clock-chopping delay
Clock-distribution chip Tuning delay-
Tune-point level 3
Clock-distribution chip Clock-powering delay
On-chip delay
Bistable-element clock-input delay

Fig. 15. Clock Tuning Points [Wagner, D&T]

With the advent of integration the systems have shrunk dramatically in size. Today a processor including several levels of cache memory contained entirely on a VLSI chip is quite common. The capacity of a VLSI chip for hundreds of millions of transistors makes it possible to integrate not only one processor but also a multi-processor system onto a single chip. The inability to introduce tuning elements on the chip further aggravates the problem of distributing the clock signals precisely in time since it is not possible to make further adjustment to the clock signal once it has crossed the boundaries of the VLSI chip. Therefore a careful planning and design of the on-chip clock distribution network is one of the most critical tasks in a high performance processor design.

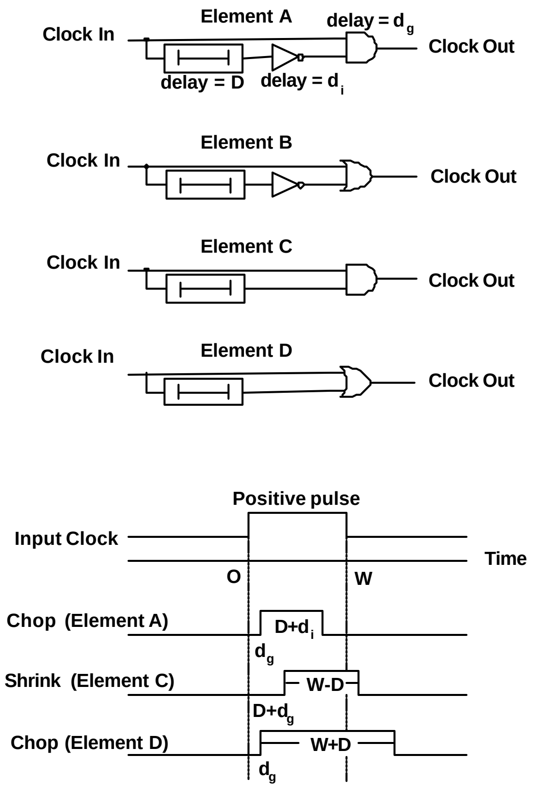
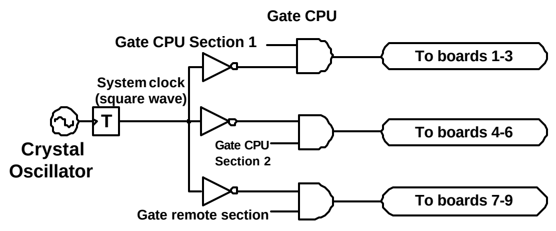
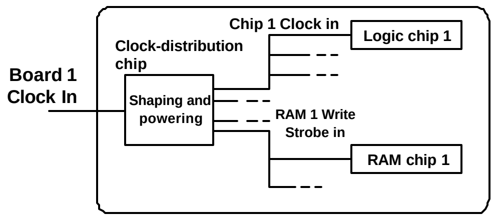


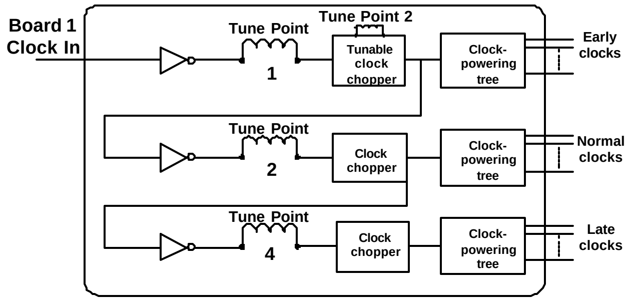
Fig. 16. Various clock shaping elements and obtained clock signals [Wagner]



(a)



(b)

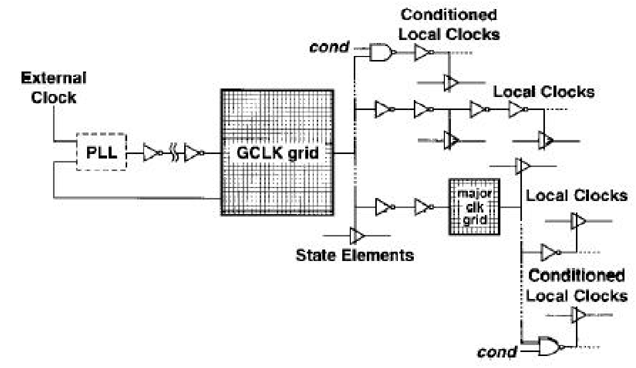


(c)

Fig. 17. Clock distribution network within a system (a), on the board (b) tuning of the clock (c) [Wagner D&T]

Typically on a complex processor chip the clock signal has to be distributed to several hundreds of thousands of the clocked timing elements (known as flip-flops and latches). Therefore, the clock signal has the largest fan-out of any node in the design, which requires several levels of amplification (buffering). As a consequence of such a load imposed to the clock signal, the clock system by itself can use up to 40-50% of the power of the entire VLSI chip [Alpha Gronowsky-98]. However, it is not only the power that represents the problem associated with the distribution of the clock signals. Since we are dealing with synchronous systems we must assure that every clocked storage element receives the clock signal precisely at the same moment in time. Tracing the path of the clock signal from its origin, entry point to the VLSI chip to different clocked storage elements receiving it, the clock signal traverses different paths on the VLSI chip. Those paths may differ quite a bit in several attributes such as: the length of the path (wire), the physical properties of the material along different paths, the differences in clock buffers on the chip as a consequence of the process variations and general effects of non-uniformities of the chip and of the process. The negative effect of those variations on the synchronous design is that different points on the chip will receive the clock signal at different moments in time. This is known as "the clock skew" and will be defined in more precise terms later in this book.

There are several methods for the on-chip clock signal distribution that attempt to minimize the clock skew and attempt to contain the power dissipated by the clock system. The clock can be distributed in several ways of which it is worth to consider the two typical cases: (a) an RC matched tree and (b) a grid shown in Fig. 14.



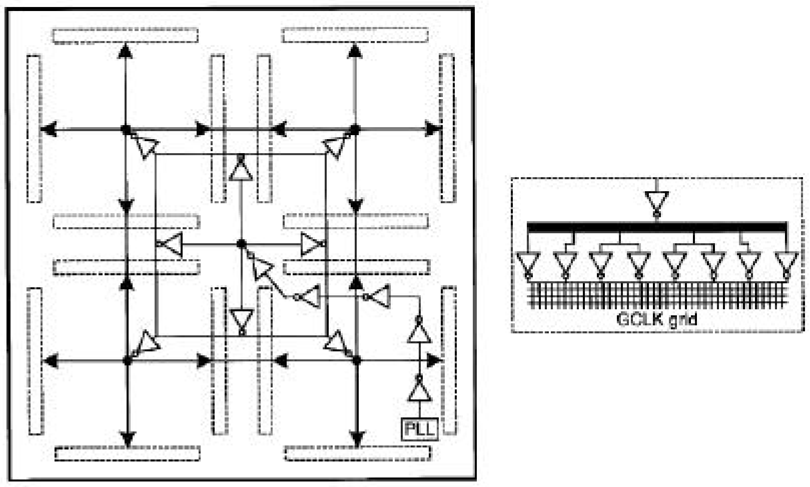


Fig. 18. Clock distribution methods: (a) an RC matched tree and (b) a grid

An RC matched tree (a) is a method of assuring (to the best of our abilities) that all the paths in the clock distribution tree have the same delay which includes the same resistance-capacitance as well as the same number of equal size buffers on the clock signal path to the storage element. There are several different topologies used to implement and RC matched tree. The common objective is to do the best possible in balancing various clock signal paths across the variations points on the VLSI chip. An example of four different topologies (as taken from Bailey [Ananthas book] is shown in Fig. 15.

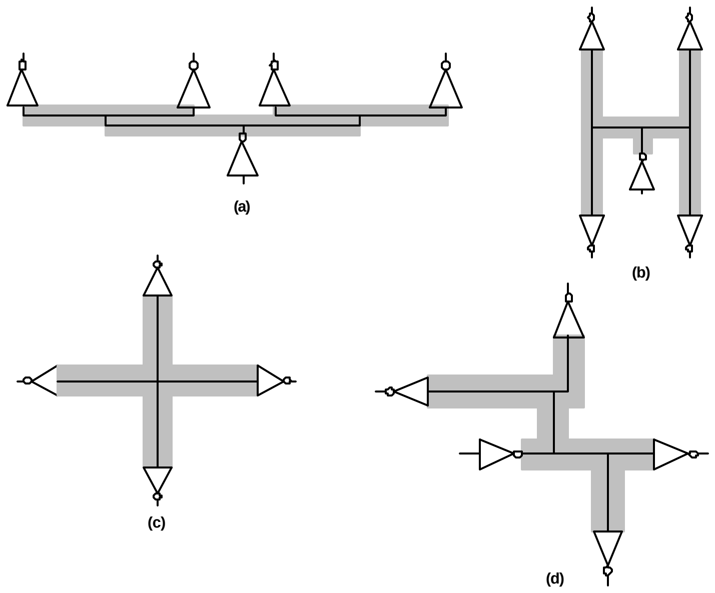


Fig. 19. Different topologies of RC delay matched clock distribution: (a) a binary tree (b) and H tree (c) and X tree (d) an arbitrary matched RC matched tree [Bailey]

If we had superior Computer Aided Design (CAD) tools, a perfect and uniform process and ability to route wires and balance loads with a high degree of flexibility, a matched RC delay clock distribution (a) would be preferable to grid (b). However, neither of that is true. Therefore grid is used when clock distribution on the chip has to be very precisely controlled. This is the case in high performance systems. One such example is DEC Alpha processor, which was a speed champion for several generations of microprocessors starting with their first 200MHz design introduced in 1992 and ending with 600MHz design in 1998 [references to alpha]. The picture of the clock distribution grid together with the clock skew is shown in Fig. 16. However, the power consumed by the clock is also the highest in cases using grid arrangement. This is not difficult to understand given that in a grid arrangement a high-capacitance plate has been driven by buffers connected at various points.

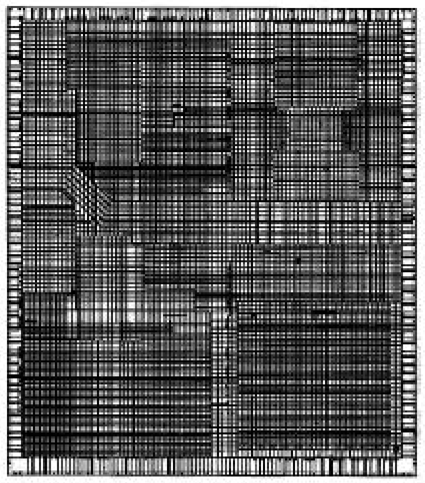


Fig. 20. Clock distribution grid used in DEC Alpha 600MHz processor [Jo SSC, Nov 98]

Timing parameters

It is appropriate at this point to consider the clock distribution system and define the clock parameters that will be used thought this text. For the purpose of the definition we should start with the Fig. 17 showing timing parameters for a single-phase clock.

The clock signal is characterized by its period *T* which is inversely proportional to the clock frequency *f*. The time during which the clock is active (assuming logic 1 value) is defined as *clock width W*. The ratio of W to T-W is also defined as *clock duty cycle*. Usually clock signal has a symmetric shape, which implies 50-50 duty cycle. This is also the best we can expect, especially when distributing a high frequency clock. Another important point is the ability to precisely control the duty cycle. This point is of special importance when each phase of the clock is used for the logic evaluation, or when we trigger the clock storage elements on each edge of the clock (as we will see later in the book). Some recently reported work demonstrates the ability to control the duty cycle to within +-.5% [Alpha600-Jossc].

