2 Centralized Clocking

Chapter Contents

Learning Goals

This chapter revisits some of the simplifications made in the previous chapter. In particular it discusses the timed nature of components. The chapter then summarizes classical approaches to build state machines with an emphasis on building clock trees.

2.1 Overview

A major advantage of the abstractions if circuit components that have been discussed in Chapter 1 is that they abstract from the timed, analog behavior of physical components. Rather, the binary steady state of all components together is considered. Depending on the design paradigm, timing has to be reintroduced into this view to a certain degree to guarantee correct circuit behavior. In the following we focus on state-machine implementations to discuss such assumptions.

Chapter 1 discussed the standard state-machine implementation within the synchronous design paradigm. With the abstractions made in the chapter there was a single timing constraint to be introduced: the worst-case delay of the combinational logic, that is, the duration from application of inputs to the outputs reaching their steady-state, must be smaller than the duration between two active clock transitions at the flip-flops.

We will revisit this scheme of implementing state-machines in Section 2.2. In particular, we will revisit the previously introduced circuit components, flip-flops and Boolean gates, with respect to timed, non-steady-state, behavior.

In Section 2.3 we will discuss timing that has to be reintroduced into the model to arrive at correct and also permanent solutions to the state-machine implementation problem.

Additional improvements to increase performance are discussed in Section 2.4.

While having detailed on the logic components and the clock distribution network in the previous sections, we have still assumed the clock source to be an (almost) perfect periodic pulse generator. Section 2.5 briefly discusses this assumption and its limitation.

Section 2.6 discusses fundamental limitations of a single clock source. In particular we will show that the clock skew within a grid network is necessarily linear in the length of the grid:

Theorem 2.4. *For $k \in \mathbb{N}$, consider a $k \times k$ grid in $\mathbb{R}^2$ in which adjacent grid nodes have distance 1. For any tree spanning the grid points, there are adjacent grid nodes that are in distance $\Omega(k)$ in the tree.*

The chapter finally discusses state-machine implementations that lift the single-clock restriction to multiple clock sources in Section 2.7.

2.2 Revisiting State Machines

Let us revisit the implementation of a state machine in synchronous logic from Chapter 1. However, in contrast to the introduced Mealy machine, we will

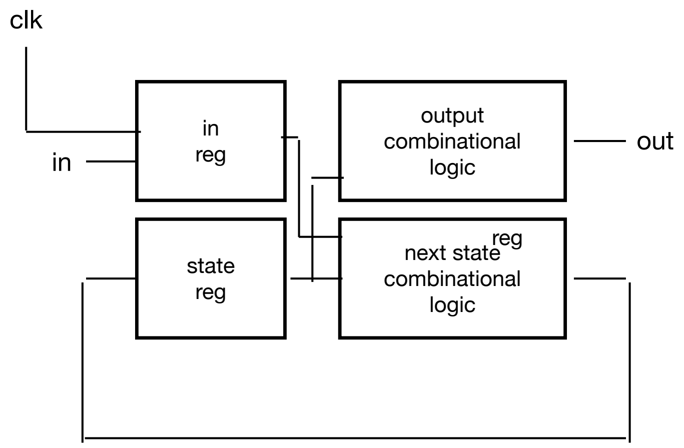

Figure 2.1
Implementation of a Moore state machine in the synchronous design paradigm.

discuss the implementation of a simpler *Moore machines* where the output function o only depends on the current state and not the presented input. We thus have $o : S \rightarrow \Lambda$. All other components of the state machine, including the transition function $t : S \times \Sigma \rightarrow S$ are as in the Mealy machine. Our choice for Moore machines has two reasons: (i) first they are simpler and thus well suited as a running example. (ii) Moore machines are typically preferred when implementing state machines as potentially unstable inputs do not directly drive outputs, leading to unstable outputs. If stable inputs can be guaranteed, Mealy implementations may lead to more efficient implementations, however. We leave a generalization of the following discussion to Mealy machines as an exercise to the reader.

Figure 2.1 shows an implementation within the widely adapted synchronous design paradigm. Its current state and previously captured inputs are stored in flip-flops–denoted as state and input registers in the figure. The next state transition function t as well as the output function o are implemented by combinational logic. In the following we will identify the state and its binary encoding, as well as assume that t and o are functions on binary tuples rather than the state space, for simplicity of notation.

On occurrence of an active (say, rising) clock transition on the clock signal CLK the result of applying the next state transition function t to the previous state $s \in S$ and the previously captured input $i \in \Lambda$ is written to the state register. Likewise the input register is updated to the current state of the input signals IN. The result $o(s)$ is then propagated to the output OUT, and $t(s, i)$ is propagated towards in inputs of the state flip-flops; ready for the next active clock transition to occur.

