An Insight into Sequential Logic Circuits

By

P.Radhakrishnan,
Senior ASIC-Core Development Engineer,
Toshiba,
1060, Rincon Circle,
San Jose, CA 95132 (USA)

July 1999 (Issue-1)

Contents

Acknowledgments

I owe an enormous amount of gratitude to the teaching community. I wish to thank my maths teacher Mr.Balaji Singh, my favourite teacher Mrs.Bharathi Dharmaraj and the staff members of GCE, Tirunelveli where I learnt the fundamentals of what I'm doing right now in the industry. My appreciations to Dr.V.Palanisamy and Prof.Krishnamachary, who took us into a different dimension of technical learning during college days. My acknowledgments to them and my sincere thanks to the efforts of Dr.L.Ganesan and to Prof.M.Chellaiah for the guidelines they provided to me. Thanks to my parents for providing the best education to me, amidst their miseries, which made me who I am today. My sincere thanks to my uncle Mr.R.Shridhar from whom, I learnt a lot about electronics through fascinating small and big projects. Special thanks to my wife and my friends who helped in evaluating this paper and gave valuable feedback. Also I wish to thank my manager Mr.S.Krishnamoorthy for his valuable feedback. I dedicate this effort to my little son.

Introduction

It is true that every aspect of the professionalism is not gained in the schools. But we need to agree that the effort one takes to understand the concepts is very important for him/her during the initial stages of the professional life, which makes him/her an expert later. In this field of electronics engineering, where technology changes faster than we think, a constant knowledge seeking has become an essential factor to survive. A good understanding of the fundamentals will always make this process, an easy endeavour.

The intention of writing this white paper is to bring some of the industrial happenings into the educational environment so as to increase the awareness of the students and to better prepare him/her for the challenges he/she is going to face during the professional life. My plan is to write about some topics in which I have gained expertise. Before we go into some of the topics in the ASIC design and related fields, I wish to spend some time in discussing the basics of digital logic design. This paper discusses in depth about designing and analyzing the synchronous sequential logic circuits. Since I am an ASIC designer, many times I tend to assume the chip design scenarios. However the design principles are the same for any kind of sequential logic.

Combinational and Sequential

Eventhough this paper is about the sequential circuits, its appropriate to spend some time to talk about the combinational circuits because the combinational logic is a part of the sequential logic most of the time. It is well known that logic decisions can be made with the basic gates like AND, OR and the NOT. These logic gates generate an output depending on the current value at the inputs. We can easily imagine a combinational cluster in which there can be many logic gates of many types connected together to achieve a desired logic function. You may already know how to build combinational logic using the Boolean equations. A combinational logic can be defined as follows.

> *A cluster of logic gates is said to be combinational if it generates a set of outputs for a set of defined inputs and the outputs change whenever there is a change in the inputs. For a unique combination of inputs there will be a unique output combination. These circuits are asynchronous in nature.*

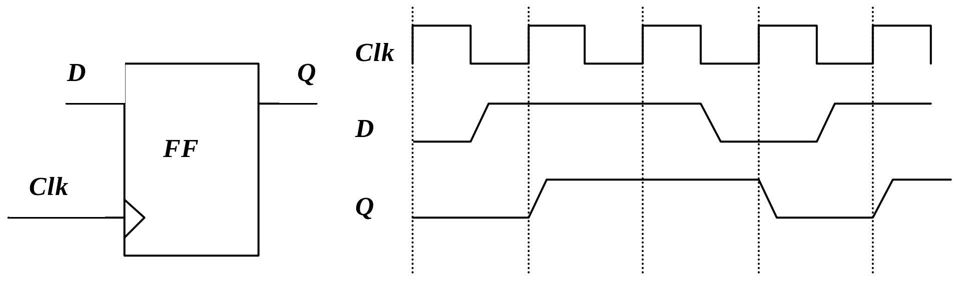
Now lets try to capture what a sequential circuit is. What comes to your mind when I say sequential circuit? I am sure you will be thinking of the flip-flops and the latches. The essential nature of the sequential logic is that it is capable of storing a set of information. This is obvious from the fact that the sequntial circuits use flip-flops (lets not worry about the latches). And you all know that a flip-flop is a one bit memory. We can define a sequential circuit as follows.

> *A sequential circuit is one, which generates a set of outputs for a set of inputs and the output changes from one state to the next state when the qualifying event happens.*

I know that this definition is quite different from what you learn from most of the texts. For time being just take my words and we will examine this while we go through the rest of the material.

Basic Flip-Flop

A flip-flop is a storing element, which can remember one bit that has been provided in its input. A basic flip-flop has a clock source and during the positive transition (zero to one) of the clock the flip-flop captures the value available in the input pin. Lets assume a D-type flop (a flip-flop is commonly called as "flop") with a rising clock throughout our discussion. As long as the input at pin D of the flop is stable when the clock rises, the flop will register the input and make it available at its output pin Q. We call this process as "*registering*" the input. Hence every time the flop hits a rising edge of the clock, the input gets registered. Now you can imagine what happens if the input at D changes between two rising edges of the clock. The change at the input D will be reflected at the Q pin of the flop just after the next clock edge. The following diagram shows the flop and the associated timing diagram.