There are two other important timing parameters that we need to define: *Clock Skew* and *Clock Jitter*.

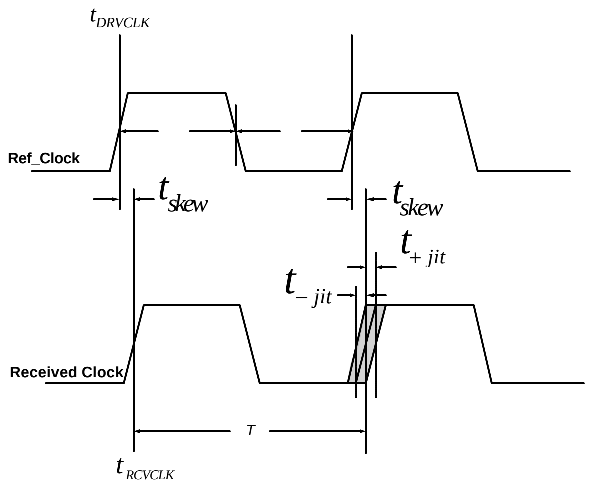


Fig. 21. Clock Parameters: Period, Width, Clock Skew and Clock Jitter

Clock Skew

Clock skew is defined as a spatial variation of the clock signal as distributed thought the system. The clock skew is measured from some reference point in the system: the clock entry point to the board or VLSI chip, or the central point from where the clock distribution starts. Due to the various RC characteristics of the clock paths to the various points in the system, as well as different loading of the clock signal at different points the clock signal arrives at different time to different points. This difference measured from the reference point to the particular timing element is defined as the clock skew. Further we can distinguish global clock skew and local clock skew. Our definition of the clock skew describes global clock skew. Clocks skew occurring between two adjacent clocked storage elements that are connected and with no logic in-between can represent a problem of data race-through. Therefore characterizing a maximum clock skew between two adjacent timing elements is important. A maximum clock skew between two adjacent timing elements is defined as local clock skew. Both of them are equally important in high-performance system design.

Clock Jitter

Clock jitter is defined as temporal variation of the clock signal with regard to the reference transition (reference edge) of the clock signal as illustrated in Fig. 17. Clock jitter represents edge-to-edge variation of the clock signal in time. As such clock jitter can also be classified as: *long-term jitter* and *edge-to-edge clock* *jitter*, which defines clock signal variation between two consecutive clock edges. In the course of high-speed logic design we are more concerned about *edge-to-edge clock jitter* because it is it is this phenomena that affects the time available to the logic. Long term jitter usually affects the processes associated with communication and synchronization between various blocks within a system that need to operate in synchrony with each other.

Theory of Storage Elements

The function of *storage elements*: flip-flops and latches, is to capture the information at a particular moment in time and preserve it as long as it is needed by the digital system. Having said so, it is not possible to define a storage element without defining its relationship to some mechanism in a digital system, which is used to determine time. This definition is general and should include various ways of implementing a digital system, including asynchronous systems. More particularly the element that determines time in a synchronous system is the *clock*.

Latch based Storage Elements

A simplest storage element consists of an inverter followed by another inverter providing a positive feedback. The information bit at the input is thus locked do to the feedback loop and it can be only changed "by force" – i.e. by forcing the output of the feedback inverter to take another logic value. This configuration is very frequently used and is also known as *"keeper"* – a circuit that *keeps* (preserves) the information on a particular node.

If we were to avoid the power dissipation associated with overpowering (forcing) the keeper to change its value we must introduce nodes that will help us in changing the logic value stored in the feedback loop. For that purpose we are free to use logic NAND or NOR gates, as shown in Fig. 18. Particularly interesting is a simple modification of the diagram, which highlights the Sum-of-Products nature of this logic topology shown in Fig. 18 (c). This topology will lead us later to the "Earl's Latch" which was used extensively in IBM mainframe machines [Ealr-Halin-Flyn].

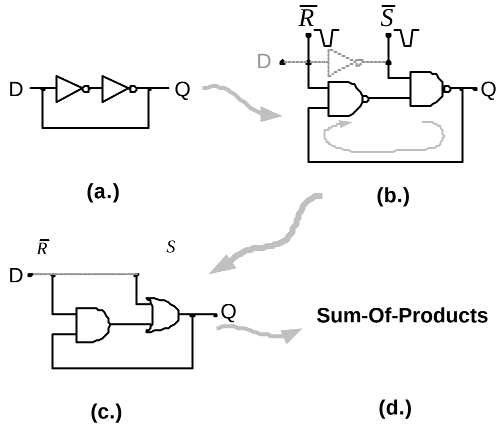


Fig. 22. Latch structure: (a.) keeper (b.) S-R latch (c.) SOP latch (d.) derivation

It is easy to derive a Boolean equation representing a behavior of presented S-R of latch. It is important to note that the next output Q_{n+1} is a function of Q_n, S and R signals. Later in this book we will exploit those simple dependencies in order to design improved clocked storage elements. Presented S-R latch can change the output Q at any point in time. In order to make it compatible with the synchronous design we will restrict the time when Q can be affected by introducing the clock signal which gates S and R inputs. If the data input D is connected to S, and the property of S-R latch, which makes S and R mutually exclusive is applied, the resulting D latch is shown in Fig. 19 (a). The associated timing diagram of a D-Latch is shown in Fig. 19 (b). The latch is "*transparent*" during the period of time in which clock is "*active*" – i.e. assuming logic 1 value.

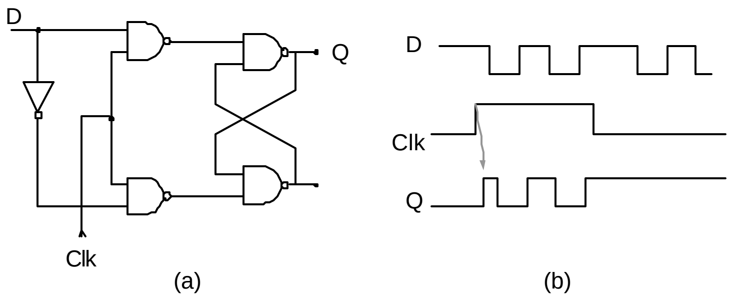


Fig. 23. (a) Clocked D-Latch (b) timing diagram of clocked D-Latch

The realization that a latch can be built in a Sum-of-Product topology (Fig. 18. (c)) tells us that is possible to incorporate logic into the latch, given that the Sum-of-Products is one of the basic realization of the logic function. This leads us to the "Earl's Latch" which was invented in the course of development of a well-known IBM S360/91 machine [reference to IBM 360]. Basic Ealr's Latch configuration is shown in Fig. 20 (a), while a latch implementing Carry function is shown in Fig. 20 (b).

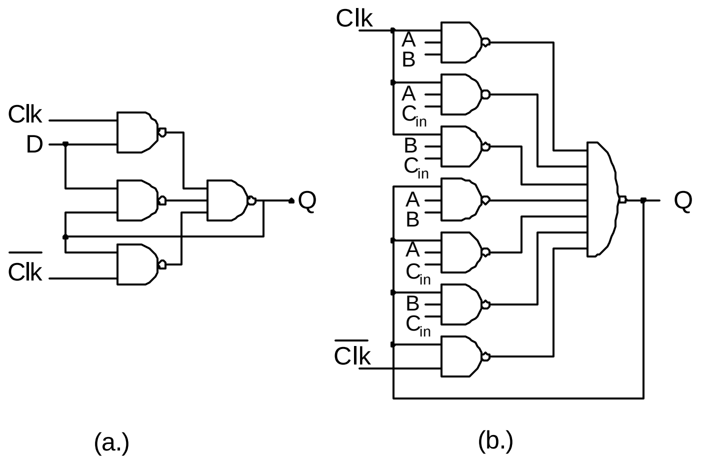
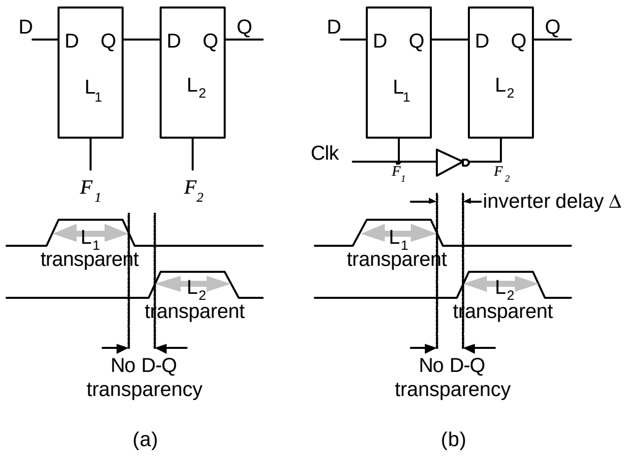


Fig. 24. Basic "Earl's Latch" (a) Implementing "Carry" function (b)

In order to avoid the *transparency* feature introduced by the latch, an arrangement is made in which two latches are clocked back to back with two non-overlapping phases of the clock. In such arrangement the first latch serves as a "*Master*" by receiving the values from the Data input and passing them to the "*Slave*" latch, which simply follows the "*Master*". This is known as a Master-Slave (M-S) Latch arrangement or L1 – L2 latch (in IBM). This is not to be confused with the "*Flip-Flop*", though it seems that many practitioners today do erroneously call this arrangement a Flip-Flop (F-F). We will insist on the terminology that distinguishes Flip-Flop from M-S Latch and we will explain the fundamental differences between the F-F and M-S Latch later in this book.



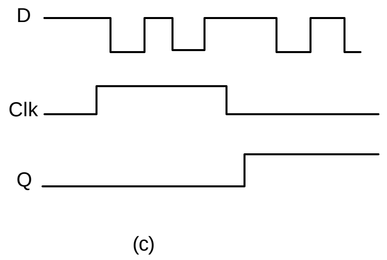


Fig. 25. Master-Slave Latch arrangement with: (a) non-overlapping clocks (b) single external clock (c) timing diagram.

In a Master-Slave arrangement the "Slave" latch can have two or more masters acting as an internal multiplexer with storage capabilities. The first "*Master*" is used for capturing of data input while the second Master be used for other purposes and can be clocked with a separate clock. One such arrangement, which utilizes two Masters is a well known IBM Level-Sensitive-Scan-Design (LSSD).

In LSSD design (shown in Fig. 40 and 41) during the normal operation the system is clocked with clocks C and B and the storage elements are acting as standard M-S latches. However, all storage elements in the system are interconnected in a long shift register using the alternate Master. The input and the output of such shift-register are brought out to the external pins. In the test mode the system is clocked with A and B thus, acting as a long shift register so that the state of the machine can be *scanned out* of the system and/or a new state *scanned in*. This greatly enhances the *controllability* and *observability* of the internal nodes of the system. LSSD became a mandated standard practice of all IBM designs and it has migrated into the industry as a "*Boundary Scan*" IEEE Standard 1149.