Clearly the above implementation has to meet certain timing constraints in order to work. For this purpose, let us take a closer look at the components used within the circuit.

2.2.1 Boolean Gate

Exemplary for Boolean gates let us consider a buffer element BUF. The Boolean function of a buffer element is the identity, i.e., $\textsc{Buf}(i) = i$ for $i \in \{0, 1\}$.

A physical implementation within standard CMOS logic encodes these logical values by two disjoint voltage ranges within the low voltage 0 V and the high voltage of $V_{\mathrm{dd}} > 0$. The value of V_{dd} varies depending on the used technology and power considerations and may even be modified during operation. When not specified otherwise, we will assume a normalized $V_{\mathrm{dd}} = 1$.

The Boolean value 0 of the buffer input or output is then encoded by the buffer input or output voltage being within $[0, V_{\mathrm{dd}}]$, and Boolean value 1 is encoded by its voltage being within $[V_{\mathrm{hi}}, V_{\mathrm{dd}}]$. From the assumption that both ranges are disjoint, $V_{\mathrm{lo}} < V_{\mathrm{hi}}$.

Stead-state behavior. Figure 2.2 shows two buffers BUF1 and BUF2 in series as well as signals a and b being the first buffer input and output, respectively.

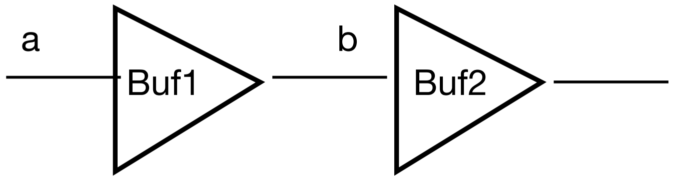

Figure 2.2
Two buffer elements in series with signals a and b shown.

Given the Boolean specification of a buffer element, as well as its encoding of Boolean values within a low and high voltage range, a feasible steady state behavior that fulfills these constraints is shown in Figure 2.3: low inputs are mapped to low outputs and high inputs to high outputs. While this S-shaped curve is typical, steady-state behavior of buffers may vary widely though, e.g., with different slopes in the middle of the curve between low and high values.

Dynamic behavior. By definition the steady-state behavior does not include any information on the timing of the gate. Before discussing how timing can be added to this picture, let us look at the timed behavior of a buffer gate.

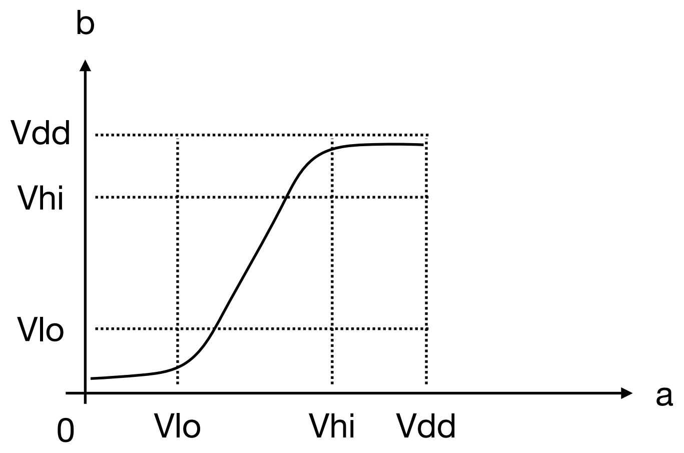

Figure 2.3
Steady-state behavior of a buffer.

Figure 2.4 shows a detailed analog Spice simulation of the two buffer setup from Figure 2.2. The figure show the signals over time during 40 ps. In the setup $V_{dd} = 0.8$ V as can be seen from the buffer signals reaching steady-state for high inputs at times around 20 ps. Additionally, red dots indicate when the signals cross $V_{dd}/2 = 0.4$ V.

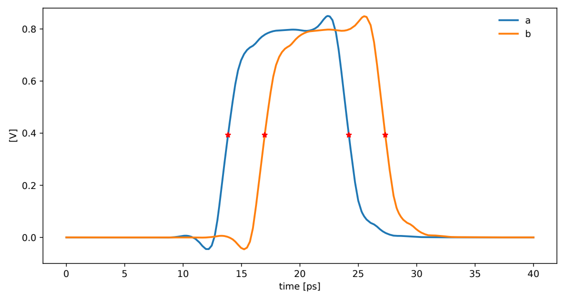

Figure 2.4
Dynamic behavior of a buffer.