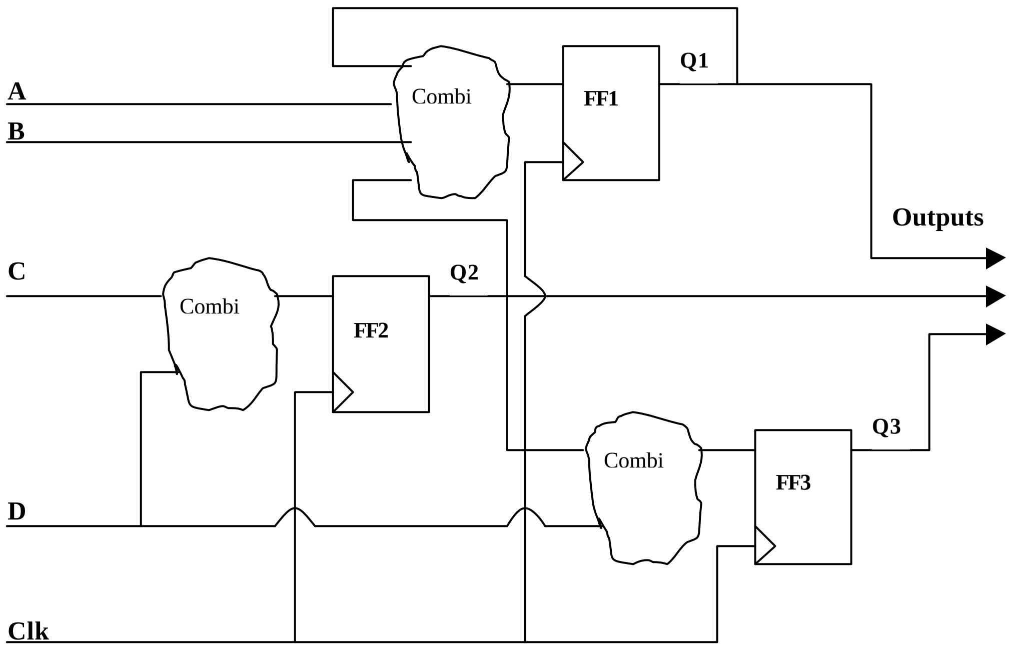
From the above figure you can see that the flop updates its output while encountering the positive edge of the clock. This is the qualifying event for the flop to do the registering of the input. Now imagine a set of flops connected to a set of inputs and operated by the same clock. These flops will generate outputs at every active clock edge (i.e. the rising edge) and the output of each flop simply depends on what is available at its input at the time of encountering the active edge of the clock.

Sequential logic

There are two types of sequential circuits. They are, synchronous sequential circuits and asynchronous sequential circuits. In a synchronous sequential circuit the sequential elements (the flip-flops) use a basic clock for the state transition. The active edge of the clock acts as the qualifying event to capture the input value to the output. The most commonly used sequential element is the D flip-flop with clock. However there are sequential circuits, which can function asynchronously also. These circuits change their state whenever there is a change in (atleast) one of its inputs. These logics also remember a set of values. These types of circuits use flip-flops with asynchronous "set" and "reset" inputs or "delay units' as the sequential elements. The RS latch with two cross-coupled NAND gates, which you can see in most of the texts, is an example for an asynchronous sequential circuit.

I wish to limit the scope of this paper to the synchronous sequential circuits to get a better understanding of the timing aspect of the circuits. We are going to discuss about the timing aspect in the rest of the paper. So, now onwards in this paper if I say sequential logic, it means the "synchronous sequential logic".

In a sequential circuit there can be many flops which may get input from various other flops. It may get its own output into the input. This is a very common scenario in a sequential logic circuit. Hence the output of the sequential logic at any qualifying event (i.e. the active edge of the clock for a synchronous sequential logic) will depend on the external input and also on the current output. Now you must have figured out that the sequential logic has combinational logic also as a part of it. This is depicted in the following figure.



This figure is a conceptual representation of any sequential logic. In a fairly complex circuit the number of flops and the number of combinational clusters will be much more than what you see here. But this representation is good enough to understand the operation of the sequential logic. There are three flops in this logic, generating three bits as the output. There are four primary inputs to the circuit and they are named as A, B, C and D. These are called "primary inputs" because these inputs arrive from the external world into the sequential circuit. By the same definition, we can call the three bits of outputs as the "primary outputs". All the three flops in the circuit are triggered by the same clock, and all of them are positive edge triggered flops. Every time a flop encounters a positive edge of the clock, the value at its D pin is captured into it and reflected in the Q pin. As you can see, the value at the input pin D of the flop is the output of the combinational cluster, which in turn depends on a set of the primary inputs and also some of the current outputs of the flops. Hence immediately after the clock edge, the value of the primary outputs (we will call this as the next state) depends on the current value at the primary output and the current primary inputs. Now we shall define a synchronous sequential circuit in a better way to express what we understood.

A sequential circuit stores a set of value as its outputs, and its next state of the output depends on a set of inputs (primary inputs) and also on the current state of its output. The state change of this type of circuits takes place synchronous to a clock.

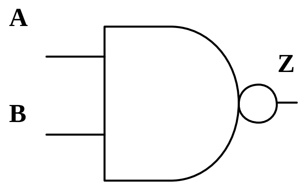
I guess this definition is the same as what you learn in many texts!

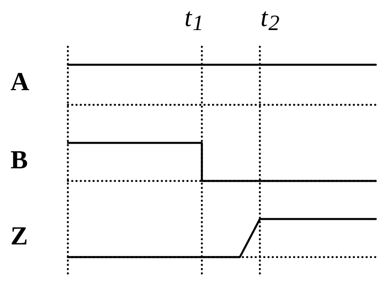
Timing aspect

A sequential logic is said to be functioning correct if it meets both the functional and timing goals. Let us examine what these functional and timing goals are. The functional aspect is very easy to understand. The logic is functionally correct if it generates a set of expected outputs while applying a defined input. This simply means that, for e.g. a two input EX-OR gate generates a HIGH output when only one of its inputs is HIGH. For the other two cases it generates a LOW output. If this is verified, the gate is said to be functionally working correct. But in a fairly complex sequential circuit there may be thousands of flops being fed from combinatorial (this is another word for combinational) clusters having hundreds of gates. In the industry, people use various simulation tools to verify the functional correctness of a large sequential circuit. These tools need test vectors to verify the logic. If all the possible input sequences are verified and if we get the correct response at the outputs, we can say that the circuit is functionally correct.

Now let me introduce the timing aspect of a logic circuit. Any logic function, when implemented in silicon using a set of transistors, becomes a physical device. Lets assume you have a NAND gate with two inputs. In silicon this can be assembled with four CMOS transistors. Whenever one of the inputs of this gate becomes "*zero*" the output turns to "*one*". ("*zero*" is the dominant input for a NAND gate). Assume that both the inputs of this gate are in state "*one*" initially. At time $t=t_1$ one of the inputs is changed to "*zero*". Now the output changes to "*one*". But this does not happen instantaneously. The logic takes a definite amount of time to change from state "*zero*" to state "*one*". Assume that at time $t=t_2$, the output turns to "*one*". Hence the gate has taken t_2-t_1 time to respond to the change in the input. This delay is known as the "intrinsic delay" of the gate or more commonly known as the "propagation delay". This is shown in the following figure.

