Chapter 5
Electronic Aspects of Digital Design

This chapter is devoted to those electronics aspects important for digital circuit design. The digital devices are built with analog components, and then some considerations should be taken into account in order to obtain good and reliable designs.

Some important electronics aspects, related to the circuit design, the timing and synchronization aspects are discussed in this chapter. Most of those details are hidden in reprogrammable logic for simplicity, but these do not eliminate the consequences.

5.1 Basic Electronic Aspects of Digital Design

Digital devices represent signals by discrete bands of analog levels, rather than by a continuous range. All levels within a range represent the same signal state. Typically the number of these states is two, and they are represented by two voltage bands: one near zero volts (referenced also as ground or earth) and a higher level near the supply voltage, corresponding to the "false" ("0") and "true" ("1") values of the Boolean domain, respectively (Fig. 5.4).

5.1.1 Basic Concepts

Before we start looking at more specific ideas, we need to remember a few basic concepts.

J.-P. Deschamps et al., *Guide to FPGA Implementation of Arithmetic Functions*,
Lecture Notes in Electrical Engineering 149, DOI: 10.1007/978-94-007-2987-2_5,

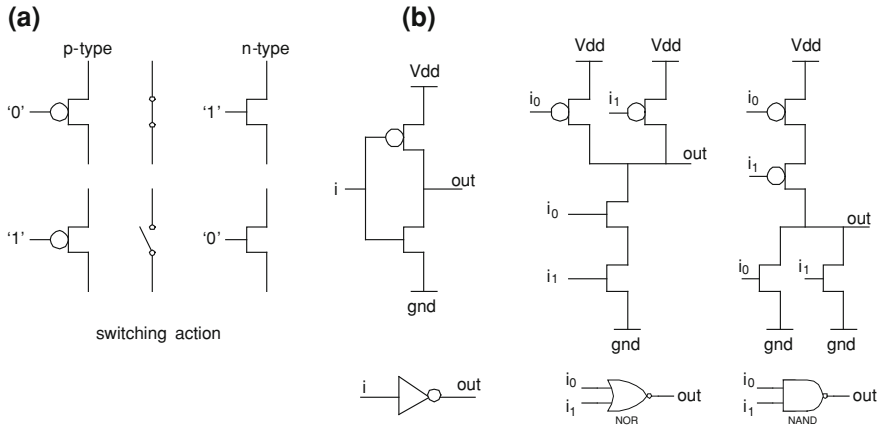


Fig. 5.1 CMOS transistors. **a** Switching action for p-type and n-type. **b** Some basic gates

5.1.1.1 CMOS Circuits

Complementary metal–oxide–semiconductor (CMOS) is the dominant technology for constructing digital integrated circuits since mid 1970s. The word "complementary" refers to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions. The key characteristics of CMOS devices are high noise immunity and low static power consumption. Neglecting other technological details, the power is only drawn while the transistors in the CMOS device are switching between on and off states. The CMOS transistors allow creating the logic gates of Fig. 1.1 that are the basics of the digital design. The key idea is the use of transistors as switches as described in Fig. 5.1a. The p transistor conducts better the supply voltage meanwhile the n-type conducts to the ground. With this, the main principle behind CMOS circuits is the use of p-type and n-type transistors to create paths to the output from either the voltage source or ground. Figure 5.1b shows the "complementary-symmetric" interconnecttion of p-type and n-type transistors to build some basic gates. More technological details of these constructions are out of the scope of this book. One can find excellent surveys and material in [1, 2].

The FPGA technology is a CMOS (re)programmable Application Specific Integrated Circuit (ASIC). Hence, all the characteristics, peculiarities and consequences of an ASIC design are present.

5.1.1.2 Fan-in and Fan-out

In ASICs technologies and others the fan-in and fan-out are measured in capacitance units (in actual technologies in *picofarad* or *femtofarad*). So, the fan-in is the

Table 5.1 Fan-in, fan-out, internal and external delay of typical gates

	fan-in (pf)	fan-out (pf)	t_int_hl (ns)	t_int_lh (ns)	t_ext_hl (ns/pf)	t_ext_lh (ns/pf)
INV	0.003	0.337	0.079	0.151	2.710	4.891
BUF	0.004	0.425	0.265	0.056	1.334	2.399
AND2	0.003	0.334	0.105	0.144	4.470	4.271
AND3	0.007	0.673	0.211	0.131	1.362	2.376
NAND2	0.004	0.197	0.105	0.144	4.470	4.271
NAND3	0.003	0.140	0.071	0.192	6.088	7.212
NOR2	0.004	0.205	0.091	0.162	3.101	8.035
XOR2	0.008	0.645	0.279	0.331	1.435	2.560
CKBUF	0.006	1.160	0.355	0.350	0.782	1.782
CKBUF_N	0.006	1.460	0.183	0.537	0.628	1.881
DFF	0.003	0.703	0.402	0.354	1.256	2.360

capacitance that an input of a gate has. Thus, the fan-out is the maximum capacitance controllable by a gate, while providing voltage levels in the guaranteed range. The fan-out really depends on the amount of electric current a gate can source or sink while driving other gates. Table 5.1 shows examples of fan-in and fan-out for gates in 0.25 μm technologies (a ten years old technology). Observe, for example, that a NAND2 gate can drive up 49 similar gates (197*pf*/4*pf*) if we neglect the capacitance of interconnection.

In the case of FPGA the concepts of fan-in and fan-out are simplified and measured in number of connections. Remember that most of the logic is implemented in look-up tables (LUTs). Then the fan-in is the number of inputs a computing block has, like a two input AND gate implemented in a LUT has a fan-in of two, and a three input NAND gate a fan-in of three. The concept of fan-out is used to express the number of gates that each gate has connected at the output. In some cases there appears the concept of maximum fan-out as the maximum number of connections that we can control by a gate.

5.1.1.3 Drive Strength or Drive Capabilities

This means the amount of power a gate or circuit can output (i.e. same concept of fan-out). The larger drive strength a circuit has, the more power it can deliver and thus the more capable it is to drive a higher number of other gates. In the particular case of ASIC libraries, a typical gate has different drive strengths (x1, x2, x4, etc.) to be able to drive more gates. This allows the designer or the synthesizer to choose the right one at each time. In FPGAs the internal drive strength is not controlled by programmers. Nevertheless, the output strength of the output buffers can be controlled.

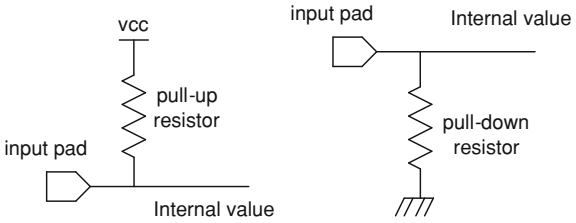


Fig. 5.2 Pull-up and pull-down resistors connected to input pads

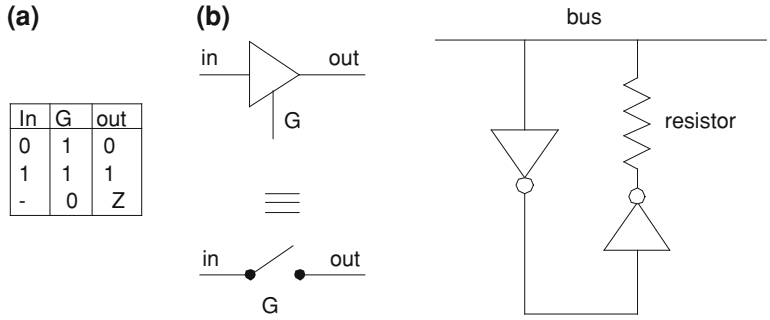


Fig. 5.3 Tri-state buffer and bus keeper

5.1.1.4 Pull-up and Pull-down Resistors

The purpose of these resistors is to force an input (or output) to a defined state. The pull-up and pull-down resistors are mainly used to avoid a circuit input from being floating if the corresponding external device is disconnected. Figure 5.2 shows a pull-up and a pull-down resistor. The pull-up resistor "weakly" pulls the internal voltage towards Vcc (logical '1') when the other components are inactive, and the pull-down resistor "weakly" pulls the internal voltage towards GND (logical '0').

Pull-up/down resistors may be discrete devices mounted on the same circuit board as the logic devices, but many microcontrollers and FGPA have internal, programmable pull-up/pull-down resistors, so that less external components are needed.

5.1.1.5 Tri-States Buffers and Bus-Keeper

Tri-state (also named as three-state or 3-state) logic allows an output port to assume a high impedance state (Z, or Hi-Z) in addition to the 0 and 1 logic levels. The tri-state buffers are used to connect multiple devices to a common bus (or line).