A natural, simple timed model of a buffer gate is by adding a constant delay $d \geq 0$ between the time an input change occurs and the time the outputs settle at the newly induced steady-state. Inspecting Figure 2.4, one notes that the time the steady state is reached varies significantly depending on the precise definition: (i) the analog steady state may not even be reached in finite time or may depend on small fluctuations from environmental factors difficult to control, and (ii) the time until the Boolean steady-state depends on the V_{lo} and V_{hi}. Two conventions are typically used here: $V_{lo} = 0.3V_{dd}$ and $V_{hi} = 0.7V_{dd}$, or $V_{lo} = V_{hi} = 0.5V_{dd}$.[2] We will stick to the latter convention, here, and thus observe $d \approx 2.5$ ps from the simulation. One, however, already observes a small difference between the delay for low inputs and high inputs in the figure that can be more pronounced for other Boolean gates. This leaves us with one delay for each input and its corresponding steady-state output.

Environmental factors like temperature, supply voltage provided to the gate, as well as load and state of the successor gate, etc., further influence the delay of the gate. One thus, typically provides delay ranges rather instead of constant delays per input and its corresponding output.

Finally, if it cannot be assumed that the outputs have already reached the state-state when the input is applied, statefull delay models have to be considered. The simplest such, is a first order approximation, assuming that the current output charges or discharges by input changes according to first-order dynamics. For the input buffer with input a and output b we thus have,

$$\frac{db(t)}{dt} = -\frac{1}{\tau} \cdot (b(t) - \textsc{Buf}(a(t))) = \tau \cdot (b(t) - a(t)) \ , \tag{2.1}$$

where $\tau > 0$ is the time-constant of the first-order dynamics and we used the fact that $\textsc{Buf}$ is the identity function. The larger the difference of the current output $b(t)$ to the intended output $\textsc{Buf}(a(t))$, the stronger the output b will decrease. Solving this ordinary differential equation for $y(0) = y_0 \geq 0$ and an input switching to $x(t) = x \geq 0$ at time $t = 0$, gives

$$b(t) = x + e^{-\frac{1}{\tau} \cdot t}(y_0 - x) \ . \tag{2.2}$$

From the solution we see that the output b will exponentially diminish the initial difference between the current output y_0 and the intended output x (see the second term) and settle at the indented steady-state output x (first term).

From (2.2) one derives the delay as specified before. For example, given that the output resides at $y_0 = 0$, and the input switches to $x = 1$, we obtain

$$b(t) = 1 - e^{-\frac{1}{\tau} \cdot t} \ . \tag{2.3}$$

Setting it equal to $V_{dd}/2 = 0.5$, one gets

$$t = \tau \log_e(2) \approx \tau \cdot 0.69 \tag{2.4}$$

[2] Formally, we would have to choose to both values slightly apart from $0.5V_{dd}$.

as the delay for rising transitions from an output initially resting at 0.

E2.1 Derive the delay for falling transitions and compare it to the delay for rising transitions. You can assume that the output is initially resting at 1. How would you adapt the model to explain different delays for rising and falling transitions? Can you also explain delay ranges with different initializations of the output?

Equivalent circuit for wires. We have previously concentrated on gate delay and assumed that first-order dynamics is valid approximation for the voltage at a gate output. Let us take a closer look on wires in the following.

A wire segment, or interconnect, within an IC is a geometric piece of conducting material on a chip. As such it has a resistance R and a capacitance to all other conducting elements on the chip. For simplification, one typically assumes for delay considerations, that there is only a capacitance C to ground. A, sufficiently small, wire segment with input a and output b thus behaves approximately like the equivalent circuits depicted in Figure 2.5.

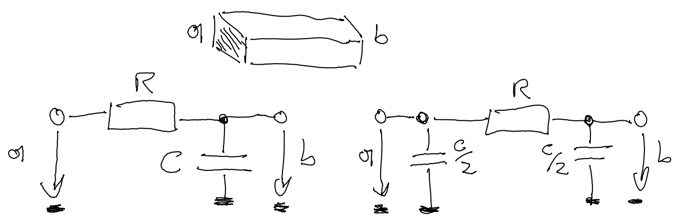

Figure 2.5
Dynamic behavior of a small wire segment. The left wire model is called L model in literature, the one on the right Π model.

Note that the circuit on the right in Figure 2.5 was simply obtained by splitting the capacitance C into two parts and adding one at the input and one at the output instead of both at the output. The obtained model is called the Π model as opposed to the one-sided model which is called the L model. Intuitively, this accounts better for symmetry of a wire segment where input and output should behave identically if switched. Indeed, the Π model outperforms the L model in terms of accuracy for a wire segment of the same length, in practice.

In general a gate drives other gates with their inputs connected to the driving output via a tree of wire segments. For the tree we thus obtain Π segments with their R and C parameters. Depending on the granularity of the segments, the same wiring may results in trees of different size. In Figure 2.6 a coarse tree is shown with only 3 segments.