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0



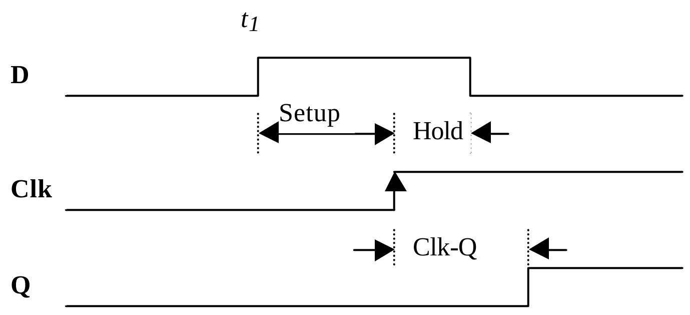


This intrinsic delay is applicable to sequential elements also. These are the numbers you see when you open a data sheet of a flip-flop. For a flip-flop the intrinsic delay is measured in the following manner. The time difference from the time at which the active clock edge happens to the time at which a stable output is available at the Q pin of the flop, is the intrinsic delay. Often times this number is referred as the "clock to Q delay" of the flop. There are other timing parameters associated with a sequential element and we will touch upon that real soon.

What happens if the input D of the flop is unstable while it hits the active edge of the clock? It is hard to register an unstable value, and the output becomes unpredictable. There is a technical term for this, which you might have heard. The flop enters a metastable state where the output is unpredictable. By this it is evident that for a successful registering of a signal into a flop, the input has to be stable while the capturing happens during the rising edge of the clock. This means that there is a window around the clock edge where the input signal has to stable to be registered by the flop. By stable I mean the input has to be a "*one*" or a "*zero*". This window is defined by the terms "setup time" and "hold time".

Setup time is the minimum time for which the data input must be stable before the clock transition.

Hold time is the minimum time for which the data input must be stable after the clock transition.



In the above diagram the input is changed from "*zero*" to "*one*" at time t1. The flop-flop captures the input on the rising edge of the clock. As you can see, the input gets stable before the rising edge to allow setup time. Similarly the input is held at "*one*" after the capturing clock edge. This is to allow sufficient hold time.

To conclude this section of the paper, let me say that in addition to the correct logic functionality, a sequential circuit has to meet its timing requirements also to operate correctly. The following sections discuss about this topic in a greater depth.

Binary Counter

Having known about the setup/hold time requirements and the "Clock to Q" delay of a flop, let us discuss about the simple counter. For this we shall assume a binary counter with three bits which runs from 000b (binary) to 111b (binary) and folds back to 000b. Let us have a control signal called "en" to control the counting. When "en" is a "*zero*", the counter resets to 000b. When "en" is "*one*" the counter runs and keeps incrementing. We can represent this circuit by the following truth table. The equations representing the next state of each of the bits in the counter are given along with truth table in the figure below. I encourage you to derive these equations using one of the Boolean reduction techniques you already learnt.

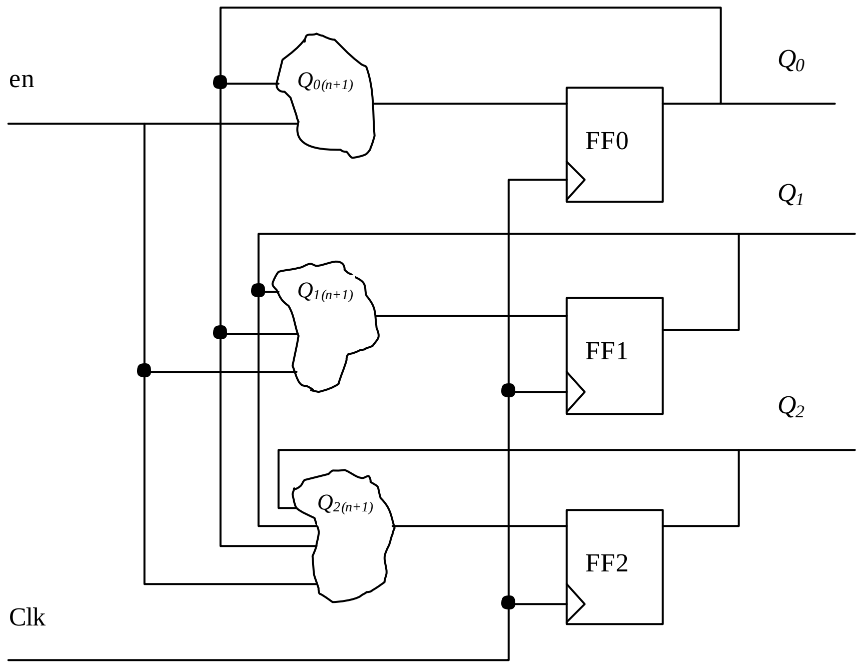
en	Current State(n)			Next State(n+1)		
	$Q_{2(n)}$	$Q_{1(n)}$	$Q_{0(n)}$	$Q_{2(n+1)}$	$Q_{1(n+1)}$	$Q_{0(n+1)}$
0	*x*	*x*	*x*	*0*	*0*	*0*
1	*0*	*0*	*0*	*0*	*0*	*1*
1	*0*	*0*	*1*	*0*	*1*	*0*
1	*0*	*1*	*0*	*0*	*1*	*1*
1	*0*	*1*	*1*	*1*	*0*	*0*
1	*1*	*0*	*0*	*1*	*0*	*1*
1	*1*	*0*	*1*	*1*	*1*	*0*
1	*1*	*1*	*0*	*1*	*1*	*1*
1	*1*	*1*	*1*	*0*	*0*	*0*

$$Q_{0(n+1)} = en.Q_{0\ (n)}^{'}$$

$$Q_{1(n+1)} = en.[Q_{0(n)} \wedge Q_{1(n)}]$$

$$Q_{2(n+1)} = en\{Q_{2(n)}[Q_{0\ (n)}^{'} + Q_{1\ (n)}^{'}] + Q_{2\ (n)}^{'}\ Q_{1(n)}\ Q_{0(n)}\}$$