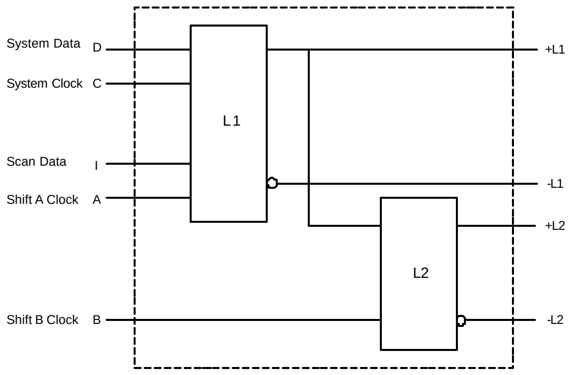


Fig. 26. IBM LSSD compatible storage element

True-Single-Phase-Clock (TSPC) Latch

TSPC Latch (Fig. 21), developed by Afghahi and Svensson [Afghahi, Svensson] is a fast and simple structure that uses a single-phase clock. This latch was constructed by merging two parts consisting of CMOS Domino and CMOS NORA logic. During the active clock (Clk=1), CMOS Domino evaluates the input in a monotonic fashion (only a transition from logic 0 to 1 is possible), while NORA logic is pre-charging. Alternatively during inactive clock (Clk=0) Domino is being pre-charged (thus non-transparent) while NORA is evaluating its input. The combination of NORA and Domino logic blocks results in a Master-Slave Latch that requires only a single clock (TSPC).

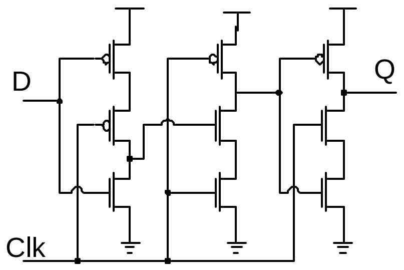


Fig. 27. True Single Phase Clock (TSPC) Latch introduced by Afghahi and Svenson [Afgh-Svenson]

The operation of TSPC latch is illustrated in Fig. 22. While Clk=0, the fist inversion stage is transparent and the second half of TSPC is pre-charged. Thus, at the end of the half-cycle during which Clk=0, the input D is present at the input of the Domino block as its complement $\overline{D}$. When the clock switches to logic 1 (Clk=1), Domino logic evaluates and the output $\overline{Q}$ either stays at logic 0 or makes transition from 0 to 1 depending on the sampled input value $\overline{D}$. This transition cannot be reversed until the next clock cycle. In effect the fist inverter connected to the input acts as a "Master Latch", while the second (Domino) stage acts as a "Slave Latch". The transfer from Master Latch to Slave Latch occurs while the clock changes its value from logic 0 to logic 1. Thus, TSPC behaves as a "raising edge" triggered Flip-Flop. It is also frequently called a Flip-Flop, though by the nature of TSPC operation this classification is incorrect.

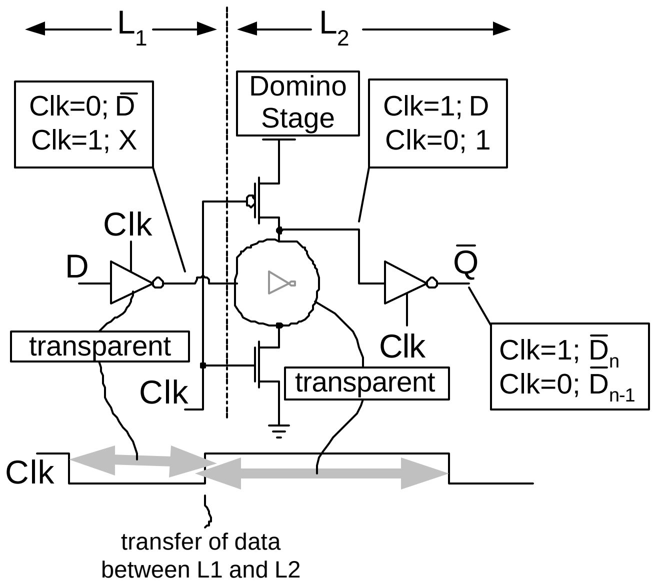


Fig. 28. TSPC Latch operation

Due to its simplicity and speed TSPC was very popular way of implementing clocked storage element. However, TSPC structure suffered a drawback exhibited in sensitivity to glitches created by the clock. This glitch is exhibited on the output holding a logic value of "1", while the input is transitioning from D=0 to D=0.

Pulse Register Single Latch

Recognizing the overhead imposed by Master-Slave latch design and the hazards introduced by a single-latch design, an idea of a single latch design clocked by locally generated short pulses evolved. The idea is to make a clock pulse very short and thus reduce the time window during which the latch is transparent. There still exist a hazard of "short paths" that may be captured during the same clock. Given that the clock pulse is short this hazard is greatly reduced and it is possible to "padd" (add inverters) those paths so that they would not represent a problem. However, such a short clock cannot be distributed globally because the clock distribution network would absorb it. There is also a danger because due to the process variations the duration of that clock pulse will vary from place to place and from chip to chip. Therefore the pulse clock is generated locally and it usually drives a register consisting of several such single-latches physically located very close to each other. It is obvious that this method would loose its advantages if every single latch would require separate clock generator as seen from Fig. 23 (a) and (b).

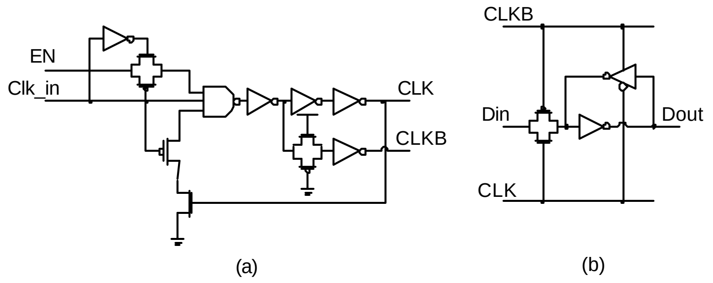


Fig. 29. Pulse Latch: (a) local clock generator, (b) single latch

The produced by local clock generator must be wide enough to enable the latch to latch its data. In the same time it must be sufficiently short to minimize the possibility of "critical race". Those conflicting requirements make use of such single-latch design hazardous reducing the robustness and reliability of such design. Nevertheless, such design has been used in some processors due to the critical need to reduce cycle overhead imposed by the clocked storage elements. Another advantage benefit of this design is low power consumption due to the common clock signal generator and simple structure of such a single latch.

Flip-Flop

The Flip-Flop and the Latch operate on different principles. While a Latch is *"level-sensitive"* which means it is reacting on the *level* (logical value) of the clock signal, Flip-Flop is *"edge sensitive"* which means that the mechanism of capturing the data value on its input is related to the changes of the clock. Thus, the two are designed to a different set of requirements and thus consist of inherently different circuit topology. Level sensitivity implies that the latch is capturing the data value during the entire period of time when clock is active (logic one), thus the latch is *transparent*. The capturing process in the Flip-Flop occurs only during the transition of the clock (from zero–to-one or from one-to-zero), thus the Flip-Flop is not transparent. In fact even the Flip-Flop can have a very small period of transparency associated with the narrow window during which the clock changes, as it will be discussed later. In general we treat Flip-Flop as a *non-transparent* clocked storage element. Given that the triggering mechanism of a Flip-Flop is the transition of the clock signal, there are several ways of deriving it from the clock. For better understanding it pays to look at an early version of the Flip-Flop as used in early computers and digital systems shown in Fig.21. The pulse, which causes the change, is derived from the clock by using a simple differentiator consisting of a capacitor C and resistor R. One can also understand a danger introduced by the Flip-Flop. If the clock transition is slow such a derived pulse may not be capable of triggering the Flip-Flop. On the other hand, even a small glitch on the clock line may cause false triggering.

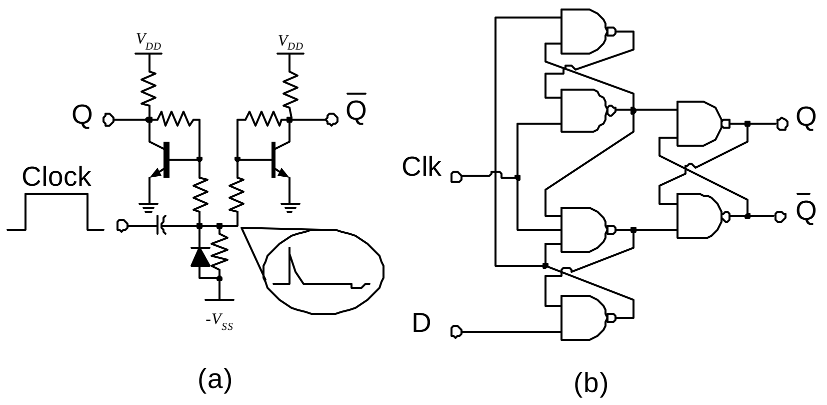


Fig. 30. Early version of a Flip-Flop (a) Texas Instrument SN7474 (b)

A general structure of the Flip-Flop is shown in Fig. 22. It is worth noting the difference between a Flip-Flop structure and that of the M-S Latch arrangement shown in Fig. 19. A Flip-Flop consists of two stages: (a) Pulse Generator - PG (b) Capturing Latch - CL. The pulse generator PG generates a negative pulse on either S-not or R-not lines which are normally held at logic “one” level. This pulse is a function of Data and Clock signals and is of sufficient duration to be captured in the capturing latch CL. The duration of that pulse can be as long as half of the clock period or it can be as short as one inverter delay. On the contrary M-S Latch generally consists of two identical clocked latches and its non-transparency feature is achieved by phasing of the clocks C_1 and C_2 clocking master latch L_1 and slave latch L_2.

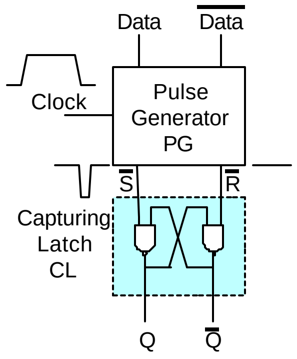


Fig. 31. General Flip-Flop structure

Particularly interesting is SN7474 Flip-Flop that was introduced by Texas Instrument as shown in Fig.26 (b). In order to behave as a Flip-Flop (sensitivity to the change of the raising edge of the clock), an intricate race is introduced in the PG block that prevents any change on S-not and R-not lines after the clock has transitioned from logic "zero" to "one". Analysis of the PG block of SN7474 can be done with help of Fig.28 (a) Delay mismatch that can occur due to the process variations can result in malfunctioning of this Flip-Flop as shown in Fig. 28 (b).

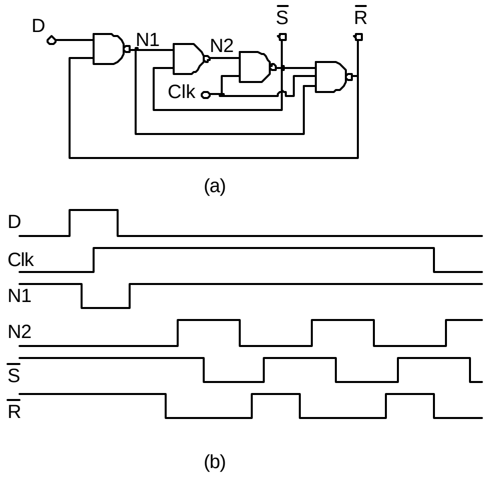


Fig. 32. Pulse Generator block of SN7474 (a) malfunctioning due to a gate delay mismatch (b)

The relationship of S-not and R-not signals with respect to Data (D) and Clock (Clk) signal can be expressed as:

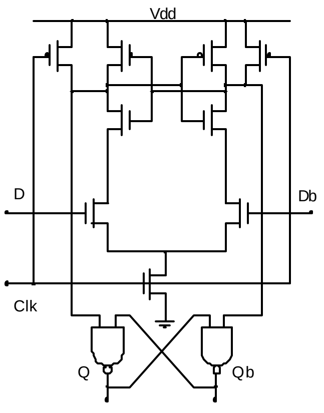
$$S_n = Clk\,\overline{R}(D+S) \qquad \text{and} \qquad R_n = Clk\,\overline{S}(\overline{D}+R) \tag{1}$$

The expressions for the next value of the set signal S_n (as well as reset signal R_n) provide a quick and simple insight into the functioning of the PG block of this Flip-Flop. Simply stated in words the equation for S_n tells us the following:

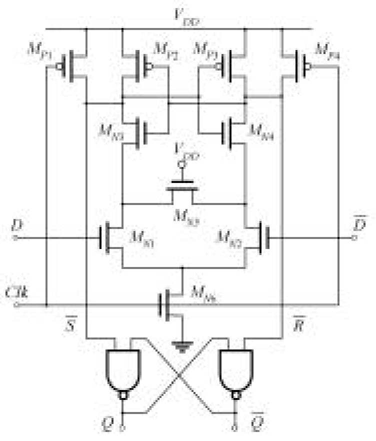
The next state of this Flip-Flop will be set to "one" only at the time the clock becomes "one" (raising edge of the clock), the data at the input is "one", the flip flop is in the "steady state" (both S and R are "zero"). The moment Flip-Flop is set (S=1, R=0) no further change in data input can affect the Flip-Flop state: data input will be "locked" to set by (D+S)=1, and reset R_n would be disabled (by S=1).