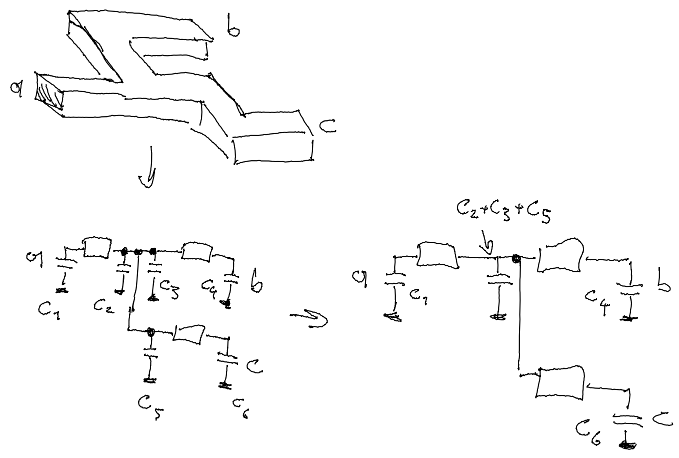

Figure 2.6
A tree of wire segments, each modeled by a Π model and its simplification.

The equivalent circuit may be immediately simplified by adding up all parallel capacitances without resistances inbetween into a single capacitance; see Figure 2.6. Solving for the voltage response at a leaf of this tree, given that the input of the tree makes a sudden rise to 1 or falls to 0, will give a higher-order solution in general. However, a natural question is to approximate the circuit by its dominant first order response. This approach is called the Elmore delay approximation. It states that if the output at leaf j is approximated by a first-order dynamics as in (2.3), then the term τ is obtained as follows: Let $0, i_1, i_2, \ldots, i_N$ be nodes in the tree along a path from the root 0 to leaf $i_N = j$. Let $R(\ell)$ be the resistance that is just upstream of node ℓ. Let $C_{\text{down}}(\ell)$ be the sum of all capacitance that are downstream of node ℓ. Then,

$$\tau = \sum_{\ell \in \{0, i_1, i_2, \ldots, i_N\}} R(\ell) C_{\text{down}}(\ell) \ .$$

In fact this behavior is also obtained by an equivalent circuit that comprises that of a single L model circuit with $RC = \tau$. One thus calls the Elmore delay model a lumped delay model, since it lumps all Rs and Cs of the tree into a single R and C.

E2.2 Bonus exercise: Difficult and must not be prepared for pre-reading (requires basic circuit knowledge). Show that an L model circuit behaves as (2.3) with $\tau = RC$.

Together with (2.4), the delay thus is roughly $\tau \cdot 0.69$.

E2.3 Not for pre-reading but for class: derive the Elmore delay for the tree in Figure 2.6.

2.2.2 Flip-flops

Chapter 1 has introduced flip-flops as state-holding circuit components that copy the data input D to the output Q at the occurrence of a rising transition at the clock input clk.

A physical implementation is necessarily timed and needs time to copy the input to the output. The delay is called the clk-to-Q delay $d > 0$. Additionally, correct copying of the input to the output requires the input to be stable during the process of being copied. This range is given by the setup time T_{setup} and the hold time T_{hold} with the requirement that the digital input signal is stable during

$$t + [-T_{\text{setup}}, T_{\text{hold}}] \ ,$$

where t is the time the active, rising clock transition occurs at the clock input. While typically positive, T_{setup} and T_{hold} may be negative in principle for some flip-flop implementations; the interval in (2.5) always has strictly positive length, however. Figure 2.7 depicts a scenario of a flip-flop copying its input with the delay annotated.

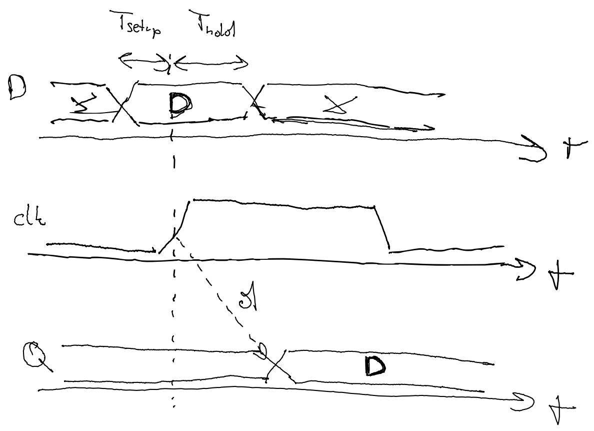

Figure 2.7
Flip-flop copying its input D to the output Q at the occurrence of a rising clock transition at clock input clk.

2.3 Clock Tree Synthesis Problem

We are now in the position to discuss a clocked Moore state machine implementation within a model with timing. We start by introducing clock trees.