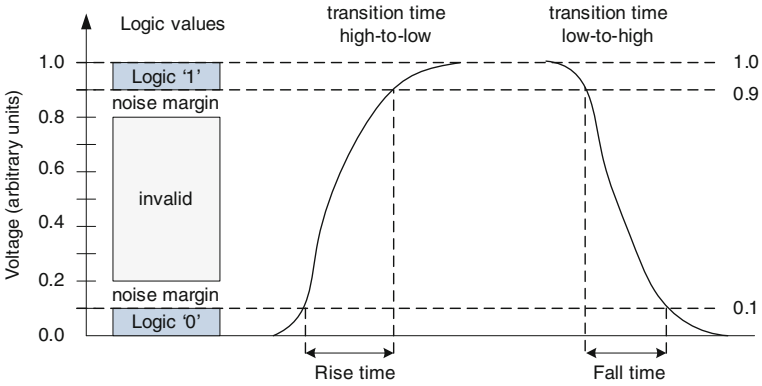


Fig. 5.4 Logic values and transition times (rise and fall time)

As is suggested in Fig. 5.3a, a tri-state buffer can be thought of as a switch. If *G* is on, the switch is closed, transmitting either 0 or 1 logic levels. If G is off, the switch is open (high impedance).

A related concept is the bus-keeper (or bus-holder). The bus-keeper is a weak latch circuit which holds the last value on a tri-state bus. The circuit is basically a delay element with the output connected back to the input through a comparatively high impedance. This is usually achieved with two inverters connected back to back. The resistor drives the bus weakly; therefore, other circuits can override the value of the bus when they are not in tri-state mode (Fig. 5.3b).

Many microcontroller and FGPA have internal, programmable tri-states buffers and the bus-keeper logic at outputs pins, so that minimal external components are needed to interface to other components.

5.1.2 Propagation Delay—Transition Time

It is the time that an electronic circuit needs to switch between two different stable states. In a logic circuit performing a change of state, it identifies the *rise-time* and the *fall-time* of the output voltage. These times are related to the times to charge and discharge capacitances. Figure 5.4 shows typical charge and discharge ramps of a CMOS gate.

5.1.2.1 Rise Time (Transition Time Low-to-High)

It refers to the time required for a signal to change from a specified low value to a specified high value. Typically, these values are 10% and 90% of the step height.

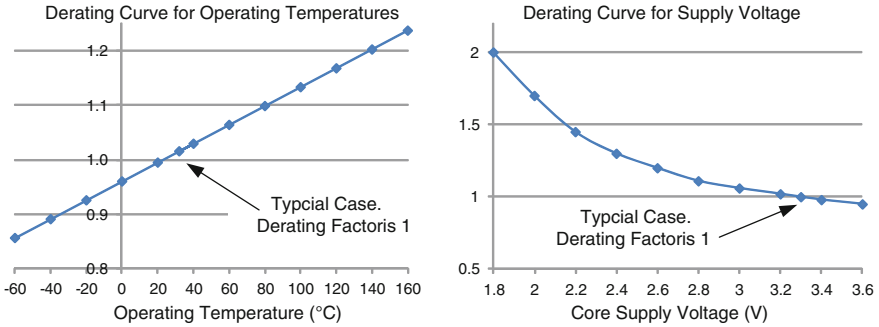


Fig. 5.5 Typical derating curves for temperature and supply voltage

5.1.2.2 Fall Time (Transition Time High-to-Low)

It is the time required for the amplitude of a pulse to decrease (fall) from a specified value, typically 90% of the peak value, to another specified value, usually 10% of the minimum value.

5.1.2.3 Slew Rate

This concept is used in linear amplifiers but also in digital circuits. In the second case, it is an approximation of the time necessary to change a logic value, without distinguishing between high-to-low and low-to-high transitions. In the FPGA arena, the output pins can be configured as slow slew rate and fast slew rate. The second one is faster, but consumes more power and is prone to transmit internal glitches to the outputs.

5.1.2.4 Propagation Delay, Intrinsic and Extrinsic Delays

The propagation delay of a gate or digital component depends on two factors, the intrinsic and the extrinsic delays. The intrinsic delay is the delay internal to the gate (also known as gate delay or internal delay). In other words, it is the time taken by the gate to produce an output after the change of an input (in ns, ps, etc.).

On the other hand, the extrinsic delay (also called, differential, load-dependent or fan-out delay) depends on the load attached to the gate. This value is expressed in ns/pf. Table 5.1 shows some intrinsic and extrinsic delays for 0.25 μm gates. The intrinsic delay in fact depends on which input is changing, but for simplicity this is not taken into account in the table. Observe that the high to low (HL) and low to high (LH) values are different. As an example, consider an inverter (INV) whose output is connected to 10 inverters. Assume that the capacitance of every

Table 5.2 A derating table for temperature and supply voltage

Vcc	Temperature (°C)								
	120	100	80	60	40	32	20	0	−20
2.6	1.39	1.36	1.32	1.27	1.24	1.20	1.19	1.15	1.12
2.8	1.29	1.25	1.22	1.18	1.14	1.11	1.10	1.07	1.03
3.0	1.23	1.20	1.17	1.12	1.09	1.06	1.05	1.02	0.99
3.2	1.18	1.15	1.12	1.08	1.05	1.02	1.01	0.98	0.95
3.3	1.16	1.13	1.10	1.06	1.03	1.00	0.99	0.96	0.93
3.4	1.14	1.11	1.08	1.04	1.01	0.98	0.97	0.94	0.91

connection is equal to 0.1 pf. What is the time required for the INV gate to propagate the transitions 0 to 1 and the 1 to 0?

$$\begin{aligned} T_{hl} &= t\_int_{hl} + t\_ext_{hl} * \text{ capacity} \\ &= 0.079\text{ns} + 2.710\text{ns/pf} * (10 * 0.003\text{pf} + 0.1\text{pf}) = 0.4313\text{ns} \end{aligned}$$

$$\begin{aligned} T_{lh} &= t\_int_{lh} + t\_ext_{lh} * \text{capacity} \\ &= 0.151\text{ns} + 4.891\text{ns/pf} * (10 * 0.003\text{pf} + 0.1\text{pf}) = 0.7868\text{ns} \end{aligned}$$

In FPGA technologies these concepts are hidden, given only a propagation delay for internal nodes (LUTs, multiplexers, nets, etc.), and do not distinguish between high to low and low to high transitions.

5.1.2.5 Timing Derating Factors

The propagation delays depend also on the temperature and the supply voltage. Variations in those parameters affect the circuit delay. The circuit manufacturers even give tables, graphics or embed the information in the static timing analyzer to adapt the timing to the operating conditions.

The derating factors are coefficients that the timing data are multiplied by, in order to estimate the delays corresponding to the operating conditions. Figure 5.5 shows typical derating curves for different operating temperature and supply voltage (the example is based on a 180 nm process). Table 5.2 is another form to show the same information.

5.1.3 Glitches in Digital Circuits

An electronics glitch is an undesired transition that occurs before the signal settles to its proposed final value. In other words, a glitch is an electrical pulse of short duration, usually unwanted, and typically produced by an imbalance in internal interconnection delays. A simple example is depicted in Fig. 5.6 where different

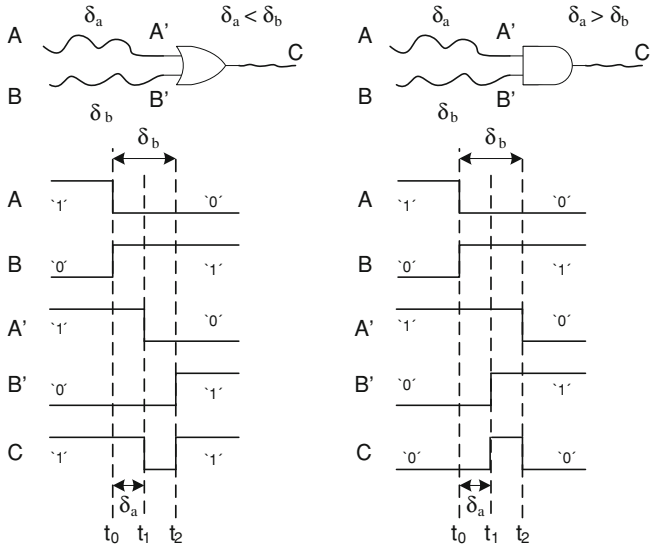


Fig. 5.6 Glitches of a simple gate due to unbalanced interconnection delay

delays in interconnections produce unnecessary changes at the output. For simplicity, in the preceding figure the gates delays are assumed to be equal to 0.

The glitch effect grows with the logic depth. As a simple example, consider two levels of XOR gates (Fig. 5.7). The four inputs change at the same time (t_0) but the net delays produce different times of arrival to the gates. In this case, changing from 1010 to 0101 should not produce any output change. Nevertheless, in the example, four transitions in output G are generated.

5.1.3.1 Runt Pulse and Spikes

A related concept to glitches is the runt pulse. It is a pulse whose amplitude is smaller than the minimum level specified for correct operation. It is a narrow pulse that, due to rise and fall times of the signal, does not reach a valid high or low level value. Typically, it has no influence on the operations, but it increases the power consumption (see Sect. 5.4).

Some authors define the concept of "spike" as being a short pulse similar to a glitch, but caused by ringing (an unwanted oscillation of a signal) or crosstalk (undesired capacitive or inductive coupling).