These equations represent the next states of each of the bits of the 3-bit binary counter. There is an external input in this circuit viz. “en”. This input controls the operation of the counter. The three bits of the counter are the primary outputs in this case. Each of the bits is represented by the combinatorial equation in which the terms are made up of some combinations of the primary input and Q bits of the counter. These Q bits of the counter are the feedback inputs to generate the next state outputs. Thus at every clock edge the next state of the counter depends on the primary input “en” and the current state of the counter. A schematic representation of this sequential circuit as shown in the figure below. Here I have the three flip-flops, which function as the storing elements. There is a combinatorial cloud (cluster) for every flop, the output of which is connected to the D input of the flops. Each of the combinatorial cloud generates the output of the respective terms indicated by the equation, thus satisfying the next state generation. With this arrangement, the sequential circuit will keep counting from 000b to 111b as long as the “en” pin is held “HIGH”.



What should be the frequency of the clock for this logic to function with out any problem? To answer this question let us look at the same diagram above. It is known that each of the combinatorial cluster will take a definite amount of time to generate its output. This time depends on the number of levels of logic in the cluster. This in turn depends on the number of

Sum of Product (SOP) terms that appear in the equation. It also depends on the type of logic gates you have with you (e.g. two input or three input gates) to realise the equation. The larger the number of levels, larger will be the delay to get the output from the logic. Now, if the clock period is less than the time needed to generate the combinatorial output, the flop will not capture the correct value. So the clock period should be greater than the time needed to generate the combinatorial output. To be more specific, there are three combinatorial equations in this logic. Each of the clouds, which generate the combinatorial output may take different amount of time depending on the complexity. Hence to capture the correct value of the next state, the period of the clock should be higher than the largest of the three equation's generation time. With these information can you now find out the maximum frequency at which this sequential circuit can be operated?

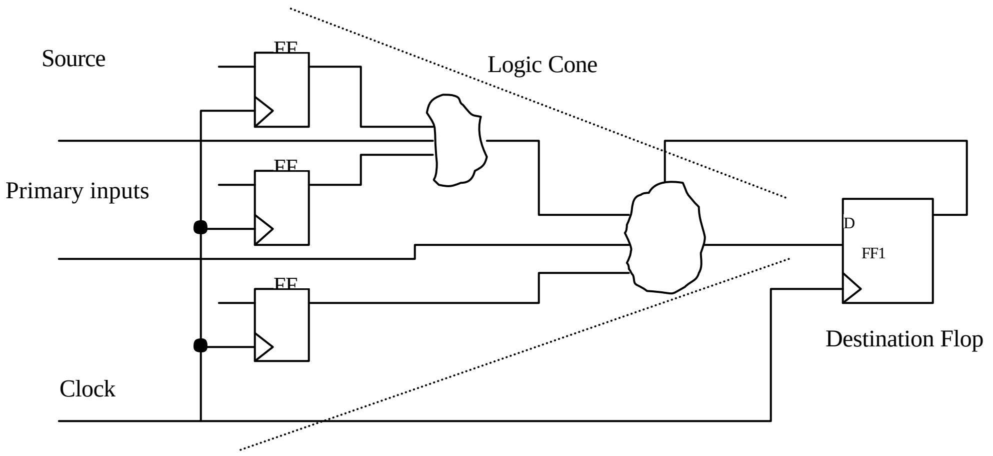
Static Timing Analysis

So far we have been talking about sequential circuits which had only a few flops. But in real life in an IC like a processor or a communication controller, there may be millions of flops connected in a cluster to generate various functions of the chip. If we assume for now that all these flops are triggered using the same clock (this is the nature of a synchronous sequential logic), each of these million flops will be registering the value in its input at every active edge of the clock. Eventhough I have mentioned about the functional aspect and the timing aspect of a circuit, lets concentrate on the timing aspect in this section. The functional aspect is something that has to be verified with all possible combinations of the input sequences. Let us assume that the functional aspect of the circuit is already verified and found correct.

For each of these flops to capture the D input correctly, it has to simply meet the two requirements. i.e. the flop should have enough setup time and sufficient hold time with respect to the clock edge used for registering. If one can verify all the flops in the circuit against the setup and hold time requirements, it can be said that the timing requirement for the entire circuit is met. This process of verifying the timing requirement of a sequential circuit is known as Static Timing Analysis (STA). It is called static analysis because the external inputs for the circuit are not dynamically changed unlike the functional testing using test vectors in a simulation. Here the inputs are assumed to be static (either "*one*" or "*zero*"). Also the whole analysis assumes just one clock cycle with a source edge and a capturing edge. In the following paragraphs we will learn more about the STA.

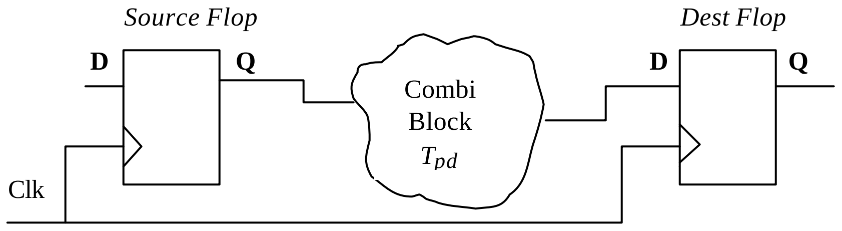
Eventhough the STA is a process of checking the setup and hold time of every flop in a sequential circuit, other factors like variation in the performance of the device and factors like the clock skew, makes STA a very involved process. We will learn about this real soon.