This assures the "*edge sensitivity*" – i.e. after the transition of the clock and setting of the S or R signal to its desired state, the Flip-Flop is "locked" and no changes can occur until clock transition to "zero" (making both S=R=0), thus enabling the Flip-Flop to receive the new data.

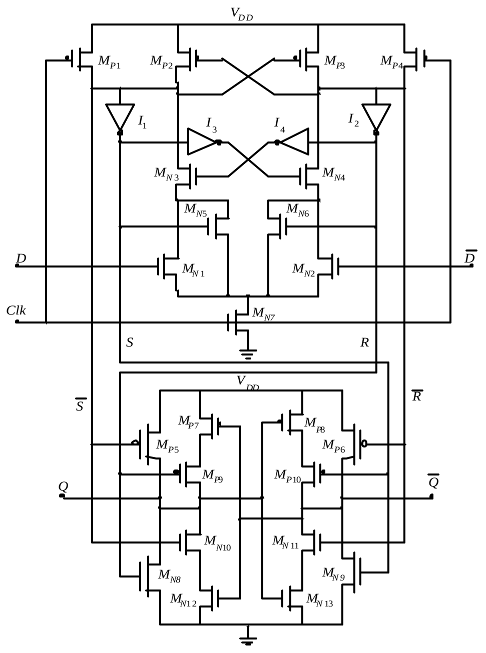
It is interesting to note that it took engineers several attempts to come to the right circuits topology of this Flip-Flop. The Flip-Flop used in the third generation of Digital Equipment Corp. 600MHz Alpha processor used a version of the Flip-Flop introduced by Madden and Bowhill, which was based on the static memory cell design [Madden and Bowhill patent]. This particular Flip-Flop is known as Sense Amplifier Flip-Flop (SAFF). Development of the Pulse Generator block of this Flip-Flop is illustrated in Fig.29.



(a)



(b)



(c)

Fig. 33. Pulse Generator stage of the Sense Amplifier Flip-Flop: Madden and Bowhill [patent 4,910,713] (a) Improvement for floating nodes, Doberpuhl [Doberpuhl StrongARM](b) improvement by proper design Nikolic and Oklobdzija [ESSCIRC-99] (c)

The behavior of SN7474 Flip-Flop and Alpha's SAFF is identical. When setting the Flip-Flop both of them hold S-not (or R-not) line at logic "zero" for the duration of the clock active (logic "one") value and reset them to logic "one" once the clock returns to logic "zero" (inactive state).

One of the objectives of this book is to clear up the confusion caused by introduction of various types of clocked storage elements that were introduced under various names and classifications. We will examine another way used in practice to create the Flip-Flop. In SN7474 disabling of the input is done after a short delay necessary to set S (or R) to the next value, thus achieving the "*edge property*". That short delay is essential and cannot be avoided. It is reflected in the Setup and Hold time parameters of the Flip-Flop.

Time Window based Flip-Flops

Digital circuits are based on discrete events. Not only are the logic signals a set of discrete voltage levels, but also the time is based on either the events of the clock (rising or falling edge) or finite delay based on the propagation through one or more logic elements used. Therefore, determining when to shut the Flip-Flop is also based on discrete time events with reference to the clock, such as one or more inverter or gate delay units. The common technique used to generate the time reference signals is to generate a short pulse using the property of *re-convergent fan-outs with non-equal parities of inversion*. This method is illustrated in Fig. 25. The trailing edge of this short pulse is used as a time reference for shutting the Flip-Flop off. By using this short pulse and depending on the particular implementation of the Flip-Flop a possible short transparency window of the Flip-Flop may be introduced. This short transparency has been a stumbling block and a source of misunderstandings in classifying the timing element introduced. One such a Flip-Flop introduced as a Hybrid Latch Flip-Flop (HLFF) is a Flip-Flop, which we will use as an example in this book. The confusion caused by the short transparency caused its inventor to treat it as a latch, but since its behavior was not that of the latch a dual name HLFF was the result of it [Partovi HLFF]. HLFF is shown in Fig. 26.

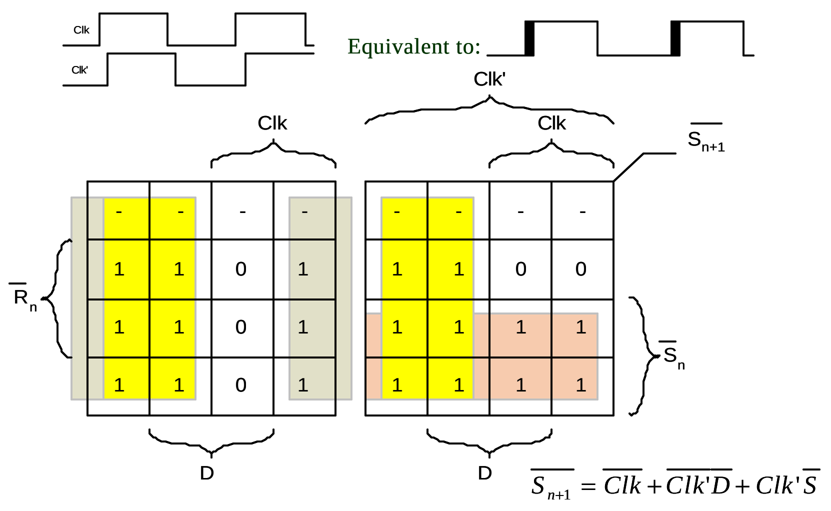


Fig. 34. Method of creating the time reference points for opening and shutting the Flip-Flop

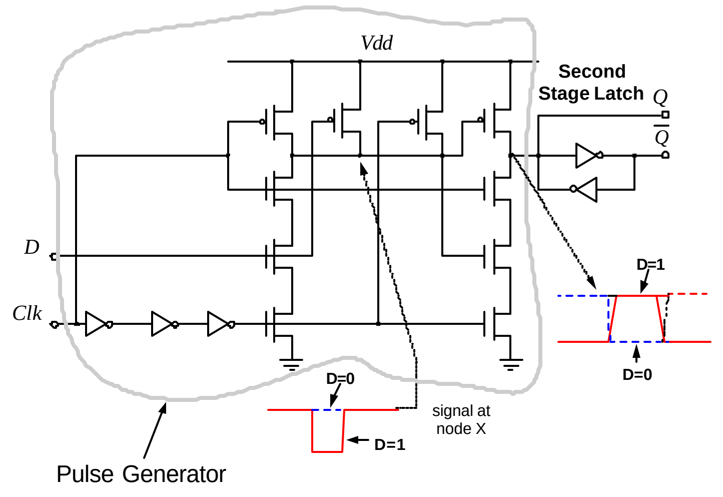


Fig. 35. Hybrid-Latch Flip-Flop (HLFF) introducedby Partovi [Partovi HLFF ISSCC]

Rigorous analysis will show incompleteness of this design, which resulted in imperfections demonstrated by this Flip-Flop. Logic representation of this Flip-Flop shows two NAND gates connected in series (Fig. 27). The first NAND gate creates the pulse if D=1. Data is serving as a pulse enabler or pulse inhibitor, depending of the value of D.

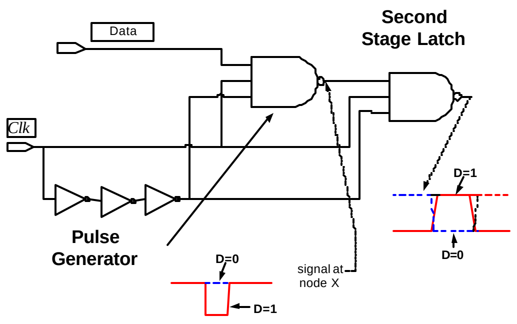


Fig. 36. Logic representation of Partovi's Hybrid-Latch Flip-Flop (HLFF)

The problem with this structure comes from its incompleteness in the second stage. In order to avoid an excessive number of p-MOS transistors the second NAND gate is not complete and its output node is floating when the output node X from the first NAND is at logic "one" and the clock pulse has ended. In essence this node (X) represents the Snot signal from the pulse generator. The absence of the R-not signal, due to the single ended implementation of this Flip-Flop hinders that ability to realize the Flip-Flop function completely, as it the case of complete SAFF [ESSCIRC, Nikolic, Oklobdzija]. This floating output node is susceptible to glitches and even slightest mismatch of clock signals. When data input D=1, the output will first capture X=1 followed by X=0 causing a glitch on its output. This is an inherent problem of HLFF structure.

More systematic approach in deriving a single-ended Flip-Flop is shown in Fig. 28.

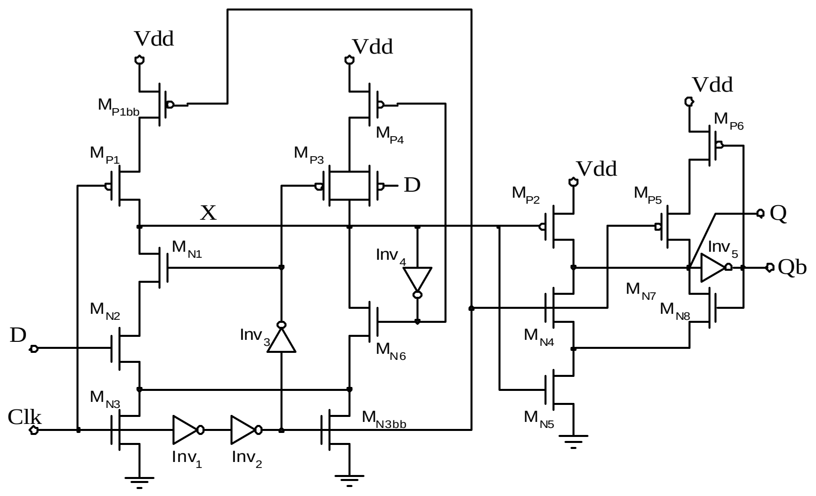


Fig. 37. Systematically derived single-ended Flip-Flop (Nedovic, Oklobdzija) [Nedovic Oklobdzija ICCD2000]

Flip-Flop shown in Fig. 28. has three time reference points: (a) raising edge of the Clk signal, (b) falling edge of the Clk signal after passing through three inverters: Inv$_{1\text{-}3}$, (c) raising edge of the Clk signal after passing through two inverters: Inv$_{1\text{-}2}$. The derivation of this circuit models raising edge triggered Flip-Flop behavior using three time reference points. Those equations describing the behaviour of this Flip-Flop are presented:

The node X is represented as:

$$X = \overline{(Clk + CLKbb) * (D * Clkbbb + \overline{X})} \qquad (2)$$

The nMOS transistor section is a full realization of this equation. The pMOS section can be somewhat abbreviated for performance reasons into:

$$X = \overline{(Clk + CLKbb) * (Clkbbb + \overline{X})} \qquad (3)$$

The second stage(capturing latch) is implemented as:

$$Q = \overline{X * (CLKbb + \overline{Q})} \qquad (4)$$

The clock signal *Clk*, after two inversions is designated as *Clkbb* while after three inversions as *Clkbbb*.