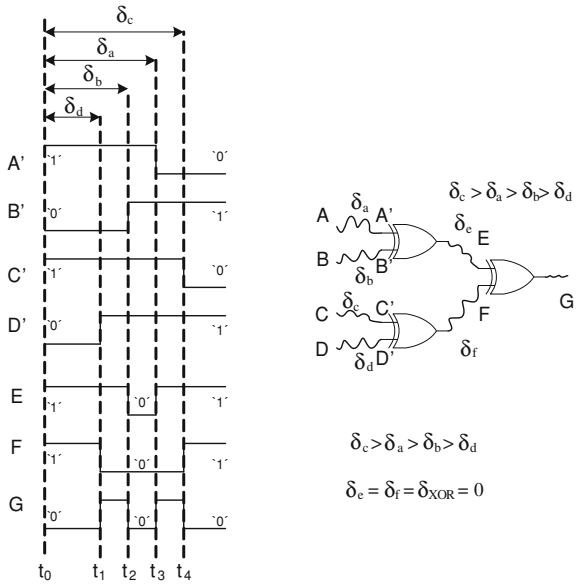


Fig. 5.7 The avalanche effect of glitches as the logic depth grows

5.2 Synchronous Design Issues

Digital circuits are mainly "synchronous". The Register Transfer Level (RTL) model recognizes register and combinational logic connected between them. The typical form, adopted to synchronize these components, is the True Single Phase Clock (TSPC). In TSPC a single clock signal is supposed to be distributed at different register levels. All the registers (flip-flops) are capturing on the rising (or falling) edge. Figure 5.8 illustrates the classical situation. A clock edge (rising in Fig. 5.8) captures the information into the first level of registers (ff_i). The contents of registers ff_i (B) are propagated into the combinational logic and generate new results. The output of the combinational circuitry (C) should be stable before the next clock arrives to the next register stage (ff_j). The following clock edge captures the content of C in ff_j and makes D available.

In an ideal synchronous circuit, every change in the logical levels of its registers (flip-flops) is simultaneous. These transitions follow the level change of the clock (*clk*) signal (positive or negative clock edge). In normal function, the input to each storage element has reached its final value before the next edge clock occurs, so the behavior of the whole circuit is exactly predictable.

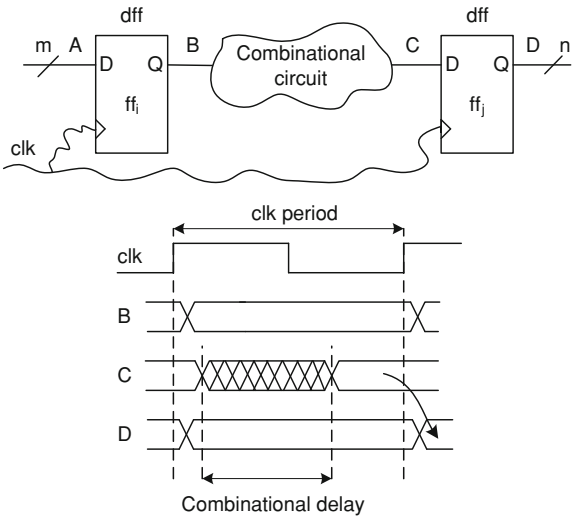


Fig. 5.8 Typical single phase synchronization

5.2.1 Edge Sensitive and Level Sensitive Registers

The edge sensitive and level sensitive registers (flip-flop and latch) are electronic circuits that have two stable states and can be used to store binary information. The circuits are made to change state by signals applied to one or more control inputs and will have one or two outputs. The latches are level sensitive circuits that pass their input D to their output Q when the clock is high (or low) (transparent mode), and the input sampled on the falling edge of the clock is held stable when the clock is low (or high)-hold mode. The flip-flops (also called edge-triggered latches) are edge sensitive circuits that sample the inputs on a clock transition (positive edge-triggered: 0 → 1 or negative edge-triggered: 1 → 0). They are built using latches (e.g., master–slave flip-flops). Historically, there have been different classes of flip-flops (FF) depending on the way they work (SR –"set-reset", D –"data", T – "toggle", and JK). Today's FPGA have registers that can be configured as latches or flip-flop, but only as a D-FF.

5.2.2 Temporal Parameters of Flip-Flops

The synchronous flip-flop requires that the data to be sampled is stable some time before and after (setup and hold times) the clock edge Fig. 5.9. If the input data changes in the setup-hold windows, metastability could occur (next section). To summarize: the *setup time* (t_{su}) is the minimum amount of time the data signal should be held steady *before* the clock event; and the *hold time* (t_h) is the minimum amount of time the data signal should be held steady after the clock event, so that

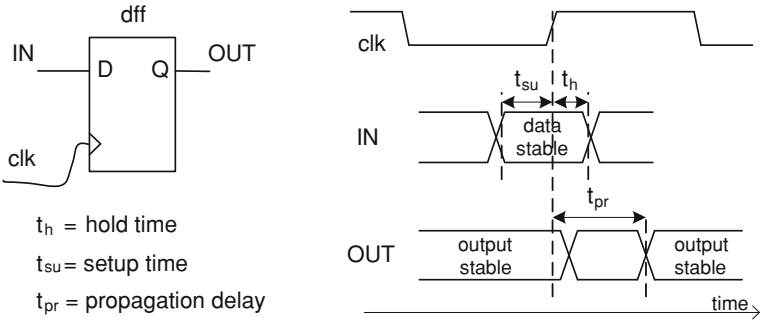


Fig. 5.9 Setup time, hold time and propagation delay of a register

the data are reliably sampled. These times are specified for any device or technology, and are typically between a few tens of picoseconds and a few nanoseconds for modern devices.

The data stored in the flip-flop is visible at its output as a *propagation delay* (t_{pr}) after the clock edge. This timing value is also known as *clock-to-output delay*.

Another related concept is the *minimum clock pulse* of a flip-flop. It is the minimum width of the clock pulse necessary to control the register.

5.2.3 Metastability

Whenever there is a setup or a hold time violation, the flip-flop could enter in a metastable state (a quasi-stable state). In this state the flip-flop output is unpredictable and it is considered as a failure of the logic design. At the end of a metastable state, when the output could reveal an "in between" value, the flip-flop settles down to either '1' or '0'. This whole process is known as metastability. Figure 5.10 illustrates this situation. The duration of the metastable state is a random variable that depends on the technology of the flip-flop. The circuit's vendors provide information about the metastability in their devices. As an example, the main vendors of FPGA provide this kind of information [3–5].

Comments 5.1

1. Not all the setup-hold window violations imply a metastable state. It is a probabilistic event.
2. Not all the metastability states cause design failures. In fact, if the data output signal resolves to a valid state before the next register captures the data, then the metastable signal does not impact negatively in the system operation.

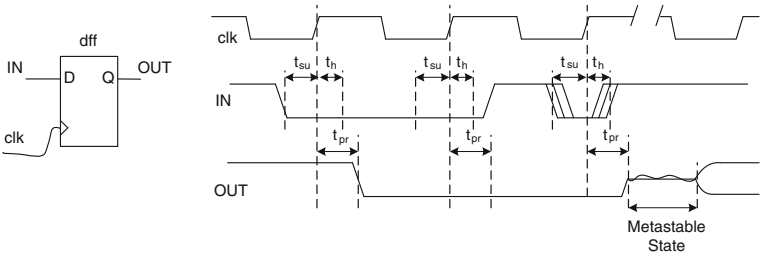


Fig. 5.10 Metastability capturing data in a flip-flop

5.2.3.1 Main Causes of Metastability

As previously mentioned, a setup or hold time violation, could produce metastability, so we have to see when signals violate this timing requirement:

- When the input signal is an asynchronous signal.
- When interfacing two domains operating at two different clock frequencies.
- When the clock skew is too high. (Sect. 5.3.1)
- When the clock slew rate is too high (rise and fall times are longer than the tolerable values).
- When interfacing two domains operating at the same frequency but with different phase.
- When the combinational delays are such that flip-flop data input change within the critical window. (setup-hold window)

Observe that that only the first two cases are real metastability problems and we will discuss possible solutions in the next sections. The other cases are related with the clock frequency and simple increases of the clock period could solve the problem. Even a static timing analysis detects these kinds of problems.

5.2.3.2 Mean Time Between Failures (MTBF) in Metastability

The Mean Time Between Failures (MTBF) is a general concept used in reliability. This concept, applied to the metastability, gives the average time interval between two successive failures due to this phenomenon. The expression that computes MTBF is:

$$MTBF = \frac{e^{T_{rec} \cdot K_2}}{K_1 \cdot f_{clk} \cdot f_{data}} \tag{5.1}$$

Where:

fclk is the frequency of the clock receiving the asynchronous signal.
fdata is the toggling frequency of the asynchronous input data signal.