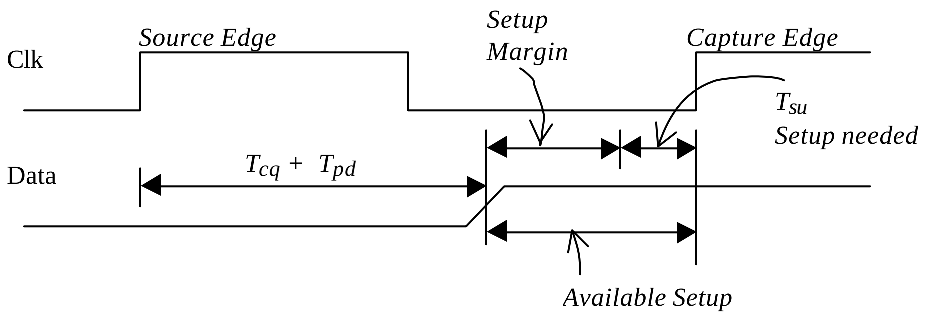
Let us consider a sequential circuit in which there are a known number of flops and the associated combinatorial clusters. In a practical circuit the combinatorial clusters will be either from the primary inputs or from the outputs of other flops or from both. For time being let us forget about the primary inputs. (I'll explain later what effect these primary inputs have on STA). So the sequential circuit can be imagined now as a set of flops, each of which has a combinatorial cluster at their inputs. For each of these combinatorial logic the inputs are from the output of other flops. (As I said do not worry about the primary inputs now). We can call these flops as the source flops. So for every flop in the design, there is one or more flops acting as the source flops. In other words for every flop (FF1) in the design there is a set of logical paths and each path has a source flop at the starting point. All these paths converge into the D input of the flop FF1 making the combinatorial cluster. These are called logic cones (shown in fig). As we discussed earlier the longest path in this cone will determine the frequency of operation of the circuit. Now that we know what a source flops is, lets call the FF1 (the flop being analysed) as the destination flop. So, any complex sequential circuit can be broken into destination flops with a logic cone having various paths with the source flops at the starting point of every path.



While doing the static timing analysis, we are concentrating on the timings of the destination flops. For a given frequency of operation for the logic, the combinatorial delays between the source flop and the destination flops must be less than the clock period for the proper functioning of the logic. Apart from the combinatorial delays there are other timing numbers one has to consider while doing this calculation. As you know already, any flop would require a setup time at its input before which the input has to be stable for a successful capturing. Let the clock period be "*T*"sec. When the source flop hits an active edge of the clock (we will call this edge as the source edge of the clock) the output will appear after the "Clock to Q" delay "T_{cq}" of the source flop. Once the output appears in the Q pin of the source flop, the combinatorial logic in the path would take a definite amount of time to generate its output. This will be the propagation delay for this logic circuit "T_{pd}". The output from the combinatorial logic is captured at the destination flop using the next edge of the clock. Hence this clock edge can be called as the capturing clock edge. From the source edge to the capturing edge we have one full clock period to accommodate the delays that happen in the path. Till now we have two delay figures in the path and they are the T_{cq} of the source flop and the T_{pd} of the combinatorial cluster. When considering the setup time (T_{su}) of the destination flop, we can say that the following relation has to be satisfied for the destination flop to capture the logic value correctly.

$$T_{cq} + T_{pd} + T_{su} = T$$

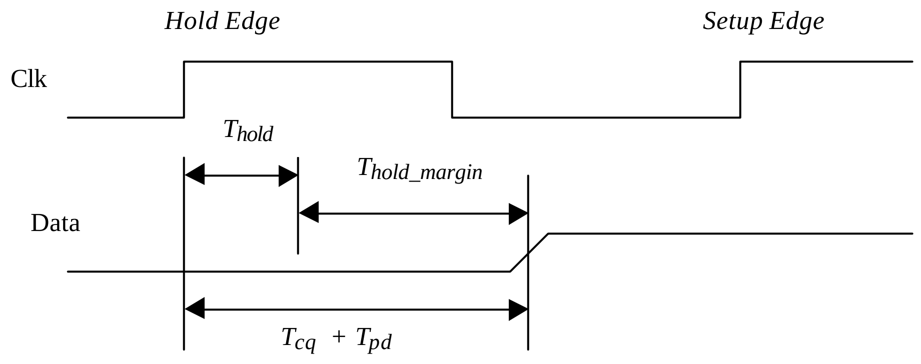




Let us modify the previous equation slightly to accommodate the term T_{su_margin}. This number represents the available margin, which can be used by the combinatorial logic without risking the capturing at the destination flop. When this number becomes zero it would mean that the logic is just meeting the setup requirement. Hence,

$$T_{cq} + T_{pd} + T_{su} + T_{su\_margin} = T$$

Now let us give some thought about the hold time requirement of a flip-flop. As we already defined, the input at the destination flop has to be held at the same value for at least equal to the hold time requirment of the flop. The hold check is done at one clock edge before the edge at which the setup check is done. If the circuit meets the hold check at this edge then for every clock edge, the hold time will be met (Sometimes the hold check is done at other clock edges also due to various reasons. We will see this at a later stage).



We can write a similar equation for the hold time relation also. It is given below.

$$T_{cq} + T_{pd} = T_{hold}$$

Or, $$T_{cq} + T_{pd} - T_{hold\_margin} = T_{hold}$$

Now let us take a small example and apply what we learnt so far and find out the timing margins. In the figure given below, there are two flops connected to each other using two different combinatorial paths, each of which has its own characteristic delays. Let us assume that the timing numbers T_{cq}, T_{su} and T_{hold} for these two flops are the same and equal to 1 ns. Also let us assume that the frquency of operation is 50MHz. Since there are two flops in this logic, we need to perform the setup and hold timing checks for both these flops. First let us take the FF2. For this flop, the FF1 is the source flop with a combinatorial path of 18ns. Using the equations for the setup,

$$T_{su\_margin} = T - T_{cq} - T_{pd} - T_{su}$$

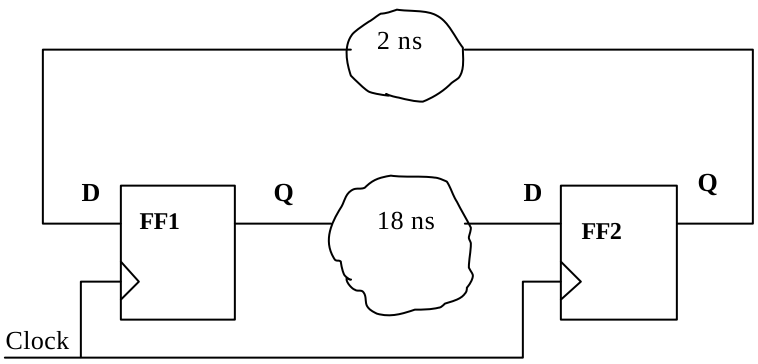
$$= 20 - 1 - 18 - 1 = 0 \text{ ns}$$

So this path barely meets the setup time requirment. The hold margin is

$$T_{hold\_margin} = T_{cq} + T_{pd} - T_{hold}$$

$$= 1 + 18 - 1 = 18 \text{ ns}$$ (This path has a large hold margin.)