This Flip-Flop does not have hazards and is outperforming HLFF as well as SDFF Flip-Flops [KlassSDFF].

Flip-Flop and Latch parameters

Flip-Flop and Latch do behave differently, however, it is possible to establish some common parameters for both. We will define them on a Flip-Flop and extend this definition to a Latch.

Setup Time

First, we should define an *"active edge"* of the clock as an event in time causing the activity of the storage element. For the purpose of the further discussion we may temporarily assume that the leading edge (transition from logic "zero" to logic "one") of the clock is the "*active edge*".

Setup time is defined as period of time before the active edge of the clock during which the data need to be stable in order to assure a reliable storage of information into the clocked storage element.

Hold Time

Hold Time is defined as a period of time after the active edge of the clock has occurred during which the data is not allowed to change in order to assure reliable storage of information into the clocked storage element.

Both Setup-Time and Hold-Time define a *Sampling Window*, which is the total time during which data must remain stable in order to assure reliable storage of information. Further, we define the time during which clock is active (assuming logic "one" value) as the *Clock Width*. For illustration purposes, Setup Time, Hold Time, Sampling Window and clock Width for a Flip-Flop are shown in Fig. 29.

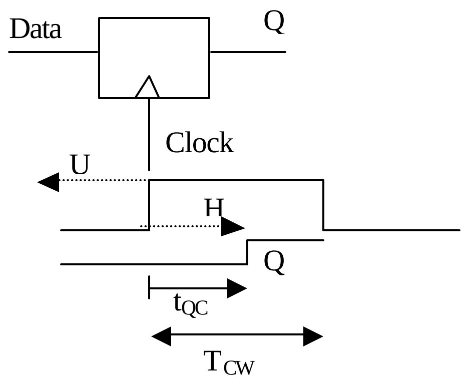


Fig. 38. Definition of: Setup Time, Hold Time, Sampling Window and Clock Width for a Flip-Flop.

The minimal clock width T_{CW} necessary for reliable capture of data into the clocked storage element is not simply equal to the *Sampling Window*, as it may appear, because the clock signal and the data signal may not necessarily have the same paths as well as the same loads inside the clocked storage element.

The situation with the latch is different as illustrated in Fig. 30. The setup time for the latch starts from the trailing edge of the clock signal because closing of the latch is the action that would capture the last data that was present in the latch. In addition there are two delay times defined t_{CQ} (as in the Flip-Flop) and t_{DQ} because of the two possible scenarios: (a) data being present and waiting for the clock to open the latch (b) data arriving while latch is open.

Setup and Hold Time Properties

Having defined Setup and Hold time the question about failing mechanism of the clocked storage element remains. When will the process of storing the information fail due to the Setup or Hold time violation? This is not an abrupt process as the definition of Setup and Hold time implies. If we establish an experiment in which we will delay the data arrival closer to the clock we will observe that at first Clock-to-Q delay (t_{CQ}) of the storage element will start to increase before the capturing mechanism fails. Similar happens on the other end when Hold time is gradually violated. This behavior is shown in Fig. 31. Thus, a valid question is raised: how do we define Setup and Hold time? Is it the moment t_{CQ} starts to raise, the moment t_{CQ} reaches a certain value, or the moment the clocked storage element starts to fail ? Those questions require some careful consideration. Obviously we do not want to allow the data to come to close to the failing region from the fear that we may have an unreliable design. However, keeping the data too far from the failing region takes away our precious cycle time, thus impacting the performance negatively.

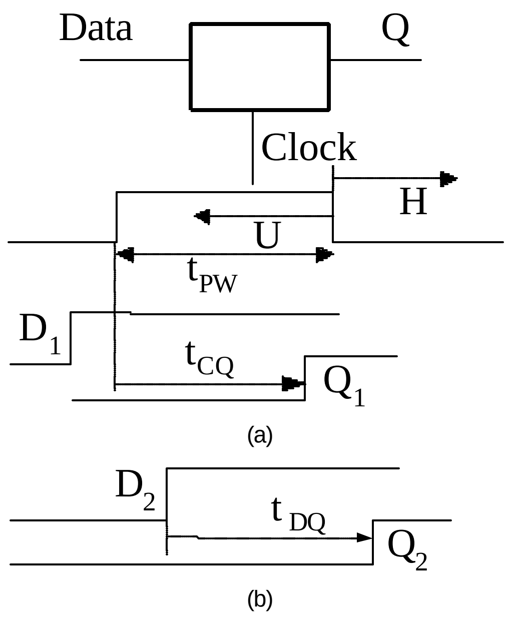


Fig. 39. Latch: Setup and Hold Time: (a) early data D_1 arrival (b) late data D_2 arrival

It would be appropriate to remark that the failure mode of the Flip-Flop does not necessarily follow the failure mode of the Latch as a result of the Setup or Hold time violations. Depending on the Flip-Flop implementation, violation of the Setup or Hold time may lead to oscillations in the Pulse Generator stage of the Flip-Flop. As a result, once the oscillation occurred it is unpredictable what the output value Q to which the Capturing Latch will be set will be. These oscillations in the Flip-Flop usually occur abruptly as opposed to the more gradual delay increase encountered with the latch. Therefore one needs to be much more careful with the Flip-Flop than with the latch based design.

This is obviously a dilemma. Some designs resort to establishing somewhat arbitrary number of 5-20% such that the Setup and Hold times are defined as the points in time when the Clock-to-Q (t_{CQ}) delay raises for that amount. We do not find this reasoning to be valid. A redrawn picture, Fig. 32, where Data-to-Q (tdq) delay is plotted answers this question. From this graph we see that in spite of Clock-to-Q delay rising, we are still gaining because the time taken from the cycle is reduced. In other words the increase is storage element delay is still smaller than the amount of time data is delayed, thus allowing more time in the cycle for the useful logic operation. However, we are starting to encounter new phenomena known under different names such as: "*time borrowing*", "cycle stealing" and "slack passing". We will use the term "*time borrowing*" in the further text. In order to understand the full effects of delayed data arrival we have to consider a pipelined design where the data captured in the first clock cycle is used as input data in the next clock cycle as shown in Fig. 33.

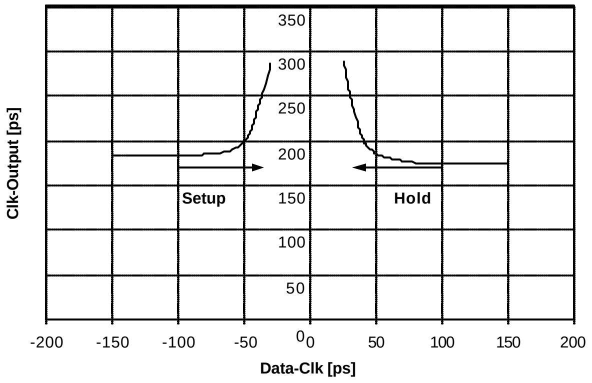


Fig. 40. Setup and Hold time behavior as a function of Clock-to-Output delay

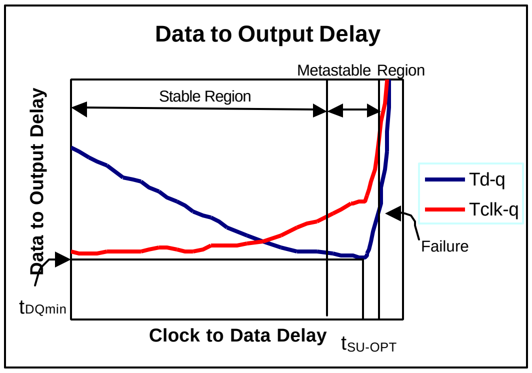


Fig. 41. Setup and Hold time behavior as a function of Data-to-Output delay

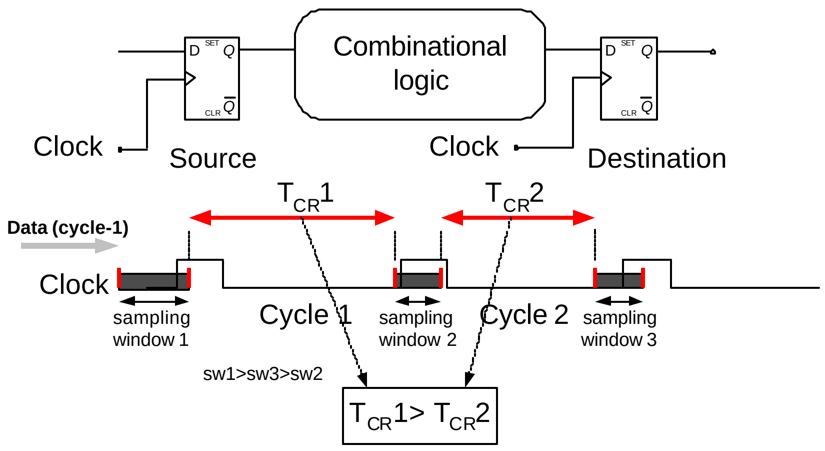


Fig. 42. "*Time Borrowing*" in a pipelined design

As it can be observed from Fig. 33., the "sampling window" moves around the time axes. As the data arrives closer to the clock the size of the "sampling window" shrinks (up to the optimal point). However, even though, the sampling window is smaller, the data in the next cycle will still arrive later compared to the case where the data in the previous cycle was ready well ahead of the setup-time. The amount of time for which the T_{RC1} was augmented did not come for free. It was simply taken away ("stolen" or "borrowed") from the next cycle T_{RC2}. As a result of late data arrival in the Cycle 1 there is less time available in the Cycle 2. Thus a boundary between pipeline stages is somewhat flexible. This feature not only helps accommodate a certain amount of imbalance between the critical paths in various pipeline stages, but it helps in absorbing the clock skew and jitter. Thus, "time borrowing" is one of the most important characteristics of high-speed digital systems.

Pipelining and timing analysis

Analysis of a System with a Single Flip-Flop

For the proper discussion we should analyze the timing situation in a pipelined system. First, we should start with a simplest case of a Flip-Flop and a single clock being used in the design. This situation is illustrated in Fig. 34. Much of this discussion is taken from the paper by Unger and Tan [Unger & Tan] with some slight changes in notation. There are two events that we need to prevent:

(a) Data arriving too late to be captured reliably in the next cycle. There are two possible scenarios here: either the data arrived far too late and is completely is missed in the next cycle, or it just sufficiently late to be violating the setup time requirement of the storage element, thus not assuring reliable capture.

(b) Data arrives too early (during the same cycle), thus violating the hold time requirement for the Flip-Flip.