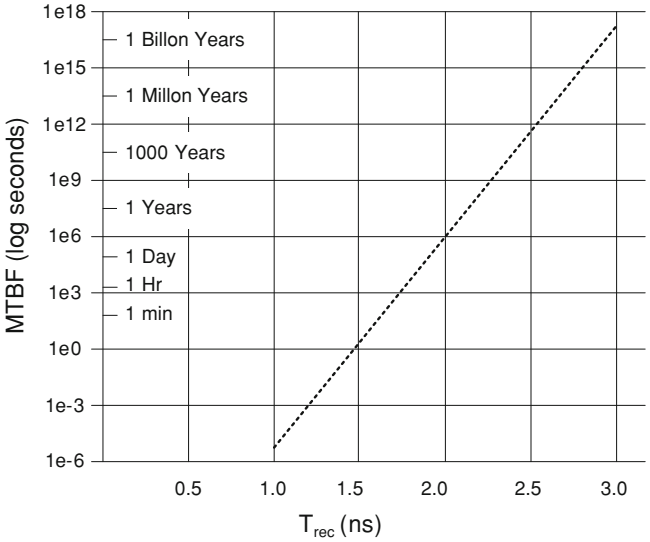


Fig. 5.11 MTBF for 300 MHz clock and 50 MHZ data in a 130 nm technology

Trec is the available metastability settling time (recovery time), or the timing until the potentially metastable signal goes to a known value '0' or '1'.

*K*1 (in *ns*) and *K*2 (in 1/*ns*) are constants that depend on the device process and on the operating conditions.

If we can tolerate more time to recover from the metastability (*Trec*) the MTBF is reduced exponentially. Faster clock frequencies (*fclk*) and faster-toggling (*fdata*) data worsen (reduce) the MTBF since the probability to have a setup-hold window violation augments. Figure 5.11 shows a typical MTBF graphic. These graphics are average data for 300 MHz clock and 50 MHz data in a 130 nm technology.

Comments 5.2

1. Observe the sensitivity to the recovery time. Following Fig. 5.11, if you are able to "wait" 2 *ns* to recover from metastability, the MTBF is less than two weeks. But if you can wait for 2.5 *ns* the MTBF is more than three thousand years.
2. Suppose for the previous data ($fclk = 300$ MHz, $fdata = 50$ MHz, $T_{rec} = 2.5$ *ns*) that give a MTBF of around 3200 years. But, if we have a system with 256 input bits the MTBF is reduced to 12.5 years. If we additionally produce 100,000 systems we have MTBF of 1 h!
3. Measuring the time between metastability events using real designs under real operating conditions is impractical, because it is in the order of years. FPGA vendors determine the constant parameters in the MTBF equation by characterizing the FPGA for metastability using a detector circuit designed to have a short, measurable MTBF [3, 5].

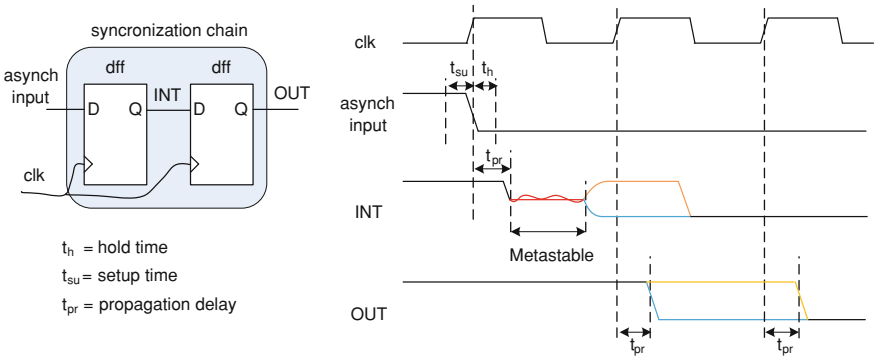


Fig. 5.12 Using a two flip-flops synchronizer to reduce metastability

5.2.3.3 How to Avoid or Mitigate Metastability

In reality, one cannot avoid metastability, without the use of tricky self-timed circuits. So a more appropriate question might be "How to mitigate the effect of metastability?"

The simplest solution is by making sure the clock period is long enough to allow the resolution of metastable states and for the delay of whatever logic may be in the path to the next flip-flop. This approach, while simple, is unpractical given the performance requirements of modern designs.

The classical solution is the use of a sequence of registers (a synchronization registers chain or synchronizer) in the destination clock domain (Fig. 5.12). The idea is to cascade one or more successive synchronizing flip-flops to the flip-flop that is interfacing with the asynchronous input.

This approach cannot guarantee that metastability cannot pass through the synchronizer; they simply reduce the probability to practical levels. In order to evaluate the effect, consider the three cases of Fig. 5.13, assuming that $t_{comb} + t_{su} \cong t_{clk}$ (combinational delay plus setup time similar to clock period). In the first case, without registering the asynchronous input, T_{rec} is near zero, raising the MTBF to an unreliable system. In the second case, with a simple flip-flop capturing the asynchronous input, $T_{rec} \cong t_{clk} - (t_{comb} + t_{su})$. In the final case, using two FFs to interface the asynchronous input, the recovery time $T_{rec} \cong t_{clk} - t_{su}$. For the data of Fig. 5.11 (i.e. $t_{clk} = 3.33$ ns) and assuming a tsu = 350 ps, the MTBF of the synchronization chain is more than 1 billon years.

5.3 Clock Distribution Network

The clock distribution network (or clock tree) distributes the clock signal(s) from a common input (an input pin) to the entire synchronous element (registers). Since this function is vital in an synchronous system, special attention is given to the

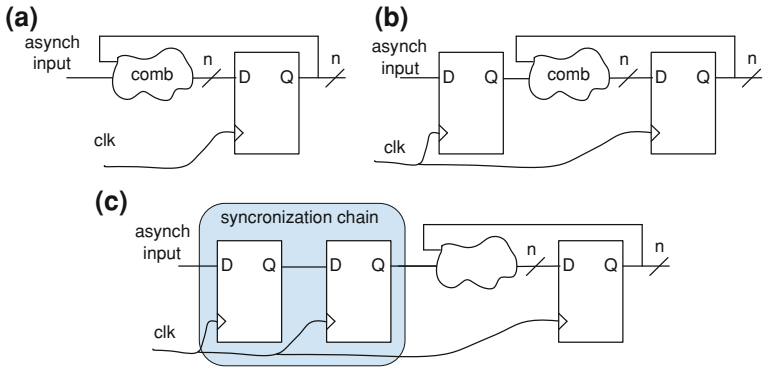


Fig. 5.13 Synchronizing asynchronous inputs in a design due to the metastability problem. **a** No registers. **b** One synchronization register. **c** Two register in synchronization chain

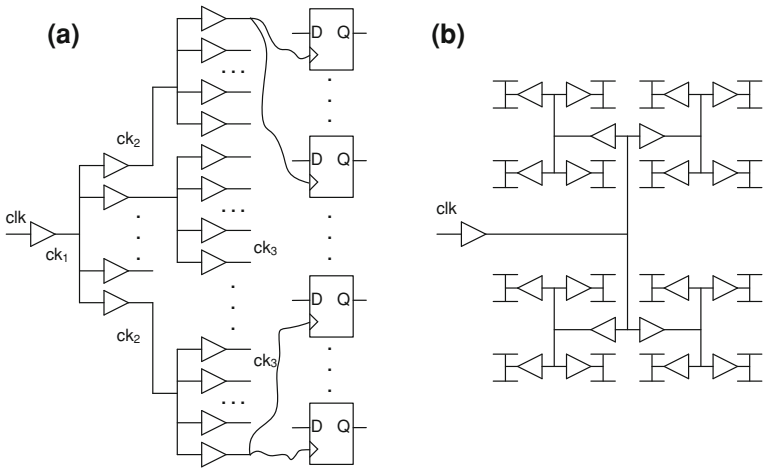


Fig. 5.14 Typical clock tree distribution. **a** Buffers tree. **b** Ideal H-routing distribution

design of this component. In ASIC design a special clock synthesizer is used (a step in physical synthesis) during the design. In FPGA several dedicated pre-fabricated clock distribution networks are present.

Clock signals are typically loaded with the greatest amount of interconnections and operate at the highest speeds of any signal in the system. Today's clock signals are easily distributed to tens of thousands of points. Due to the driving capability of the buffers, the clock distribution is designed as trees (Fig. 5.14a).