We need to do the same analysis for the flop FF1. For this flop, the FF2 acts as the source flop with a combnatioanl dealy of 2 ns in the path. By the same equations we can find the setup and hold margin to be 16ns and 2ns respectively. This completes the STA for this small circuit and we found that the logic will function correctly at 50MHz.



Clock Skew and PVT Variations

So far in the discussion we were assuming that the clock is an ideal one. An ideal clock does not have any uncertainity. This means that the rising edge of the clock appears exactly at the same time all over the digital circuit. This is an impossible situation. We would use metal to connect the various flops to make the clock connections. This clock connection in an IC is called as “clock tree”. A clock tree is made up of clock buffers to balance the load on each branch of the tree, thus reducing the skew. When the flops are placed in slicon they will be spread all over the die. The clock tree connects them and provide the clock. There will be some difference in the time of arrival of the clock in the various branches of the tree. This is due to the delay encountered by the signal while getting transmitted through the connecting metal wires. It also depends on the number of flops that are connected in a branch of the tree (i.e. fanout). This clock balance itself is a vast topic and I’m not attempting to describe it in this paper. The idea here is to mention that there will be difference in time at which each of the flops get the clock signal. A flop closer to the clock source (e.g the clock pin of the chip and clock buffers) will hit the positive edge earlier than that located in farther location. This introduces an uncertainty in the clock, which has to be accounted while designing a sequential logic.

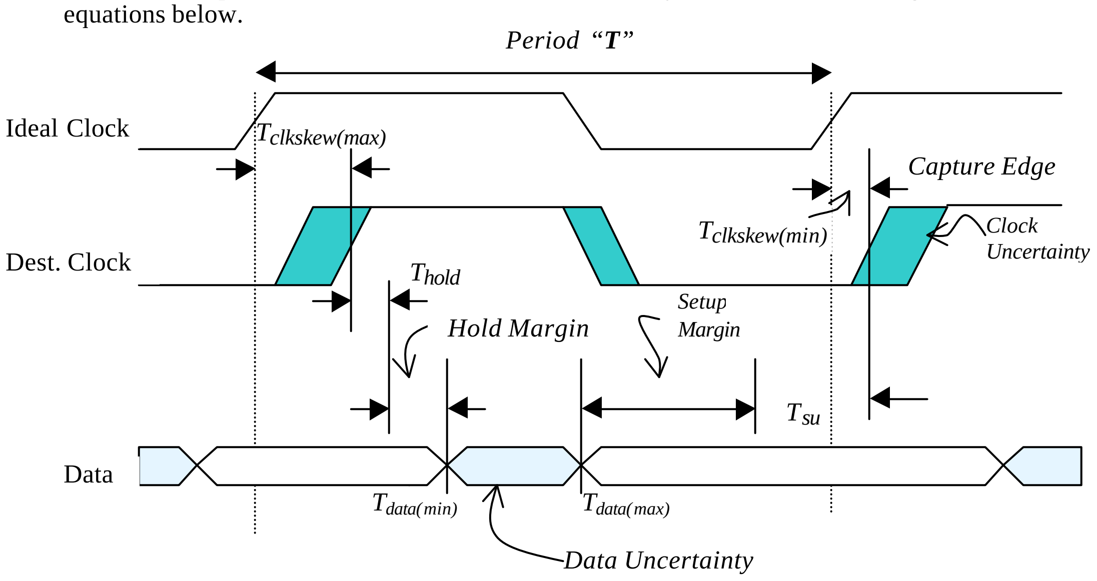
The second factor to be considered is the PVT (Process, Voltage, Temperature) variation. When a wafer is processed, the electrical characteristics of the transistors diffused at various wafers will not be exactly the same. There could be some variations in the electrical properties of the transistors, from wafer to wafer. This is known as process variation. A device is designed to function correctly when a supply voltage within a range is applied to the VCC. Most of the ICs these days have either 5Vor 3.3V as their VCC. If you look into the data sheet of a device, a range of voltage (e.g 5V±5%) will be mentioned. The device will function correctly if the supply voltage is within the range. So, the electrical parameters of the IC will vary if the supply voltage varies within the allowable range. The third factor temperature will also cause variations in the operation of the device. A device operating at 0^{o}C will have significantly different timing parameters when comparing to the one operating at 125^{o}C. The difference in the junction temperature will change the electrical characteristics.

The PVT variations are due to the physical nature of the silicon. These variations affect the timing parameters of the device, and make it to fluctuate within a range of values. When you look at the data sheet of any device, there will be a maximum value, a minimum value and a typical value for the timing parameters. These are the effect of the PVT variation we have been talking about so far. So we can say that there is worst case for a device timing in which all the PVT variations made the timing to be the worst. Also there is a best case timing parameter where the device enjoys a favourable situation like a good process, accurate supply voltage and a nice ambient temperature. For example we can take an AND gate in the data sheet and you will see three different propagation delay values max, min and typical. The max value corresponds to worst PVT and min value corresponds to best PVT. The typical value is what you will see in most of the common conditions.

Effect of Skew and PVT Variations

Hope I have slowly introduced the factors of uncertainty that can happen in a device! Could you imagine how this is going to affect the STA? The way in which we were doing the STA so far will no longer be accurate, since the effect of the clock skew and PVT variation are not taken into account. So we need to account these factors also in our calculations. Let us try to find the relation between all these variations and skew in the following section of this paper.

For this discussion let us assume a setup similar to what we had in the previous section. There are two flops, a source flop and a destination flop and the difference from the previous setup is that these two flops are far apart in the silicon. Because of the clock tree that connects these two flops, the arrival time of the clock at the clock pins of the two flops are not the same. The clocks at both the flops are skewed. This variation is represented by a min value and a max value. The min value corresponds to the earliest time of arrival of the clock at the destination flop, and the max value corresponds to the latest time at which the clock is seen at the destination flop. This is known as the "*Clock Uncertainty*" and shown in the figure and in the equations below.