2.3.1 Clock Tree

In a classical synchronous setting all flip-flops are driven by clock signals that are derived from a single clock source. The clock signal thus has to be distributed from the source to the clock inputs of the flip-flops. This is done by the so called clock distribution network; a circuit that is dedicated to the distribution of the clock signal. While such a circuit is typically built hierarchically and may comprise non-trivial circuitry in general, we will start our considerations with the simplest network: a directed tree whose components are interconnect, clock buffers, and inverters. Figure 2.8 depicts such a tree.

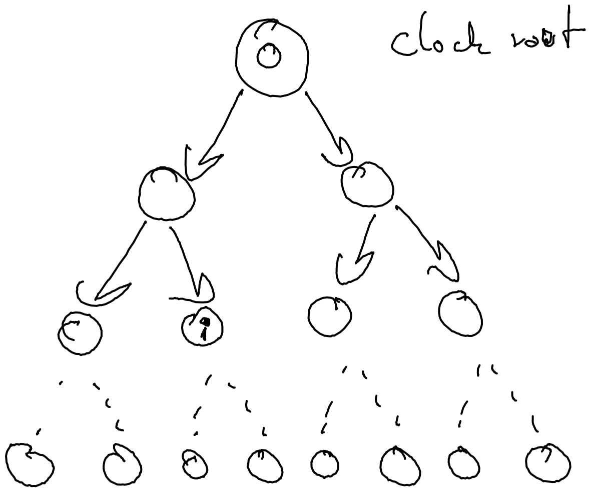

Figure 2.8
Clock tree. The root node 0 drives each downstream clock node $i \in \{1, 2, \ldots, N\}$ for $N \in \mathbb{N}$.

A node in the tree, called clock node, is from $\{0, 1, 2, \ldots, N\}$ with $N \in \mathbb{N}$. Clock node 0 is the clock source. Clock nodes i other than 0 represent signals in the clock distribution tree. The signal corresponding to clock node i is either directly the output of a clock buffer or an inverter or after being propagated along a wire segment. Edges in the clock tree are labeled by propagation delays; for simplicity, we assume constant delays that are identical for rising and falling transitions. Without loss of generality the clock tree is assumed to be binary: whenever a driver, drives multiple outputs that fork at the same geometric position, we choose an arbitrary binary splitting and delays of 0.

2.3.2 Source to Sink Constraints

We start by identifying constraints that arise from the timed behavior of the components. For that purpose, consider the path along a source flip-flop and a sink flip-flop, both of which are potentially identical. Figure 2.9 shows such a path for the synchronous implementation of a Moore state machine.

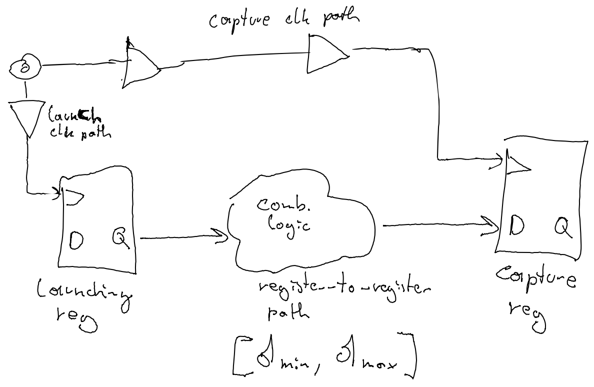

Figure 2.9
Source to sink constraints in a synchronous implementation of a Moore state machine.

The launching register ℓ is triggered via the launch clock path and the capture register c via the capture clock path. For the combinational logic path from register ℓ to register c, we done its range of delays with $d_{\text{cmb}}(\ell, c)$. The register-to-register path additionally includes the clk-to-Q delay of the launching register and its delay range is denoted by $d_{\text{r2r}}(\ell, c)$ for the path from register ℓ to register c.

Before stating the timing constraints that guarantee a correct operation of the synchronous implementation, we introduce some definitions. Let $t^k(i)$, with $k \geq 1$ and i being a node in the clock network, be the time of the k^{th} transition at node i. We then define:

- The k^{th} clock cycle length at the clock node i,

$$T^k_{\text{cyc}}(i) = t^{k+1}_{\text{clk}}(i) - t^k_{\text{clk}}(i) \ . \tag{2.5}$$

- The local skew $t^k_{\text{skew}}(i, j)$ of the k^{th} active clock transition from clock node i to j is defined as

$$t^k_{\text{skew}}(i, j) = t^k_{\text{clk}}(i) - t^k_{\text{clk}}(j) \ . \tag{2.6}$$

Further, let

$$t_{\text{skew}}(i, j) = \max_{k \geq 1} t^k_{\text{skew}}(i, j) \ . \tag{2.7}$$