As a technological example, suppose use of the data of Table 5.1, and use the buffer BUF to distribute a clock signal. If we neglect the interconnection load (an unreal case), a simple BUF can drive 141 flip-flops (DFF) (0.425 *pf*/0.003 *pf*) and also 106 similar buffers (0.425 *pf*/0.004 *pf*). If we want to distribute the signal to

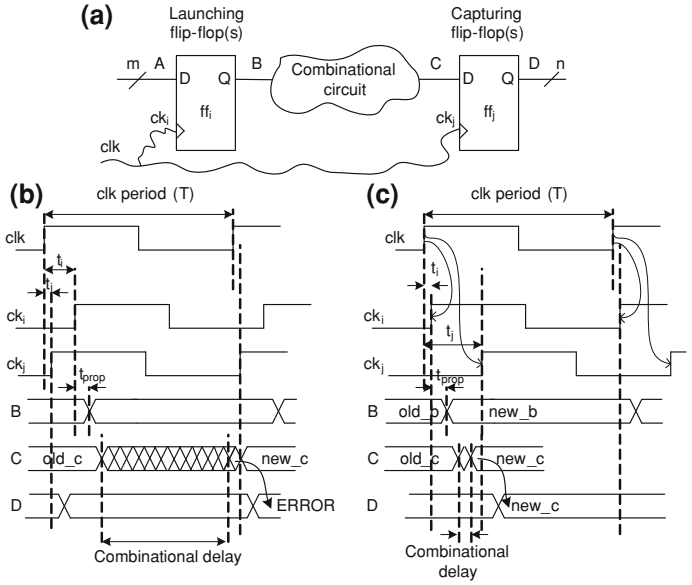


Fig. 5.15 Setup and hold violations due to clock skew. **a** Example circuit. **b** Setup violation or long-path fault. *B*. Hold violation or race-through

64,000 FF we need at least three levels of those buffers. Real clock trees use special buffers with more driving capability and with other special characteristics, and have additional problems related with the interconnections. Figure 5.14 shows a classic clock tree that present multiple levels of buffers, and the typical H-routing used to reduce the skew.

5.3.1 Clock Skew

The variation in the arrival times of a clock transition at two different locations on a chip is commonly known as the clock skew (also timing skew). Being the cause of the unbalanced delays in the clock distribution network, clock edges could arrive at clock pins ck_i and ck_j in Fig. 5.15 at different times. This spatial variation can be caused by many different things, such as wire-interconnect length, temperature variations, material imperfections, capacitance coupling, etc. The skew between two registers stages i and j could be defined as

$$skew_{i,j} = delay(ck_i) - delay(ck_i) = t_i - t_j \tag{5.2}$$

That leads to two types of clock skew: positive skew ($skew_{i,j} > 0$) and negative skew ($skew_{i,j} < 0$). Positive skew occurs when the transmitting register (launching flip-flop, ff_i) receives the clock later than the receiving register (capturing

flip-flop, ff_j). Negative skew is the opposite; the sending register gets the clock earlier than the receiving register. Two types of time violation (synchronization failures) can be caused by clock skew: setup and hold violations. They are described in what follows.

5.3.1.1 Setup Violation due to Clock Skew

If the destination flip-flop receives the clock edge earlier than the source flip-flop (positive skew), the data signal has that much less time to reach the destination flip-flop before the next clock edge. If it fails to reach the destination timeously, a *setup violation* occurs, so-called because the new data was not stable before the Tsu (setup time) of the next clock edge. This error is drawn in Fig. 5.15b and is also called *zero-clocking* (because nothing is captured properly) or also *long-path fault*.

In this case, at time $t_i^{\max}$(assuming several FF at level i, the worst case), the clock edge arrives at flip-flops ff_i. In the worst case, to propagate through the flip-flops and the combinational logic it takes$t_{prop}^{\max} + t_{comb}^{\max}$. The signal must have settled down for a duration of $t_{su}^{\max}$before the next clock edge arrives at the next flip-flop stage (ff_j) at time $t_j^{\min} + T$in order to be captured properly. That transform in the following general setup time constrain:

$$t_j^{\min} + T > t_i^{\max} + t_{prop}^{\max} + t_{comb}^{\max} \tag{5.3}$$

Observe that this inequation could be always satisfied if the clock period (T) could be incremented. In other words, a positive clock skew, as shown in Fig 5.15b, places a lower bound on the allowable clock period (or an upper bound on the operating frequency) as follows:

$$T > t_{prop}^{\max} + t_{comb}^{\max} + t_{su}^{\max} + skew_{i,j}^{\max} \tag{5.4}$$

While a positive $\text{skew}_{\text{i,j}}$ decreases the maximum achievable clock frequency, a negative clock skew ($\text{skew}_{\text{i,j}} < 0$) actually increases the effective clock period. Effectively, we may have a combinational block between two adjacent flip-flops that has a propagation delay longer than the given clock period.

5.3.1.2 Hold Violation due to Clock Skew

The hold violation is caused when the clock travels more slowly than the path from one register to another (negative skew), allowing data to penetrate two registers in the same clock tick, or maybe destroying the integrity of the latched data. This is called a **hold violation** because the previous data is not held long enough at the destination flip-flop to be properly clocked through. This situation is also known as

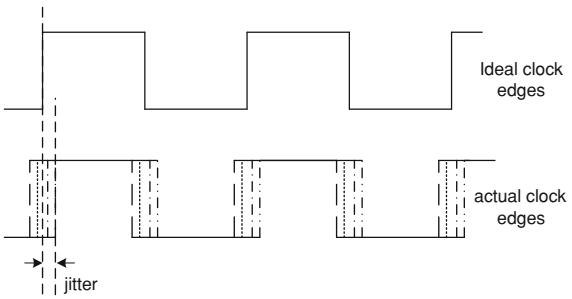


Fig. 5.16 Clock jitter example

double-clocking (because two FFs could capture with the same clock tick) or *race-through* and is common in shift registers.

To describe this situation, consider the example in Fig. 5.15c. At time $t_i^{\min}$, the clock edge triggers flip-flops ff_i, and the signals propagates through the flip-flop and the combinational logic $t_{prop}^{\min} + t_{comb}^{\min}$ (we use the shortest propagation delay, because we are considering the possibility of data racing). The input signal at ff_j has to remain stable for $t_{hold}^{\max}$ after the clock edge of the same clock cycle arrives ($t_j^{\max}$). We are using max value since we are considering the worst case. To summarize the constraint to avoid race condition (hold violation) is:

$$t_j^{\max} + t_{hold}^{\max} < t_i^{\min} + t_{prop}^{\min} + t_{comb}^{\min} \tag{5.5}$$

This can be rewritten as a constraint for the skew:

$$skew_{i,j} > t_{hold}^{\max} - t_{prop}^{\min} - t_{comb}^{\min} \tag{5.6}$$

Observe that a hold violation is more serious than a setup violation because it cannot be fixed by increasing the clock period.

In the previous analysis we consider several launching and capturing flip-flops. If the entire FFs have the same characteristics we do not need to consider maximum or minimum setup and hold time in equations 5.3, 5.4, 5.5 and 5.6.

5.3.2 Clock Jitter

The clock edges at a flip-flop may sometimes arrive earlier and sometimes later with respect to an ideal reference clock, depending on the operating condition of the circuit. Such temporal variation in the clock period is referred to as clock jitter.

The concept of absolute jitter refers to the worst case deviation (absolute value) of the arrival time with respect to an ideal reference clock edge (Fig. 5.16).

The clock period may be shortened or lengthened by the clock jitter. For timing purposes the worst case is considered, then jitter impacts negatively in the maximum frequency of operation.

There are several sources of clock jitter. The analog component of the clock generation circuitry and the clock buffer tree in the distribution network are both significant contributors to clock jitter. There are also environmental variations that cause jitter, such as power supply noise, temperature gradients, coupling of adjacent signals, etc.

5.3.3 Clock Gating

Since the clock distribution network could take a significant fraction of the power consumed by a chip and, moreover, a large amount of power could be wasted inside computations blocks even when their output are not used (unnecessary switching activity), a typical power saving technique applied is the clock gating, which selectively switches off part of the clock tree.

Clock gating works by taking the enable conditions attached to registers, and uses them to gate the clocks (Figs. 5.17a and b). The clock gating logic can be added into a design mainly in two ways. Coding into the RTL code as enable conditions and using *automatic clock gating* by synthesis tools, or manually inserting library specific modules that implement this functionality.

Using clock gating in FPGA deserves some especial considerations. The FPGA have several low skew and low jitter dedicated clock trees; the use of this network is vital to distribute the clock and achieve high clock frequencies. In order to disable safely the clock tree, special clock buffers should be used that allow switching off the clock without glitches. Observe the potential problem to use general logic to implement gated clock in the timing diagram of Fig. 5.17c.

Today's FPGA have several clock trees and also clock buffers to disable part of it safely at different granularities (more or less logic elements). Moreover, the synthesis tools allow applying this technique automatically.

5.3.4 Clock Managers

A clock manager is a general name to describe a component that can manipulate some characteristics of the input clock. The most typical actions performed by a clock manager are:

- Delay Locked Loop (DLL), synchronizes the input clock signal with the internal clock (clock deskew, Fig. 5.18).
- Frequency Synthesis (FS): Multiply and divide an incoming clock.
- Phase Shifter (PS): a phase shift (skew) with respect to the rising edge of the input clock may be configured.
- Recondition clock signal: Reduce jitter, duty cycle correction, etc.