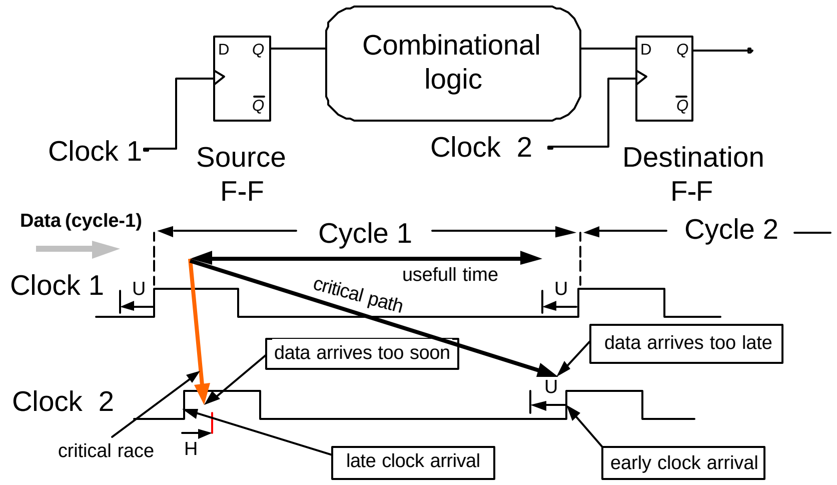


Fig. 43. Timing in a digital system using a single clock and Flip-Flops

In either of the cases we can not assure that the data will be capture in the next cycle, therefore we are not able to guarantee a reliable operation of the system. In order to perform a simple analysis of this system, let us assume that the clock skew and jitter combined can cause a maximal deviation of the leading edge of the clock for T_L amount of time from the nominal time of arrival (and T_T for the trailing edge). If we set the time reference to t=0 for the nominal leading edge of the clock for the Cycle 1 than we have a following relations:

$$t_{DLN} = T_L + D_{CQM} + t_{CR} \tag{5}$$

The latest possible arrival of the data in the next cycle occurs under the following circumstances: (a) data was captured at the latest possible moment due to the clock skew and jitter, (b) the Flip-Flip that captured the data was the slowest possible (keep in mind that Flip-Flop delays will vary due to the process variations), (c) this data traveled through the longest path in the logic (critical path).

$$t_{DLN} = P - T_L - U \tag{6}$$

The clock leading edge in the next cycle arrived at the earliest possible moment *P-*T_L , However, in order to capture the data reliably the data should arrive at least for the setup time *U* before the leading edge of the clock. This leads to the following equality:

$$P - T_L - U \geq T_L + D_{CQM} + t_{CR} \tag{7}$$

From this equality a constraint for the clock period P (speed of the clock) is derived:

$$P \geq 2T_L + U + D_{CQM} + t_{CR} \tag{8}$$

Alternatively for a given clock speed the longest critical path in the logic has to be shorter than:

$$t_{CR} \leq P - 2T_L + U + D_{CQM} \tag{9}$$

This is one of the fundamental equations. Basically it shows that the time available for information processing is equal to the time remaining in the clock period after the clock skew is subtracted for both edges and the time data spend traveling through the storage element.

It is commonly misunderstood that the Flip-Flop provides edge-to-edge timing and is thus easier to use because it does not need to be checked for hold-time violation. This is not true, and a simple analysis that follows demonstrates that even with the Flip-Flop design the *fast paths* can represent a hazard and invalidate the system operation.

If the clock controlling the Flip-Flop releasing the data is skewed so that it arrives late, and the clock controlling the Flip-Flop that receives this data arrives late, a hazard situation exists. This same hazard situation is present if the data travels through a *fast path* in the logic. A *fast path* is the path that contains very few logic blocks, or none at all. Referring to Fig. 34. this hazard, which is also referred to as *critical race* can be described with a following set of equations:

$$t_{DEArr} = -T_L + D_{CQm} + D_{Lm} \tag{10}$$

$$t_{DEArrN} = -T_L + H \tag{11}$$

Equation (10) represents the time of the early arriving signal t_{DEArr}, which should not be earlier than the time described by (11), otherwise there will be a hold-time violation of the data receiving Flip-Flop. This condition is represented by the inequality (12):

$$-T_L + D_{CQm} + D_{Lm} > -T_L + H \qquad (12)$$

In this equations D_{CQm} represents the minimal Clock-to-Q (output) delay of the Flip-Flop and D_{Lm} represents minimal delay through the logic (as opposed to the use of index M where D_{CQM} and D_{LM} represent maximal delays).

Equation (12) gives us a constraint on the fast-paths: i.e. no signal in the logic should be taking the time shorter than D_{LB} otherwise there will be hold-time violation in the circuit.

$$D_{Lm} > D_{LB} = 2T_L + H - D_{CQm} \qquad (13)$$

Further, the clock has to be active for some minimal duration (in order to assure reliable capture of data):

$$W \geq T_L + T_T + C_{Wm} \qquad (14)$$

Equations (9),(13) and (14) provide timing requirements for reliable operation of a system using Flip-Flops.

Analysis of Single-Latch Based System

Single latch based system is more complex to analyze than Flip-Flop based system. However, its analysis is still much simpler than a general analysis of a two latch (Master-Slave) based system, which is shown in [Unger-Tan]. Use of a single latch represent a hazard due to the transparency of the latch, which introduces a possibility of races in the system. Therefore, the conditions for single-latch based system must account for critical race conditions. As the previous analysis showed, presence of the storage element delay decreases the "useful time" in the pipeline cycle. Therefore, in spite of the hazards introduced by such design, the additional performance gain may well be worth the risk.

Some well-known systems such as CRAY-1 super-computer do use single latch. This decision was based on performance reason. Second generation Digital Corp. "Alpha" 21164-processor uses single-latch based design as well. A difference between "Alpha" and "CRAY-1" is in the way single latch has been used in the pipeline. Two ways of structuring the pipeline with the single latch is shown in Fig. 35. In Fig. 35. (a) a straight forward way of using a single latch is shown. Here all the latches in the system are "transparent" while the clock is active (logic 1) and all the latches are "*opaque*" (non-transparent) when the clock is inactive (logic 0).

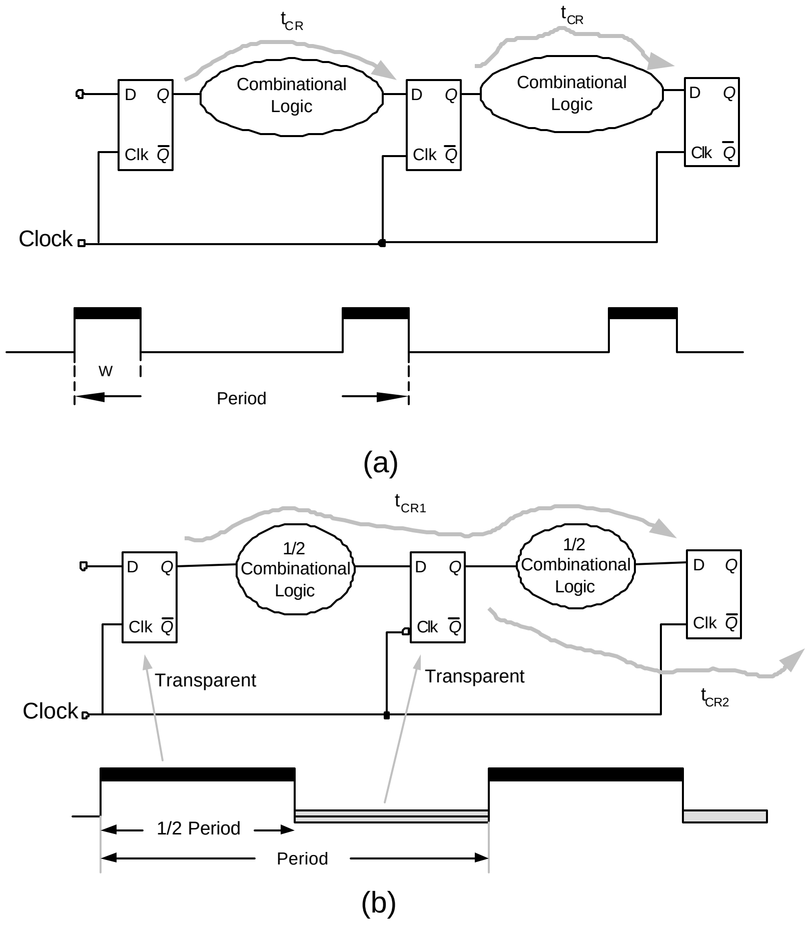


Fig. 44. Two ways of using a single-latch in a single-latch based system

We will base the analysis of a single-latch based design on the well known paper by Unger and Tan [Unger-Tan]. The case (a) is easier to analyze while the case (b) becomes more complex.

In the case of a latch signal need to arrive for setup time U before the trailing edge of the clock (which closes the latch). However, this edge could arrive earlier because of the clock skew. So, the latest arrival of the data that assures reliable capture after the period *P* has to be:

$$t_{DLArr} \le W - T_T - U + P \qquad (15)$$

Data that is to be captured at the end of the clock period could have been a result of two events (whichever later):

(a) data was ready, clock arrived at the latest possible moment T_L and the worse case delay of the latch was incurred which is D_{CQM}.

(b) clock was active and data arrived at the last possible moment, which is a setup time U and clock skew time T_T before the trailing edge of the clock.

In both cases (a) and (b) the path through the logic was the longest path D_{LM}.

Thus the data to be captured in the next cycle, under the worse scenario (either (a) or (b)) has to arrive in time to be reliably captured in the next cycle:

$$\max\{T_L + D_{CQM}, W - T - U + D_{DQM}\} + D_{LM} \ge W - T - U + P \qquad (16)$$

This gives us a constraint for the clock speed in terms of the duration of the period P such as:

$$P \ge \max\{T_L + T_T + U + D_{CQM} - W, D_{DQM}\} + D_{LM} \qquad (17)$$

This inequality breaks down into two inequalities (18) and (19):

$$P \ge D_{LM} + D_{CQM} + T_L + T_T + U + -W \qquad (18)$$

$$P \ge D_{LM} + D_{DQM} \qquad (19)$$

Equation (19) shows the minimal bound for P_m, which is the time to traverse the loop consisting of the maximal delay of the data passing through latch and through the longest path in the logic. In other words:

"Starting from the leading edge of a clock pulse, there must be time, under worse case, before the trailing edge of the clock, for a signal to pass through the latch and the logic block in time to meet the setup time constraint" [Unger-Tan].

The value of $P = P_m$ determines the highest frequency of the clock for that system. However, this does not come without price. Given that the loop through the logic and the latch is open, we have to assure that any of the "*fast paths*" that may exist in the logic does not arrive sooner than the next period of the clock. This leads to the following analysis for fast paths.

The fastest signal, traveling through the fastest path in the logic, should not arrive before the latest possible arrival of the same clock:

$$t_{DEArrN} \ge t_{CLT} + H = W + T_T + H \qquad (20)$$

There are two possible scenarios for the early arrival of the fast signal: (a) it was latched early and it passed through a fast path in the logic, or (b) it arrived early while the latch was open and passed through the fast latch and fast path in the logic. This is expressed in equation (21):

$$t_{DEArrN} = \min\left\{t_{CEL} + D_{CQm}, t_{DEArr} + D_{DQm}\right\} + D_{Lm} \tag{21}$$

The earliest arrival of the clock t_{CEL} happen if the leading edge of the clock is skewed to arrive early $-T_L$. Thus, the conditions for preventing race in the system is expressed as:

$$\min\left\{-T_L + D_{CQm}, t_{DEArr} + D_{DQm}\right\} + D_{Lm} \geq W + T_T + H \tag{22}$$

It is obvious that the earliest possible arrival of the clock plus clock-to-output delay of the latch has to be earlier in time than early arrival of the data (while the latch is open) plus data-to-output delay of the latch. Thus:

$$-T_L + D_{CQm} + D_{Lm} \geq W + T_T + H \tag{23}$$

which gives us a lower bound on the permissible signal delay in the logic:

$$D_{Lm} > D_{LmB} \geq W + T_T + T_L + H - D_{CQm} \tag{24}$$

Thus the conditions for reliable operation of a system using a single latch are described by equations (18),(19) and (24) which are repeated here for clarity:

$$P \geq D_{LM} + D_{CQM} + T_L + T_T + U + -W \tag{18}$$

$$P \geq D_{LM} + D_{DQM} \tag{19}$$

$$D_{Lm} > D_{LmB} \geq W + T_T + T_L + H - D_{CQm} \tag{24}$$

One can notice that the increase of the clock width *W* is beneficial for speed (18), but it increases the minimal bound for the fast paths (24). Maximum useful value for *W* is obtained when the period *P* is minimal (19). Substituting *P* from (19) into (18) gives us this value of *W*:

$$W^{opt} = T_L + T_T + U + D_{CQM} - D_{DQM} \tag{24}$$

If we substitute the value of the optimal clock width W^{opt} into (25) than we will obtain the values for the maximal speed (19) and minimal signal delay in the logic which has to be maintained in order to satisfy the conditions for optimal single-latch system clocking:

$$P \geq D_{LM} + D_{DQM} \tag{19}$$

$$D_{LmB} = 2(T_T + T_L) + H + U + D_{CQM} - D_{CQm} - D_{DQM} \quad (26)$$

It may be worthwhile giving a thought to those equations. What they tell us is that under ideal conditions, if there are no clock skews and no process variations, the fastest path through the logic has to be greater than the sampling window of the latch ($H+U$) minus the time the signal spend traveling through the latch. If the travel time through the latch D_{DQM}, is equal to the sampling window, than we do not have to worry about fast paths. Of course this is the ideal situation and in practice we do have to take a good care of both: *fast* and *slow* paths in the logic.