- The insertion delay $t_{\text{ins}}(i)$ of the k^{th} active clock transition at clock node i is

$$t^k_{\text{ins}}(i) = t^k(i) - t^k(0) \ . \tag{2.8}$$

Correct operation of the synchronous circuit is then guaranteed if and only if all of the following constraints hold: For all register-to-register paths from a launching register clocked by clock node ℓ to a capture register clocked by clock node c, and all rising clock transitions $k \geq 1$:

- The *setup constraint* has to hold, with

$$t^k(\ell) + d_{\text{r2r}}(\ell, c) \leq t^{k+1}(c) - T_{\text{setup}} \ .$$

- The *hold constraint* has to hold, with

$$t^k(\ell) + d_{\text{r2r}}(\ell, c) \geq t^k(c) + T_{\text{hold}} \ .$$

We may rewrite both constraints in terms for the clock cycle, the local skew, and the insertion delay, obtaining:

- For the *setup constraint*:

$$d_{\text{r2r}}(\ell, c) + t^k_{\text{skew}}(\ell, c) + T_{\text{setup}} \leq T^k_{\text{cyc}}(c) \ .$$

- For the *hold constraint*:

$$d_{\text{r2r}}(\ell, c) + t^k_{\text{skew}}(\ell, c) \geq T_{\text{hold}} \ .$$

E2.4 Show that the above rewriting is correct.

From the two constraints, we observe that, in general, a small absolute skew is preferred: Larger absolute skews between ℓ and c may help to fulfill one constraint, but make it more difficult to fulfill the other. More importantly, however, larger absolute skews are prohibitive for the case c acts also as a launching register, even, perhaps for register ℓ. One approach, while more coarse than trying to fulfill the system of inequalities, is thus to minimize the local skew between all clock nodes driving registers.

A clock source and a clock tree that together fulfill the above inequalities for a state machine implementation are called correct for this implementation. Indeed the following holds:

Theorem 2.1. *If a clock source and a clock tree are correct for a state machine implementation, then the implementation behaves as the untimed state machine from Chapter 1.*

E2.5 Argue why Theorem 2.1 holds.

2.3.3 Optimization Problem

While a correct clock source and tree guarantees correct circuit behavior, one is usually interested in optimal solutions with respect to some performance measure. Major measures are power, local skew, injection delay, and of course, frequency of the clock source. While the motivation to minimize power and frequency are clear, the goal to minimize local skew stems from the simplifying assumption that a smaller local skew makes it easier to fulfill the constraints we discussed in the previous section. The motivation to minimize the insertion delay has several reasons, one of them being the worst-case delay jitter, i.e., differences in delay, along th path from the clock root to a clock node. Another reason is the latency from changing the clock node frequency to it actually happening at the clock leaves.

The clock synthesis problem has thus been formulated as different optimization problems, depending on which of these properties are bounded and which ones to optimize.

A central measure is the power consumed by the clock network. For a periodic digital signal with frequency f occurring at a gate, we have

$$\begin{aligned} P &= P_{\text{dynamic}} + P_{\text{static}} && (2.9)\\ &= \left(P_{\text{switching}} + P_{\text{short-circuit}}\right) + P_{\text{static}} && (2.10)\\ &\approx CV_{\text{dd}}^2 f + \alpha + 0\ , && (2.11) \end{aligned}$$

where C is the capacitance that the gate drives, and α is a constant that accounts for the short-circuit power per gate. The short-circuit component is mainly dependent on the switching delay of the clock signal and thus its slew rate. Typically the gate slew rate is constrained during clock tree synthesis and the component thus upper bounded by some $\alpha \geq 0$. The static power, determined by the leakage, is often neglected since it is small in comparison to the dynamic term for current technologies. This may change depending on the used technology, however.

We may again use the Elmore approximation of a lumped L model to arrive at a single R and C. Following the Elmore approximation we proceed as follows: The *capacitance* $C_{down}(g)$ seen from a gate g is composed of the output gate capacitance of the gate, its output wires, and the input capacitances of the downstream gates. Roughly, the output and input gate capacitances of a gate g are proportional to the gate width $w(g)$, and the wire capacitance of a segment s is proportional to its length $l(s)$ and its cross sectional area $A(s)$. Thus, for a gate g with downstream gates g' in G_g and downstream wire segments s in S_g,

$$C_{\text{down}}(g) \approx \sum_{g' \in G_g} w(g') \cdot C_{\text{gate}} + \sum_{s \in S_g} A(s) \cdot l(s) \cdot C_{\text{wire}} \quad . \tag{2.12}$$

The *propagation delay* from node i to a child node j is then obtained from first computing the lumped RC derived from the Elmore approximation of the RC-tree including the gate capacitances. Using, (2.4) one then finally obtains the propagation delay.