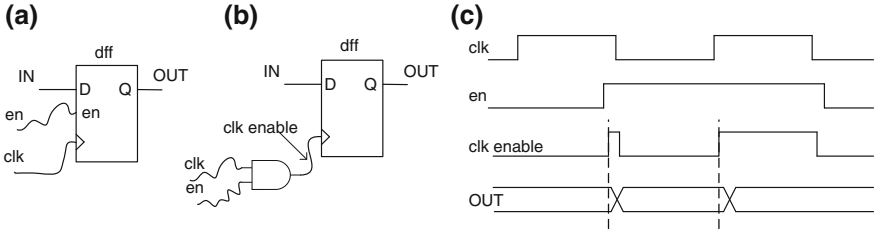


Fig. 5.17 Clock gating. **a** Register with enable. **b** Register with clock gating. **c** Clock gating risk

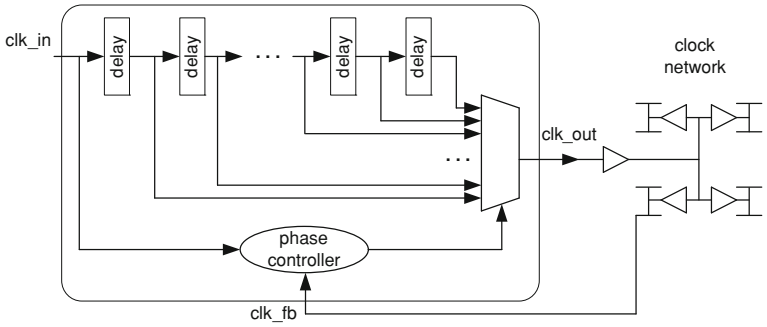


Fig. 5.18 A typical delay locked loop used to synchronize the internal clock

Today's FPGA have several of those components, for example the Xilinx Digital Clock Managers (DCM) and the Altera Phase-Locked Loops (PLL). One of the main actions performed by those components is to act as a Delay Locked Loop (DLL).

In order to illustrate the necessity of a DLL, consider the clock tree described in Sect. 5.3.1, Fig. 5.14, based on the cells of Table 5.1, that is, 64,000 FF that uses three levels of buffers. The interconnection load is neglected (an unreal case). Assume, additionally, that we decide to connect 40 buffers to the first buffer, 40 buffers at the output of each buffer of the second level, and finally 40 FFs at the output of the last 1600 buffers. We will calculate the transition time of a rising edge at input clk using the concepts of Sect. 5.1.2.4. ($T_{lh} = t\_int_{lh} + t\_ext_{lh} \cdot$ capacity). In this scenario the signal ck_1 will see the rising edge 0.44 ns later (0.056 ns + 2.399 ns/pf · (40 · 0.004pf + interconnection)). The ck_2 rising edge will be 0.44 ns after the ck_1, and finally the ck_3 will be propagated 0.344 ns after the ck_1. That is the clk signal will be at the FF 1.22 ns later (remember that for simplicity we do not consider the interconnection load). If we want to work with a clock frequency of 400 MHz (2.5 ns period) the clock will arrive half a cycle later.

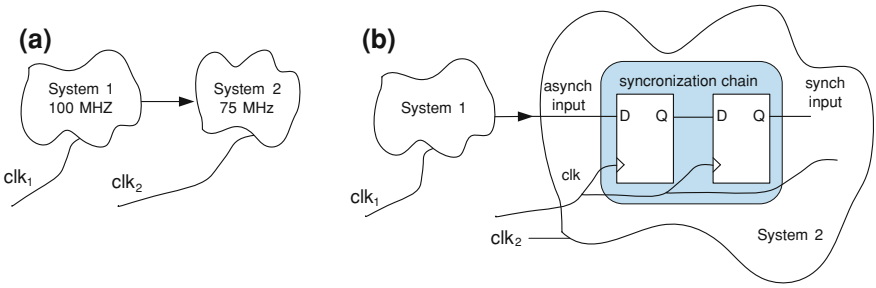


Fig. 5.19 Interfacing different clock domains using synchronization chain

5.3.4.1 Delay-Locked Loop (DLL)

A delay-locked loop (DLL) is conceptually similar to a phase-locked loop (PLL) but using a different operating principle. The main objective is to maintain the same phase of the input clock inside a device.

The main component of a DLL is a delay chain composed of many delay components. The input of the chain, and thus of the DLL, is connected to the input clock (clk_in). A multiplexer is connected to each stage of the delay chain; the selector of this multiplexer is a phase controller that compares the clock feedback (clk_fb) from the clock network and the input clock.

This circuit, after several clock cycles, ensures that the input clock rising edge is in phase with the clock feedback rising edge (offset = 360°).

5.3.5 Interfacing Different Clock Domains

Several times, a digital design needs to interface two different clock domains. This interfacing is difficult in the sense that design becomes asynchronous at the interface boundary, which could fail in setup and hold violations with the consequent metastability (see Sect. 5.2.3). Hence, particular design and interfacing techniques are necessary.

Two systems become asynchronous to each other when they operate at two different frequencies or when they operate at the same frequency, but using different and independent clock sources (Fig. 5.19a). Observe that if we use the same clock source and a multiple of the same clock (for example, divided or multiplied by 2) or a phase shifting of the clock (180°) they do not necessarily become asynchronous.

Synchronization failure is lethal and difficult to debug, so it is important to take care about this issue. If we have two systems, asynchronous to each other, and we need to transfer data between them, several methods can be considered:

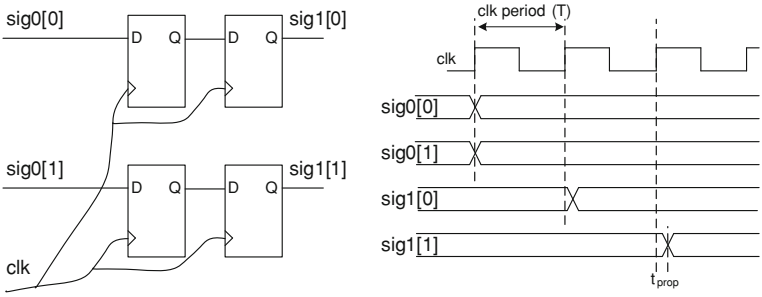


Fig. 5.20 Synchronization problems with buses

- Using synchronizer.
- Handshake signaling method.
- Asynchronous FIFO.
- Open loop communication.

5.3.5.1 Using Synchronizer

It is the simplest method, useful only for very low speed communication. As described in Sect. 5.2.3, a simple method to mitigate metastability is the use a cascade of flip-flops. Fig. 5.19b shows the simple solution using a synchronization chain (synchronizer).

This method could lead to errors when it is used to synchronize a bus. As a simple example consider Fig. 5.20. If the input bus changes at the capturing edge of the clock, different paths and different flip-flops could react differently, making a misalignment of the data in the bus. The answer to this problem is the use of a handshaking protocol.

Comments 5.3

A related synchronization pitfall could occur if a single bit is synchronized in different places. The same problem described for buses in Fig. 5.20 could appear.

5.3.5.2 Handshake Signaling

In this method system 1 sends data to system 2 based on the handshake signals *req* (request) and *ack* (acknowledge). A simple handshaking protocol (known as 4-phase) will work as follow:

- Sender outputs data and asserts *req*.
- Receiver captures data and asserts *ack*.
- Sender, after *ack* is detected, deasserts *req*.
- Receiver sees *req* deasserted, deasserts *ack* when ready to continue.

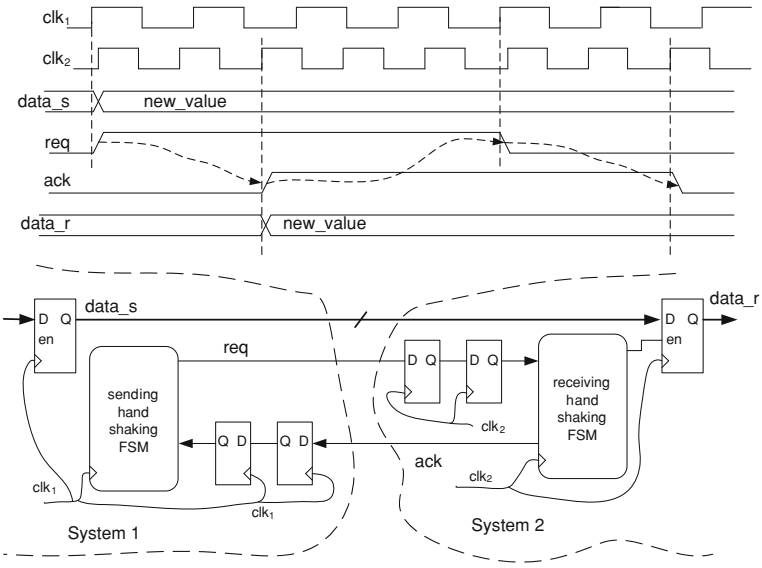


Fig. 5.21 Handshaking protocol using a two stage synchronizer

This method is straightforward, but the metastability problems could be present. In fact, when system 2 samples system 1's *req* and system 1 samples system 2's *ack* line, they use their internal clock, so setup and hold time violations could arise. To avoid this, we can use double or triple stage synchronizers, which increase the MTBF and thus reduce the metastability to a good extent. Figure 5.21 shows a handshaking protocol using a two stage synchronizer in *req* and *ack* lines.

As shown in the example, the use of double or triple stage synchronizing reduces significantly the transfer rate, due to the fact that a lot of clock cycles are wasted for handshaking.