In summary in a single latch system, it is possible to make the clock period P as small as the sum of the delays in the signal path: latch and critical path delay in the logic block. This can be achieved by adjusting the clock width W and assuring that all the fast paths in the logic are larger in their duration than some minimal time D_{LmB}.

Analysis of a System using Two-Phase Clock and Two Latches in Master-Slave Arrangement

A particular version of the use of two latches in the Master-Slave (M-S) arrangement is the most commonly used technique in digital system design. It is also a robust and reliable technique compatible with the Design for Testability (DFT) methodology. We will start with describing the most general arrangement consisting of two latches clocked by two separate and independent clocks C_1 and C_2 as shown in Fig. 36.

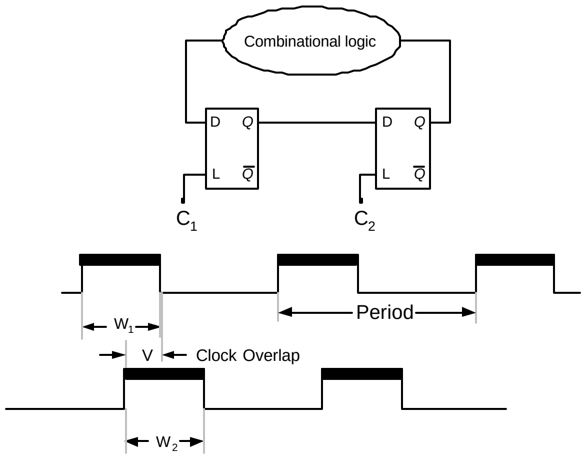


Fig. 45. System using Two-Phase Clock and Two Latches in Master-Slave Arrangement

The analysis of a system using two-phase clock is much more complex compared to the system using a single clock, because we are introducing possible skew on the second clock. Therefore the set of parameters includes clock skew on the leading and trailing edge of the first clock C_1: T_{1L} and T_{1T} and on the second clock C_2: T_{2L} and T_{2T}. In addition, the overlap between C_1 and C_2: V is to be taken into account as well as the corresponding width of the clock pulses: *W1* and *W2*.

This analysis tends to be tedious and complex. Therefore a detailed reading of the paper by Unger and Tan [Unger-Tan] is suggested to the interested reader. Without going into details of that analysis, we will only present here qualitative analysis and final derivations.

From the latest signal arrival analysis several conditions can be derived. First, we need to assure orderly transfer into L_2 latch (*Slave*) from the L_1 latch (*Master*) even if the signal arrived late (in the last possible moment) into the (*Master*) L_1 latch. This analysis yields the following two conditions:

$$W_2 \geq V + U_2 - U_1 + D_{1DQM} + T_{2T} - T_{1L} \tag{27}$$

$$W_1 + W_2 \geq V + U_2 + D_{1CQM} + T_{1T} + T_{2T} \tag{28}$$

Those conditions assure timely arrival of the signal into the L_2 latch, thus an orderly L_1-L_2 transfer (from *Master* to *Slave*).

The analysis of the latest arrival of the signal into L_1 latch in the next cycle (critical path analysis) yields to the following set of equations:

$$P \geq D_{1DQM} + D_{2DQM} + D_{LM} \tag{29}$$

The equation (29) gives us the *highest frequency* at which the system can operate. In other words, the minimal period of the clock P has to be of sufficient duration to allow for the signal to traverse the loop consisting of: L_1 latch, L_2 latch and the longest path in the logic D_{LM}.

$$W_1 \geq -P + D_{1CQM} + D_{2DQM} + U_1 + D_{LM} + T_{1L} + T_{1T} \tag{30}$$

The condition specified in equation (30) assures timely arrival of the signal that starts on the leading edge of C_1, traverses the path through L_2, the longest path in the logic and arrives before the trailing edge of C_1, in time to be captured.

If the signal, starting from the leading edge of C_2 (prior to the end of C_1) traversing L_2 and the longest path in the logic is to be captured in time in L_1, then the condition (31) needs to be satisfied.

$$P \geq -V + D_{2CQM} + U_1 + D_{LM} + T_{1T} + T_{2L} \qquad (31)$$

The equation (31) shows that the amount of overlap V between the clocks C_1 and C_2 has some positive effect on speed. The overlap V allows the system to run at greater speed. Conversely, if we increase V we can tolerate longer "*critical path*" D_{LM}. Thus, the increase of V is beneficial for he system. However, the increase of the clock overlap has its negative effects and obviously its limit.

One of the negative consequences is that overlapping clocks introduces a possibility of race conditions, thus requiring a *fast path* analysis. The analysis of *fast paths* (or *critical races*) makes the timing analysis much more complex and in general computer aided design tools do not perform this analysis very well. It is for that reason that many would sacrifice some performance for reliability and ease of design. One very commonly used clocking methodology is theuse of *Master-Slave* (L_1-L_2) latches with locally generated C_2 clock. Such arrangement assures reliability since C_1 and C_2 clocks are not overlapped, thus eliminating the need for the analysis of critical races. This arrangement is shown in Fig. 37. Unfortunately, this arrangement is also a cause of widely spread misuse of the term "*Flip-Flop*" for the structure, which is nothing other, but a *Master-Slave* (L_1-L_2) latch.

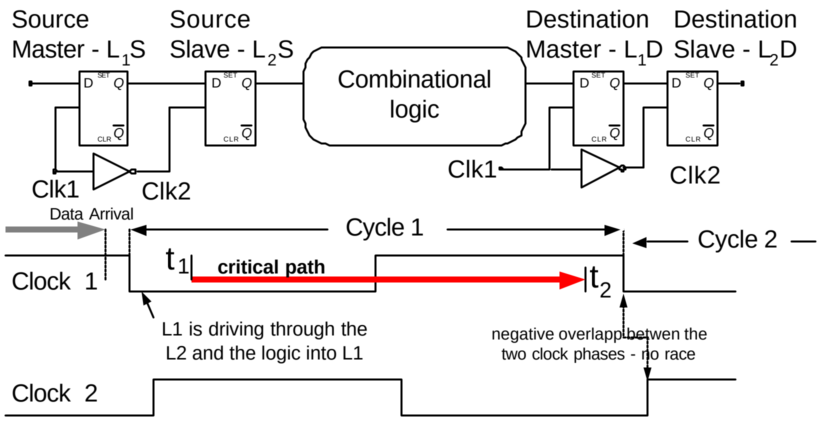


Fig. 46. Master-Slave (L_1-L_2 latch) with non-overlapping clocks C_1 and C_2 obtained by locally generating clock C_2. This is also mistakenly referred to as *Flip-Flop*.

High-performance systems are designed with the maximal performance being objective. Therefore, overlapping of the clocks C_1 and C_2 is commonly employed, thus leading to the "*critical race*" analysis (again, reader is referred to the Unger and Tang's paper). The analysis leads to the constrain of the minimal signal delay in the logic D_{LmB} in order to prevent the critical race:

$$D_{Lm} > D_{LmB} = V + H_1 + T_{1T} + T_{2L} - D_{2CQm} \quad (32)$$

What equation (32) tells us is that any amount of time we have added to the upper bound of the *critical path*, thus allowing more time in the logic, will have to be added to the minimal bound for the *short paths*, thus increasing the constraint on the short path. This may force us to add some *padding* to the short paths (insert inverters in order to increase the delay) in order to meet the constraint (32).

We may be interested to know what is the maximal amount of overlap *V* that we can utilize. This can be obtain solving the timing equations (29,31) [Unger-Tan] leading to equation (33):

$$V = T_{1T} + T_{2L} + D_{2CQM} + U_1 - D_{1DQM} - D_{2DQM} \quad (31)$$

In summary, when using a two-phase clock with *Master-Slave* (L_1-L_2) latches a conservative design would eliminate the need for the analysis of *fast paths* (critical race condition). This is achieved by using non-overlapping clocks C_1 and C_2. However, this is done at the expense of the performance. When maximal performance is the objective, it is possible to adjust the clock overlap *V* by phasing the clocks C_1 and C_2 so that the system runs at the maximal possible frequency. The maximal clock frequency is achieved when P_{min} is equal to the sum of the delays incurred when traversing the path consisting of the maximal delay in the logic and delays in the latches L_1 and L_2.

Example Alpha-2 clocking

Let us consider an example consisting of the optimal clock parameters for the single latch clocking as used in the second-generation "alpha" processor. For more detail explanation one should read [Alpha-2 paper]. For this analysis and we will use notations from [S. Unger and C. J. Tan].

Let is assume the following parameters of the system:

Clock skew: T_L = T_T = 20ps, for both edges of the clock. Latch L_1 parameters are: clock to Q delay D_{CQM} = 50ps, D_{CQm}= 30ps, D to Q delay D_{DQM} = 60ps, setup time U = 20ps, hold time H = 30ps. Latch L_2 parameters are: D_{CQM} = 60ps, D_{CQm}= 40ps, D_{DQM} = 70ps, U = 30ps, H = 40ps. The structure of L_1 and L_2 latches used in the second generation of "Alpha" processor are shown in Fig. .

The critical paths in the logic sections 1 and 2 are: D_{L1M}=200pS and D_{L2M}=170ps.

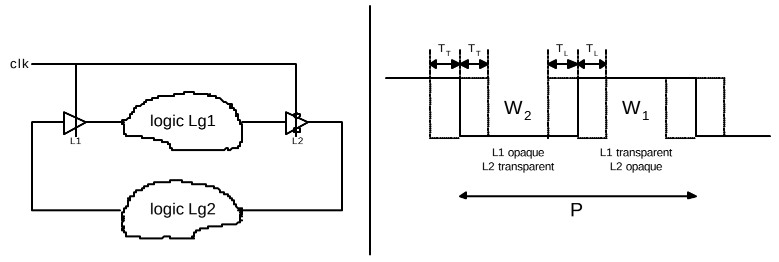


Fig. 47. Timing arrangement used in the second generation of "Alpha" processor [Alpha-2]

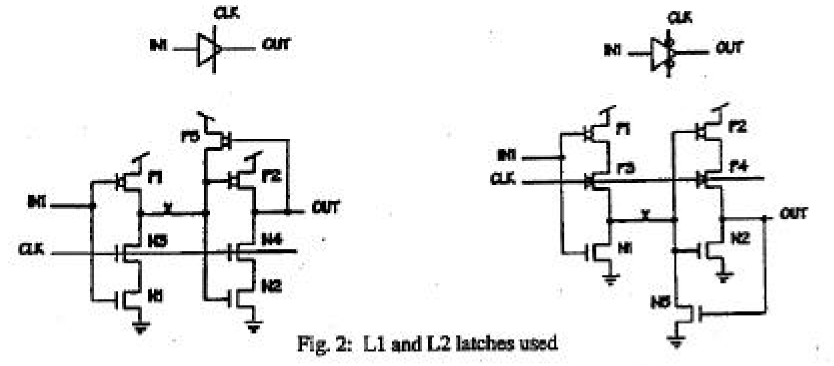


Fig. 48. Latches used in the second generation of "Alpha" processor [Alpha-2]

<u>Solution:</u>

For the given clock setup: V=0 and clearly $P=W_1+W_2$.