The data, while traversing through the combinatorial logic will experience a delay and this delay will also have a minimum and a maximum value depending on the PVT values. This effect is shown in the diagram as the "*Data Uncertainty*". Data can become valid at any time after this uncertainty region.

Now if we wish to find out the setup margin available, we need to pick the correct edges of the clock and the correct arrival time of the data. If you consider the earliest capturing clock edge and the latest time when data can change, that will indicate the worst timing for the setup. If there is enough setup for the destination flop for this situation, then the flop will meet the setup requirement for any other timing variations.

Similarly, to calculate the hold margin, we need to consider latest edge of the clock and the earliest time at which the data changes. This will give the worst scenario for the hold timing. If the logic meets the hold requirement for this situation, it will meet the same for other PVT conditions also.

Now the relations to determine the setup margin and the hold margin can be re-written as follows

$$T_{su\_margin} = [T + T_{clkskew(min)}] - T_{data(max)} - T_{su}$$

$$T_{hold\_margin} = T_{data(min)} - T_{clkskew(max)} - T_{hold}$$

Where,

$T_{data(max)} = T_{cq(max)} + T_{pd(max)}$ This is the maximum delay possible in the data path.

$T_{data(min)} = T_{cq(min)} + T_{pd(min)}$ This is the minimum delay possible in the data path.

$T_{clkskew(max)}$ is the clock delay corresponding to the max edge.

$T_{clkskew(min)}$ is the clock delay corresponding to the min edge.

T_{su} is the setup time requirement for the flop

T_{hold} is the hold time requirement for the flop

T_{su_margin} is the setup margin available

T_{hold_margin} is the hold margin available

T is the period of the clock

In the two equations you see above, the terms representing the skew ($T_{clkskew(min)}$ and $T_{clkskew(max)}$) take a positive value if the clock edge appears after the ideal clock edge, and a negative value if the edge appears before the ideal clock edge.

With this I have taken you through describing the method of calculating the timing requirements of a sequential logic, thus enabling you to design a successful sequential logic. There is another variation that can happen in the logic circuits, which I have not mentioned so far. The effect of this also has to be considered while doing the analysis. This would further complicate the arithmetics but it is not hard to visualise. When a gate or a flop switches from one state to the other, the timings for a falling signal (switching from "*one*" to "*zero*") will not be the same as that for the rising signal. This factor also has to be considered while doing the STA and the worst of the two will determine the frequency of operation of the circuit.

On Chip Variation

In the above discussion we have talked about the variations in the electrical characteristics of the elements in a device. We have also discussed about the methods to take care of the violations occurring due to these variations. The equations we derived in the above sections consider worst case data path delay ($T_{data(max)}$) and best case clock path delay ($T_{clkskew(min)}$) for the setup calculations. For the hold calculations, these equations take the best case data path delay($T_{data(min)}$) and the worst case clock path delay($T_{clkskew(max)}$). Eventhough the above equations are correct and meet the timing requirements, they posses certain amount pessimism in them due to the following reason. When a wafer is processed, the impact of process variation (the first term in PVT) will be same allover the die. This means that most of the transistors in the die will be in close vicinity in the PVT variation curve. So the data path is assumed to be in the worst case condition, it is reasonable to assume that the clock path is also in the worst case condition. We can apply this correction in the equations for setup and hold to get a practical (more reasonable) value of margin. By this argument, we can apply all the max timings for the setup calculations and apply all the min timings for the hold calculations. By doing this, both setup and hold margins will increase and thus relaxing the timing constraint. The modern STA tools take this into consideration and apply the max clock skew for the setup calculations and the min clock skew for the hold calculations. I just wanted to convey this point and wish to leave the equations as such since the equations represent the concept behind STA. One can argue about the intra die PVT variation and consider two PVT values for two different portions of the chip. This is also a valid argument, in which case the equations will take care of this.

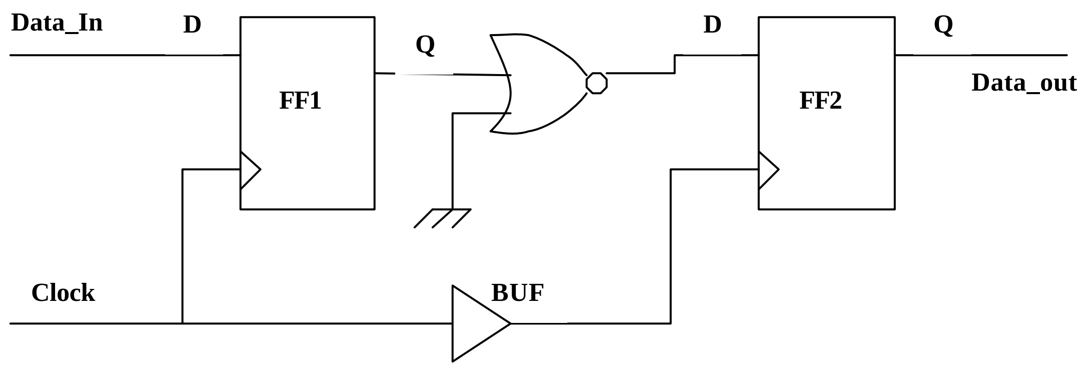
Primary Inputs

The primary inputs are the inputs arriving into the logic from external systems. For a chip, these could be the input signals from the pins. For the whole logic to function satisfactorily, these primary inputs should also meet the setup and the hold requirements. With all the above discussions, we can easily correlate the effect of the arrival time of these primary inputs. A primary input signal that arrives late will have an effect on the setup margin of the destination flop. Similarly the input that arrives early will have an effect on the hold time of the destination flop. By taking this into consideration, one can define the timing parameters of the signals at the chip level.

Example Analysis

We shall consider the following circuit and find out the maximum frequency at which the circuit can be operated. Since we are trying to find the maximum frequency, we need not allow any margin for the setup or hold timings. It is enough if the circuit just meets the setup and hold requirements. The following electrical parameters are assumed for this analysis. The setup requirement for the flop is 2 ns and that for the hold is 1ns.