However, a simpler handshaking is possible (2-phase or edge based). In this, the sender outputs data and changes the state of *req*; it will not change the state of *req* again until after the state of *ack* changes. The receiver latches data; once the receiver is ready for more, it changes the state of *ack*. This method (2-phase) requires one bit of state to be kept on each side of the transaction to know the *ack* state. Additionally, a reliable reset is necessary to start the synchronization.

Handshaking works great, but reduces the bandwidth at the clock crossing interface because many cycles of handshaking are wasted. The high bandwidth solution that maintains reliable communication is the use of asyncronous FIFOs.

5.3.5.3 Asynchronous FIFO

An asynchronous FIFO (First In First Out) has two interfaces, one for writing the data into the FIFO and the other for reading the data out (Fig. 5.22a). Ideal dual

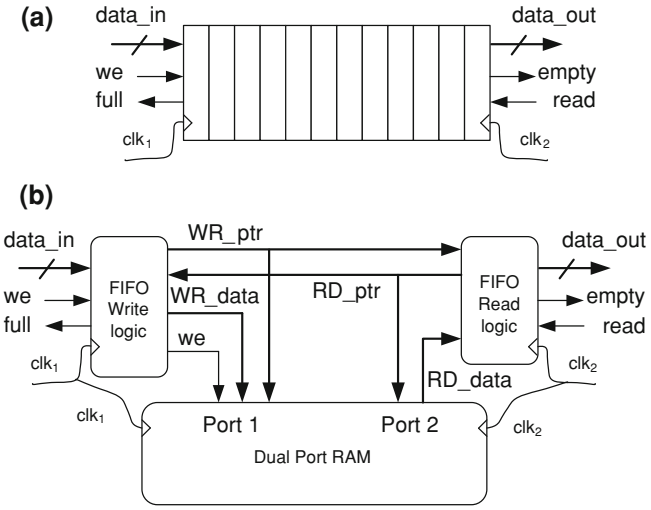


Fig. 5.22 Asynchronous FIFO. **a** Abstract FIFO design. **b** Detailed view

port FIFOs write with one clock, and read with another. The FIFO storage provides buffering to help rate match between different frequencies. Flow control is needed in case the FIFO gets totally full or totally empty. These signals are generated with respect to the corresponding clock. The *full* signal is used by system 1 (when the FIFO is full, we do not want system 1 to write data because this data will be lost or will overwrite an existing data), so it will be driven by the write clock. Similarly, the *empty* signal will be driven by the read clock.

FIFOs of any significant size are implemented using an on-chip dual port RAM (it has two independent ports). The FIFO is managed as a circular buffer using pointers. A write pointer to determine the write address and a read pointer to determine the read address are used (Fig. 5.22b). To generate *full/empty* conditions, the write logic needs to see the read pointer and the read logic needs to see the write pointer. That leads to more synchronization problems (in certain cases, they can produce metastability) that are solved using synchronizers and Gray encoding.

Comments 5.4

1. Asynchronous FIFOs are used at places where the performance matters, that is, when one does not want to waste clock cycles in handshake signals.
2. Most of today's FPGAs vendors offer blocks of on-chip RAMs that can also be configured as asynchronous FIFOs.
3. A FIFO is the hardware implementation of a *data stream* used in some computation models.

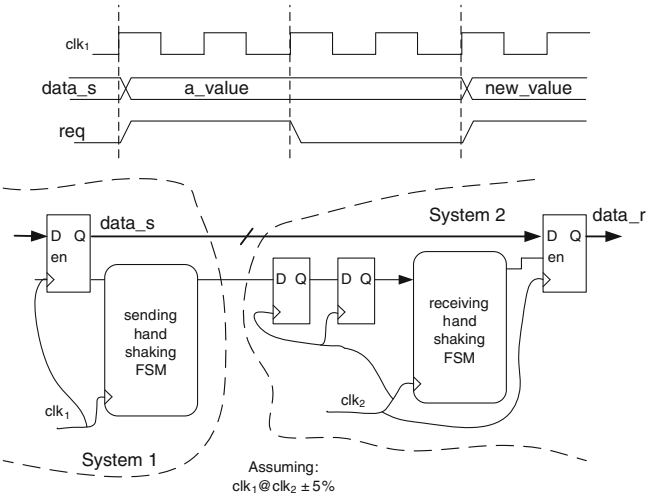


Fig. 5.23 Open loop communication (mesosynchronous)

5.3.5.4 Open Loop Communication

When two systems of bounded frequency need to communicate, open loop synchronization circuits can be used (also known as *mesosynchronous designs*). In this approach no *ack* signal is used)

The benefits of mesosynchronous designs are in less synchronization based circuitry and are of a lower latency in contrast to a 4-phase handshaking. The main drawback is that this method only works at certain frequency ratios (clk1 $\cong$ clk2 $\pm$ 5%). Figure 5.23 shows this synchronization method assuming that the sender holds two cycles of the *req* signal.

5.4 Power Consumption

Power dissipation is one of the main concerns in today's digital design. For portable devices the battery life is essential, the amount of current and energy available in a battery is nearly constant and the power dissipation of a circuit or system defines the battery life. On the other hand, the heat generated is proportional to the power dissipated by the chip or by the system; an excessive heat dissipation may increase the operating temperature and thus degrades the speed (see Sect. 5.1.2.5, derating factor) and causes circuitry malfunctions. The necessary cooling systems (fans, heatsinks, etc.) when excessive power is used, increase the total system cost. Additionally, the life of the circuitry is typically shortened when working at higher temperatures (electromigration and others).

5.4.1 Sources of Power Consumption

CMOS gates are considered very power efficient because they dissipate nearly zero power when idle. As the CMOS technology moved below sub-micron size this assumption became relative since the static power is more important. The power dissipation in CMOS circuits occurs because of two components:

- Static power dissipation. Is the power consumed when the output or input are not changing. The main contributions are the subthreshold and leakage current.
- Dynamic power dissipation. It is the power consumed during state transitions. The two main components are charging and discharging of load capacitances and the short circuit dissipation.

Thus the total power dissipation is the addition of the static power dissipation plus the dynamic power dissipation.

5.4.1.1 Static Power Consumption

In the past, the subthreshold conduction of transistors has been very small, but as transistors have been scaled down, leakages from all sources have increased. Historically, CMOS designs operated at supply voltages much larger than their threshold voltages (V_{dd} 5 V, and V_{th} around 700 mV) and the static power consumption was negligible. As transistor size is reduced, i.e. below 90 nm, the static current could be as high as 40% of the total power dissipation. The static power consumption reduction is one of the main concerns in today's technological development of new CMOS processes.

5.4.1.2 Dynamic Power Consumption

CMOS circuits dissipate power mainly by charging the different load capacitances (gates, wire capacitance, etc.) whenever they are switching (Fig. 5.24). In one complete cycle of CMOS logic, current flows from power supply V_{dd} to the load capacitance (0 to 1 transition) to charge it and then flows from the charged load capacitance to ground during discharge (1 to 0 transition). Therefore in one complete charge/discharge cycle, a total load $Q = C \cdot V_{dd}$ is thus transferred from V_{dd} to ground. If we multiply by the switching frequency to get the current, and multiply again to supply voltage we obtain the power dissipated for a gate as: $P = f \cdot C \cdot V_{dd}^2$ Some authors recognize the transition power (the power per transition) as $P = 1/2 \cdot f \cdot C \cdot V_{dd}^2$ (half power charging, half the power discharging). Since most gates do not operate/switch at the clock frequency, they are often accompanied by a factor α, called the activity factor (also switching activity). Now, the dynamic power dissipation may be re-written as $P = \alpha \cdot f \cdot C \cdot V_{dd}^2$.

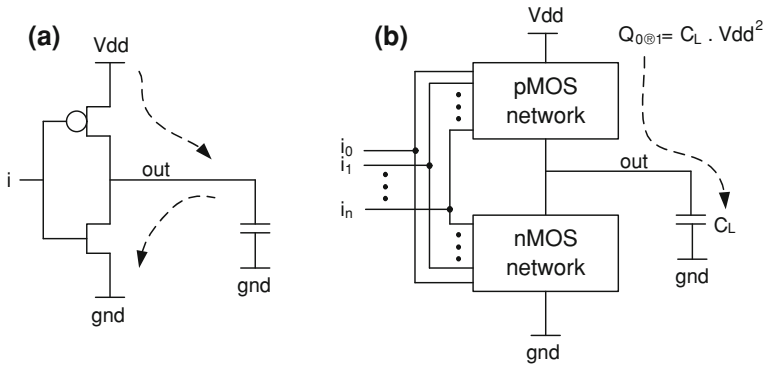


Fig. 5.24 Dynamic power consumption. **a** An inverter. **b** A general CMOS gate charging a capacitance

A clock in a system has an activity factor $\alpha = 1$, since it rises and falls every cycle. Most data has an activity factor lower than 0.5, i.e. switches less than one time per clock cycle. But real systems could have internal nodes with activity factors grater then one due to the glitches.