With the nominal time, t=0 set at the leading edge of the clock we obtain:

$$t_{D1LArr} \le V - T_T - U_1 = -T_T - U_1 \tag{32}$$

$$t_{D2LArr} \le W - T_T - U_2 \tag{33}$$

In addition we have:

$$t_{D2,LArr} = \max\{t_{D1LArr} + D_{DQM,1}, t_{C1LL} + D_{CQM,1}\} + D_{L1M} \tag{34}$$

where:

$$t_{1CLL} = V - W_1 + T_L \tag{35}$$

Substitution of equations (32) and (33) into (34) yields:

$$W_2 \geq V + U_2 - U_1 + D_{1DQM} + T_L - T_T + D_{L1M} \tag{36}$$

$$P = W_1 + W_2 \geq V + U_2 + D_{1CQM} + 2T_L + D_{L1M} \tag{37}$$

Due to the symmetry of the clocking scheme, moving the reference point from the leading edge of the clock to the trailing edge of the clock will give us the same equations with indexes interchanged:

$$W_1 \geq V + U_1 - U_2 + D_{2DQM} + T_L - T_T + D_{L2M} \tag{38}$$

$$P = W_1 + W_2 \geq V + U_1 + D_{2CQM} + 2T_L + D_{L2M} \tag{39}$$

Substituting the values into equations: (36-39) we obtain:

$$W_2 \geq 290pS$$

$$P \geq 320\,pS$$

$$W_1 \geq 230pS$$

$$P \geq 290\,pS$$

Given that $P=W_1+W_2$ we obtain for P:

$P_{min}=520pS$

Thus the maximal frequency at which this system can run is $f_{max}=1.92GHz$.

Level-Sensitive Scan Design (LSSD)

Finally, it is important to address testability issues as they are closely related to the latch design and choice of a clock storage element to be used in the system. LSSD is a design methodology developed at IBM Corp. and used systematically in all IBM designs. The origins of LSSD can be traced to the IBM System /360 models and NEC 2200/ model 700, though LSSD was fully implemented for the first time on IBM System /38 [Ref].

LSSD is one solution to the problem of test and test generation for digital systems. The basic idea of LSSD is to convert a sequential network into a combinational network by logically cutting the feedback loops. This logical dissection is performed by converting all storage elements in the Huffman Sequential Network

Model (Fig.3.) into shift register latches and connecting them into one or more shift registers as shown in Fig.40. At this point it is possible to put the logic network into any desired state by shifting-in the proper values into the *Shift Register Latches* (SRL). It is also possible to scan out any response. Thus, for testing purposes, the network appears like a combinational network, which facilitates test generation greatly.

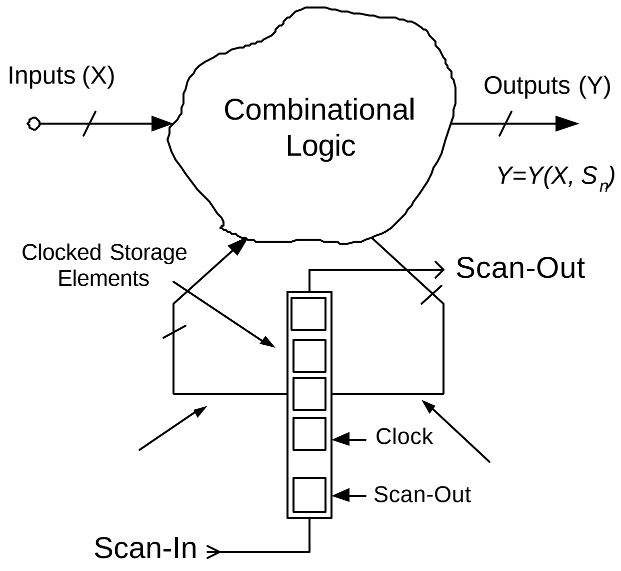


Fig. 49. General LSSD Configuration

There are two aspects of LSSD methodology as they impact timing and clock. The first attribute is a requirement for the system to be *Level Sensitive*, and the second one is requirement for *Scan Design*.

Level Sensitivity is defined in the requirements for the latch design. The latches used are assumed to be reacting to logic voltage levels and not to be affected by transition time. This is consistent with our definition of a *latch*, in this book, as opposed to a *Flip-Flop*. Further, clocks are recommended to be non-overlapping during system operation and are never overlapping during testing. Hence the network is immune to fast paths.

The requirement for *Scan Design* is spelled in the requirement that the latches used consist of Shift Register Latches (SRL), which are interconnected into one or more shift register chains. Thus, the key capability of *Scan Design* is the capability to completely control and observe all latches used in the system.

These two features are essential in making a sequential network appear like a combinational network. LSSD makes it possible to scan-in, as well as scan-out, values into and from all the latches in the system.

The advantages of LSSD are summarized as:

1. System performance is independent of time dependent characteristics of the signals, such as like rise and fall time.
2. As far as test generation is concerned, all the logic networks are treated as combinational, thus greatly simplifying testing and test generation process.
3. Ability to scan eases debugging of designs.
4. Ability to scan simplifies machine "bring-up".
5. Design verification is simplified.
6. In the case where complete systems are designed using LSSD, the same manufacturing tests can be applied to diagnosis of faults on the customer's site.

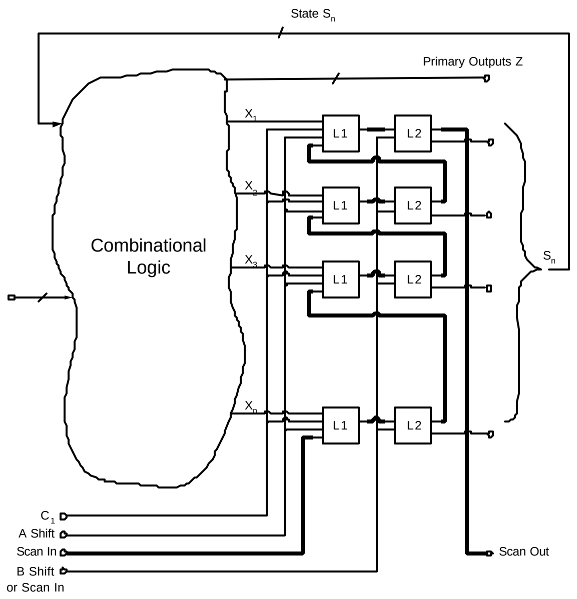


Fig. 50. LSSD Double Latch Design

There are two basic ways to design logic in LSSD. One is by using a single latch, other by using double latch design (as described in this chapter). Double Latch Design is also known as Master-Slave or Latch-Trigger design. In Double

Latch Design, shown in Fig. 41, outputs are taken from the L_2 latches. Since the L_1 and L_2 latches must have separate clocks, this design is inherently level sensitive. Double Latch Design requires no more than two system Clocks C_1 and C_2 and two shift clocks A and B. C_2 clock for the L_2 latch behaves like a Shift B clock during testing and a system clock C_2 during normal operation. It is not necessary to use two separat e clocks C_2 and B since the function can be shared during the normal operation and testing.

LSSD is a concept that can be applied to a complete system design from the module or a card to a chip.

Shift-Register-Latch is defined as a combination of two latches: *Data Input Latch*, L_1 and a second latch, L_2, which is used in normal, or shift register operation. Latch L_1 my be fed by one or more system clocks, data inputs, set inputs, reset inputs, scan data inputs and shift-A clock inputs. Latch L_2 may be fed only by latch L_1 and shift-B clock inputs.

System data outputs may be taken from Latch L_1, from Latch L_2 or from both L_1 and L_2. At least one output from L_2 must be used to provide a shift register data path.

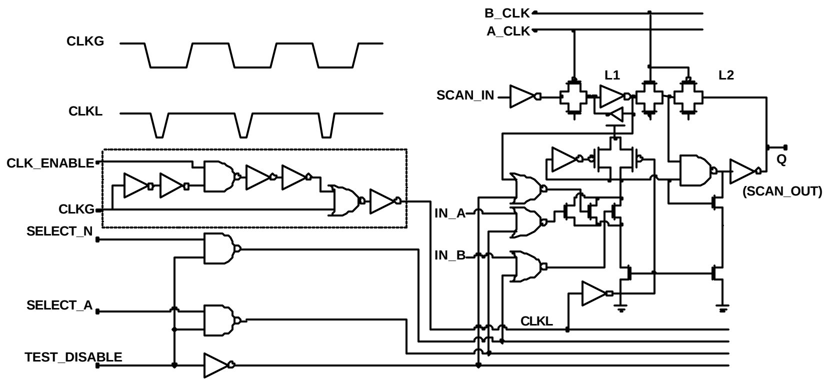


Fig. 51. LSSD SRL with multiplexer used in IBM S/390 processor

A CMOS implementation of LSSD SRL with multiplexer at the input, used in IBM S/390 microprocessor is shown in Fig. 42.

References

[1] I. A. Young, J. K. Greason, and K. L. Wong, "A PLL clock generator with 5 to 10 MHz of lock range for microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 1599 - 1607, November 1992.

[2] F. Gardner, *Phase Lock Techniques*. John Wiley and Sons, New York, 1979.

[3] V. von Kaenel, D. Aebischer, R. van Dongen, and C. Piguet, "A 600MHz CMOS PLL microprocessor clock generator with a 1.2GHz VCO," *IEEE International Solid-State Circuits Conference*, vol. XLI, pp. 396 - 397, February 1998.

[4] B. Kim, T. Weigandt, and P. Gray, "PLL/DLL System Noise Analysis for Low Jitter Clock Synthesizer Design," *Proceedings of the 1994 International Symposium on Circuits and Systems*, Vol. 4, 1994, pp. 31-38.

[5] B. Razavi (ed.), *Monolithic Phase-Locked Loops and Clock Recovery Circuits – Theory and Design*, IEEE, New York, 1996.

[6] S. Sidiropoulos, D. Liu, J. Kim, G. Wei, M. Horowitz, "Adaptive Bandwidth DLLs and PLLs using Regulated Supply CMOS Buffers," *IEEE Symposium on VLSI Ciruits*, pages 124-127, June 2000.

[7] K. Lim, C. Park, D. Kim, and B. Kim, "A low-noise phase-locked loop design by loop bandwidth optimization," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 807 - 815, June 2000.

[8] S. Sidiropoulos and M. A. Horowitz, "A semidigital dual delay-locked loop," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1683 - 1692, November 1997.

[9] S. Rusu and S. Tam, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE International Solid-State Circuits Conference*, vol. XLIII, pp. 176 - 177, February 2000.

[10] T. Xanthopoulos, D.W. Bailey, A.K. Gangwar, M.K. Gowan, A.K. Jain, B.K. Prewitt, "The design and analysis of the clock distribution network for a 1.2 GHz Alpha microprocessor," *IEEE International Solid-State Circuits Conference*, pp. 402–403, February 2001.