Clock tree optimization. We are now in the position to state the optimization problems to obtain optimal clock tree designs. The simplest formulation uses a fixed local skew budget $t_{\text{max,skew}} > 0$ and optimizes for power P.

Definition 2.2 (Clock synthesis, local skew budget)**.** *Given* $t_{\max,skew}$*, find an optimal solution to the problem:*

$$\begin{aligned} &\min P\,, \textit{ such that} \\ &\forall j, j : t_{skew}(i,j) \leq t_{\max,skew} \quad . \end{aligned}$$

As previously discussed, minimizing the skew, or maintaining it within a skew budget as in the previous optimization problem, is a simplification. Although often followed, a more refined optimization problem is:

Definition 2.3 (Clock synthesis, skew constraints)**.** *Find an optimal solution to the problem:*

$$\begin{aligned} &\min P\,, \textit{ such that} \\ &\forall i, j : \textit{ the setup and hold constraints for } i, j \textit{ are fulfilled} \quad . \end{aligned}$$

E2.6 The insertion delay does not appear as a minimization target in this optimization problem. Or does it? And if so, how? Think of delay variations over time.

E2.7 For class, not pre-reading: Can you come up with a simple heuristic that creates clock trees that has lock skew 0 (given that there are no delay variation per gate and wire)? Think of a binary, bottom-up, merging approach. How does the Elmore delay help here?

2.4 Improvements

MF: NOTE. This section is in progress and covers two advanced techniques of how to obtain higher frequencies for synchronous circuits. It will not be covered in the course (or only if you wish to briefly, but no pre-reading is required).

2.4.1 Retiming

MF: tbd

2.4.2 Clock Skew Scheduling

MF: tbd

2.5 Clock Sources

We have assumed a root clock source with perfect clock period $T_{\mathrm{cyc}} > 0$. In the following, we summarize implementations of such clock sources and limitations and extensions to the clock source model.

2.5.1 Uncontrolled Clock Source

Typically the clock source driving the clock tree is a Phase locked loop (PLL) driven by a quartz oscillator. We will discuss PLLs in greater detail in Chapter **??**. At this point we just view the quartz oscillator together with the PLL as a single clock source.

Clearly, for physical implementations, the perfect period of $T_{\mathrm{cyc}} > 0$ is oversimplified. Noise on clock transitions exists at different timescales, induced by environmental factors such as temperature, power supply variations, mechanic forces, wear-out, etc. While detailed models on clock noise exist, characterizing noise according to the spectral domain it belongs to, we will stick to a simplified, but conservative, bound on clock transitions. We assume, that for all positive clock transitions $k \geq 1$, and all $\ell \geq 1$,

$$\ell \cdot T_{\mathrm{cyc}} \cdot (1 - \rho) \leq t^{k+\ell}(0) - t^{k+1}(0) \leq \ell \cdot T_{\mathrm{cyc}} \cdot (1 + \rho) \ ,$$

where $\rho \geq 0$ is the *two-sided clock drift*. To simplify the analysis of such systems, one sometimes uses the *one-sided clock drift* representation,

$$\ell \cdot T_{\mathrm{cyc}} \leq t^{k+\ell}(0) - t^{k+1}(0) \leq \ell \cdot T_{\mathrm{cyc}} \cdot (1 + \rho') \ .$$

Both are interchangeable. Given a two-sided drift ρ', one obtains the one-sided drift as $\rho' = 1 - \frac{1+\rho}{1-\rho}$.

While the two-sided drift ρ appears as non-determinism, and thus potential instability of the clock signal, the fact that clock periods change with environmental conditions may also have beneficial effects. Indeed, e.g., the work by Cortadella [**?**] observed that the fact that the frequency of a ring oscillator, formed by a ring of inverters as shown in Figure 2.10, decreases with lower voltages can be used to automatically tune the clock source frequency in accordance to power supply variations. We will revisit such control circuits in Chapter **??**.

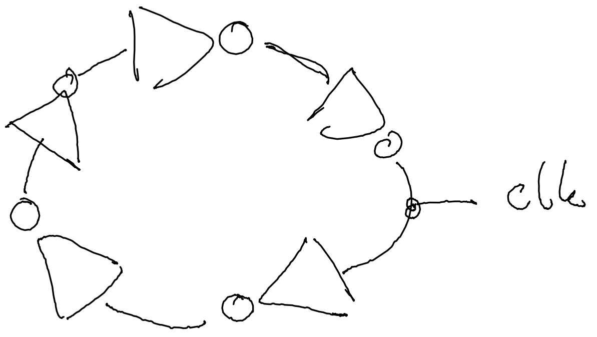

Figure 2.10
Ring oscillator implemented by an odd number of inverters driving a clock signals clk.