If correct load capacitance is calculated on each node together with its activity factor, the dynamic power dissipation at the total system could be calculated as:

$$P = \sum_i f \cdot (\alpha_i \cdot c_i) \cdot V_{dd}^2 \tag{5.7}$$

Another component in the dynamic component is the short circuit power dissipation. During transition, due to the rise/fall time both pMOS and nMOS transistors will be on for a small period of time in which current will find a path directly from V_{dd} to gnd, hence creating a short circuit current. In power estimation tools this current also can be modeled as an extra capacitance at the output of the gates.

5.4.1.3 Power and Energy

Power consumption is expressed in *Watts* and determines the design of the power supply, voltage regulators and the dimensions of the interconnections and eventually short time cooling. Moreover, the energy consumed is expressed in *Joules* and indicates the potency consumed over time, as shown in Equation 5.8.

$$\text{Energy} = \text{power} * \text{delay}\,(\text{joules} = \text{watts} * \text{seconds}) \tag{5.8}$$

The energy is associated with the battery life. Thus, less energy indicates less power to perform a calculation at the same frequency. Energy is thus independent

of the clock frequency. Reducing clock speed alone will degrade performance, but will not achieve savings in battery life (unless you change the voltage).

That is why, typically, the consumption is expressed in mW/Mhz when comparing circuits and algorithms that produce the same amount of data results per clock cycle, and normally nJoules (nanoJoules) are used when comparing the total consumption of different alternatives that require different numbers of clock cycles for computing.

5.4.2 Power Reduction Techniques

The power reduction could be tackled at different abstraction levels. At higher abstraction levels, bigger possibilities to reduce the power exist.

The static power consumption is reduced mainly at a technological level and is beyond the scope of this book. Excellent surveys and trends are in [6] and [7].

To a designer of digital systems, Equation 5.7 gives information about the main sources of reduction. The quadratic influence of the power supply is an interesting source of power reduction. The lower supply voltage the device employs, the lower power the systems will use. Today's systems are supplied at different voltages for cores and pads in order to reduce, when possible, the power budget.

On the other hand, the operation frequency (f), the activity (α), and the capacitance (c) are linear sources of power reduction.

At circuit level, the reduction of glitches, data reordering to reduce activity, or the fan-out reduction are typical techniques. Parts of these ideas are present in the automatic synthesis for low power provided by EDA tool vendors.

At algorithm level, the complexity and the regularity of algorithms, the data representation, and other techniques, offer a big scope for optimizations. The concept of energy and power is useful in this case (Sect. 5.4.1.3); a faster algorithm consuming the same power will consume less energy.

At system level, the partition of the system allows to apply dynamic power management, that is, applying sleep modes (shut down) to part of or to the complete system. These power management techniques could be applied manually by the designer or partially automated by the synthesis tools.

The power consumption is an important issue for today's FPGA vendors. Compared to ASICs, FPGAs are not power-efficient components because they use a larger amount of transistors to provide programmability on the chip (typically an order of one magnitude more power). That leads to some specific FPGAs called "low power FPGA families", optimized for power dissipation at the cost of performance. FPGA vendors use different transistors, trading off speed versus power in their devices, and also the availability to shut off the unused components. They also offer power estimation tools and some optimization tools at synthesis level.

5.4.3 Power Measurement and Estimation

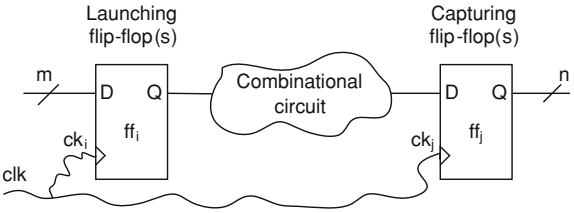
The power measurement is based on the measure of the supply voltage and the instant current (P = I * V). Some systems have supply sources that measure the current and allow measurement of the power. There also are other methods to measure the total energy consumed, but they are out of scope of the present discussion.

For power estimation, there are specific tools either for ASIC and FPGA. Some tools use estimated activities for different blocks; others are based on post place and route simulation to obtain the "exact" activity. The power estimation tool uses this activity information in Equation 5.7 to calculate the dynamic power consumption (more details are given in Chap. 6). The main drawback is the possibility to simulate a real situation to obtain the "real" activity of a circuit.

5.5 Exercises

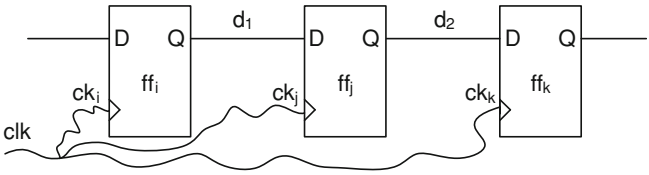
1. Redraw Fig. 5.5 (glitches for unbalanced paths) considering: (a) NAND gate, (b) NOR gate, (c) XOR gate.
2. Redraw Fig. 5.6 (cascade effect of glitches paths) considering the transition (a) from 0000 to 1111; (b) from 0100 to 1101; (c) from 0101 to 1101;
3. Supposing one has a simple ripple carry adder (Fig. 7.1) of four bits. Analyze the glitch propagation when changing to add 0000 + 0000 to 1111 + 1111.
4. What is the rise time and fall time of an AND2 gate that connects at his output four XOR2 gates? Assume a total interconnection load of 12 pf (use data of Table 5.1).
5. How many DFF can drive a CKBUF assuming the unreal case of no interconnection capacitance? Which is the propagation delay of the rising edge?
6. Assuming the derating factors of Table 5.2, what is the delay of exercise 4 and 5 for 80°C and a supply voltage of 3.0 V.
7. Determine the MTBF for $K_1 = 0.1$ ns, $K_2 = 2\ ns^{-1}$; with clock frequency of 100 MHz and data arrival at 1 MHz, for recovery time of 1, 5, 10 and 20 ns.
8. What is the MTBF expected for an asynchronous input that uses two synchronization flip-flops working at 100 MHz and using the data of the previous problem? The FFs have a setup time of one ns.
9. What is the delay of the falling edge with the data used in Sect. 5.3.4. i.e. 64,000 FF that uses three levels of BUF, neglecting the interconnection load?
10. Calculate the level of clock buffers (CKBUF) necessary to control 128,000 registers (DFF) of Table 5.1. Suppose additionally that any interconnection has a load of 0.006 pf.
11. For the previous clock tree. Determine the propagation delay from the input clock signal to the clk input of a FF.

12. Draw a timing diagram of a two stage handshaking protocol. Assume that the sending clock is faster than the receiving clock.
13. For the circuit of the figure, suppose that the FF has a propagation delay between 0.9 and 1.2 ns, a setup time between 0.4 and 0.5 ns and a hold time between 0.2 and 0.3 ns.

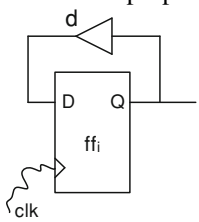


The clock arrives to the different FF of level i with a delay between 2.1 ns and 3.3 ns, and to level j with delays between 2.5 ns and 3.9 ns. What is the maximum combinational delay acceptable to work at 100 MHz?

14. Using the data of the previous exercise, what is the minimum combinational delay necessary to ensure a correct functionality?
15. A system 'A' works with a supply voltage of 1.2 V and needs 1.3 mA during 10 s to perform a computation. A second system 'B' powered at 1.0 V consumes an average of 1.2 mA and needs 40 s to perform the same task. Which consumes less power and energy?
16. In the shift register of the figure, assuming that all the flip-flops have a propagation delay of 0.9 ns, a setup time of 0.3 ns and a hold time of 0.2 ns, what is the maximum skew tolerated if the interconnection has a delay (δ1 and δ2) of 0.1 ns?



17. For the previous problem. What is the maximum frequency of operation?
18. The following FF (with same temporal parameters as in exercise 16) is used to divide the clock frequency. What is the minimum delay δ of the inverter and interconnection necessary for it to work properly?



References

1. Rabaey JM, Chandrakasan A, Nikolic B (2003) Digital integrated circuits, 2nd edn. Prentice-Hall, Englewood Cliffs
2. Wakerly JF (2005) Digital design principles and practices, 4th edn. Prentice-Hall, Englewood Cliffs. ISBN 0-13-186389-4
3. Xilinx Corp (2005) XAPP094 (v3.0) Metastable Recovery in Virtex-II Pro FPGAs, XAPP094 (v3.0). http://www.xilinx.com/support/documentation/application_notes/xapp094.pdf
4. Actel Corp (2007) Metastability characterization report for Actel antifuse FPGAs; http://www.actel.com/documents/Antifuse_MetaReport_AN.pdf
5. Altera corp (2009) White paper: Understanding metastability in FPGAs. http://www.altera.com/literature/wp/wp-01082-quartus-ii-metastability.pdf
6. Pedram M (1996) Tutorial and survey paper—Power minimization in IC design: principles and applications. ACM Trans Design Autom Electron Syst 1(1):3–56
7. Rabaey JM (1996) Low power design methodologies. Kluwer Academic Publishers, Dordrecht