Transition	D Flip-Flop (T_{cq})		NOR gate (T_{pd})		Clock BUF(T_{pd})	
	Min	Max	Min	Max	Min	Max
Low to High	4 ns	12 ns	5 ns	15 ns	3 ns	6 ns
High to Low	2 ns	10 ns	4 ns	11 ns	2 ns	8 ns



Unlike the earlier logic, there is no feedback in this diagram. The FF1 gets its input from an external circuit. We assume that this external input meets the setup and hold requirement of the FF1. So it is enough if we check the timing requirements for the FF2. Remember we need to consider two cases (rising and falling) while calculating the delays.

Consider that "D" input of FF2 is rising (changes from "zero" to "one"). So we have to take the "Low to High" delay of the NOR gate. Since one of the inputs of the NOR gate is connected to ground, for a rising signal at its output, the input must switch from "one" to "zero". This switching happens at the output of FF1. Hence we should take the "High to Low" delay number here. For the buffer in the clock path, we should take the "Low to High" delay of the BUF since we have the positive edge triggered flop. So, in this case for the data path, the min delay will be 7 ns and the max delay will be 25 ns. The earliest time the clock can be seen at the flop FF2 is 3 ns after the clock edge is seen at FF1 (this is the min delay of BUF for the "Low to High" case). This number will be used in the setup calculation. Similarly the latest time at which the clock appears on FF2 is 6 ns after that appears in the FF1. So this number has to be used for the hold time calculation. When I crunch the numbers, I get the following figure as the period.

$$T_{data(max)} + T_{su} + T_{su\_margin} = [T + T_{clkskew(min)}]$$

$$T = 25 - 3 + 2 + 0 = 24\,ns$$

$$T_{data(min)} - T_{clkskew(max)} - T_{hold\_margin} = T_{hold}$$

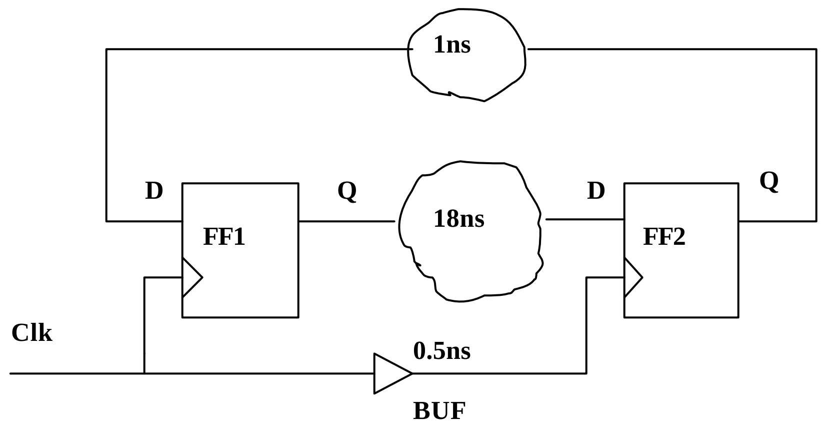
$$7 - 6 - 0 = T_{hold}$$

The equation for the hold just satisfies the requirement. So for the rising signal at the input of FF2 we have the clock period as 24 ns and the hold requirement is just met.

Now let us consider the falling signal at the input of flop FF2. By a similar method as described above, the max data delay will be 23 ns and the min data delay will be 8 ns. When we do a similar substitution we find that the clock period that satisfies the setup relation is 22ns. Also we find that for a hold margin of 1 ns the hold equation is satisfied.

Having done the analysis for both the edges now we can find out the minimum period (or the maximum frequency) with which this logic can be functional. The minimum period that can satisfy both rising and falling timings would be 24 ns. So the minimum clock period for this cicuit is 24 ns. (or the maximum frequency, f =41.67 MHz).

The following figure is the almost the same as what we saw in page-12 before. The only variation here, is in the clock tree. There is a clock buffer which introduces a 0.5 ns delay. In this calculation you may assume that the timings for the rising signal and the falling signal are the same. This is just to simplify the calculations. Take the effect of clock buffer and find out the maximum operating frequency of the circuit and check if the hold time requirement is satisfied for both the flops. Assume that the timing numbers T_{cq}, T_{su} and T_{hold} for these two flops are the same and equal to 2 ns. I leave this as an exercise for you.



Do the flops meet all the timing requirements in the above diagram? By this time you might have figured out what is to be done to fix the violations in the sequential circuits. To fix the hold violation in a path, you may introduce more delay so that the flops get more time for the hold. Such an introduction of delay will obviously worsen the setup margin because this delays the data further. Now to fix this setup issue you may have to re-arrange the combinatorial logic so as to meet the timing, and this has to be done while maintaining the logical equivalence. There are various techniques available to solve these problems. Designers use several tools to do optimal designs, which meet the timing requirements.

In the real world case, we may be asked to design a circuit to operate in a particular frequency. Most of the current applications, like the video, wireless, LAN, etc. demand a very high performance. One of the methods to achieve this, is by designing circuits with higher frequencies. It is very common to see 100MHz designs these days in the industry. With the advances in the semiconductor technology and in the design automation tools, designers meet the challenges posed by stringent requirements demanded by these type of applications.

Conclusion

There are some other factors, which are to be looked into while designing a sequential circuit. But from the basic timing point of view, what we described so far will analyse the sequential circuit to a great extent. The other factors are, the one like Multi Cycle Paths (MCP), designs using both positive and negative edged flops, other timing parameters like the width of a signal and designs which have a lot of asynchronous portion in it. There are many techniques available as standard practice (also many proprietary methods) to attack the problems faced in these types of design. I defer the discussion of these topics to another white paper.

As a summary we can say that, apart from the functional correctness of a sequential logic, it has to meet the timing requirements also for an error-free operation. Static timing analysis is the way to check if the sequential elements in a design meet the timing requirements. The effect of clock skew in the chip makes the design process a real challenge from the timing aspect.

I wish to conclude this paper with one question to you. We have discussed so much about the setup and hold timing requirements here. Can you say why there is a need for the setup or hold time for a flop?

— 0 0 —