2.5.2 Controlled Clock Source

Adapting the frequency, including stopping the clock completely, has become an integral part of chips. While the clock signal for some clock subtrees may be masked with And-gate based solutions, a technique called *clock gating*, adapting the frequency requires more complex circuitry. We will discuss such circuits in Chapters **??**, **??**, and **??**.

2.6 Fundamental Limitations

So far we have discussed how to obtain valid and optimized implementations of clock trees. But how good can we do in principle with clock trees? Are there limitations that no technique whatsoever can break?

We start this discussion by observing that a typical chip is 2 dimensional and a clock tree needs to provide the clock signal to a roughly quadratic area, where we can expect that at the very least physically close-by parts of the chip need the signals provided to them to be well-synchronized. The following result shows that at least for some nodes, a tree must fail to do so. The result is due to Fisher and Kung [5] where it is stated without proof; a proof can be found in Boksberger *et al.* [1].

Theorem 2.4. *For $k \in \mathbb{N}$, consider a $k \times k$ grid in $\mathbb{R}^2$ in which adjacent grid nodes have distance* 1*. For any tree spanning the grid points, there are adjacent grid nodes that are in distance $\Omega(k)$ in the tree.*

Proof. Observe that if a tree node v has degree larger than 3, we can reduce its degree by inserting an additional node arbitrarily close to it and attaching 2 or more of the children of v to the new node instead, like we did when introducing clock trees. This changes tree distances by an arbitrarily small amount. Accordingly, we can w.l.o.g. assume that the tree is binary.

In any tree T of maximum degree 3 (of at least 4 nodes), there is some edge so that the two components resulting from removing this edge have size at least $(n-1)/3 \in \Omega(n)$. To see this, pick an arbitrary edge, delete it, and look at the resulting components T_1 and T_2. If $|T_1|, |T_2| \geq (n-1)/3$, we're done. Otherwise, assume w.l.o.g. that $|T_1| < (n-1)/3$, i.e., $|T_2| \geq n-(n-1)/3 = 2(n-1)/3+1$. Let w be the endpoint of the deleted edge that lies in T_2. Deleting w from T_2 results in (at most) two components of T_2, as w has degree 3 (and thus at most 2 in T_2). One of these components must have size at least $(|T_2|-1)/2 \geq (n-1)/3$. Consider the edge connecting w to this component and delete it from T, resulting in components T_1' and T_2'; let's say $w \in T_1'$. By the previous considerations, we have that $|T_1'| > |T_1|$, while $|T_2'| \geq (n-1)/3$. Now either $|T_1'| \geq (n-1)/3$ and we're done, or we can repeat the argument, resulting in an edge for which one component is even larger than T_1' and the other remains of size at least $(n-1)/3$. Thus, repeating this argument inductively, we must eventually reach an edge satisfying the claim.

From the above claim, we can infer that we can partition the nodes into two sets such that (i) each set contains $\Omega(k^2)$ nodes and (ii) each set induces a subtree. We call a node a *boundary node* if it has a neighbor in the other set. From (i) we can infer that the boundary must contain nodes in distance $\Omega(k)$ from each other, as any area of size $\Omega(k^2)$ cannot be confined within a square or side length $o(k)$ (note that this argument is resilient to the fact that nodes at the boundary of the grid can also serve as boundary of the respective area). Fix two such nodes v and w in the same subtree. As they are in distance $\Omega(k)$, the path in the subtree connecting them is of that length. This in turn means that at least one of them is in distance $\Omega(k)$ of the root of its subtree. Finally, we conclude that this node is in distance $\Omega(k)$ from its neighbor(s) in the other set within the tree. □

E2.8 The statement for the grid shows that there's a problem no matter how gates are placed on the chip. Can you see why? Could you cast this into a formal statement with proof?

- If uncertainties are proportional to the length of a link, we can immediately conclude that the worst-case skew between adjacent nodes is $\Omega(uk)$, where u is the uncertainty of a link of unit length, cf. **??**.
- The above bound is tight up to constants, which is shown by an H-tree. For $k+1$ (the "width" of the grid) being a power of 2, an H-tree is constructed recursively as follows. Place the root in the center of the grid. Then connect it to two children by going $k/2$ to the right and left, respectively. Each of these children also has two children, which are in distance $k/4$ going up and down, respectively. The four nodes in depth two of the tree are now exactly in the center of four disjoint subgrids of $k/2 \times k/2$ nodes, and the construction is applied recursively for $\log k$ steps. In the end, each grid point with both x- and y-index being odd (or even, depending on indexation) is occupied by a leaf.**CL**: Depiction of an H-tree.
- The construction from the lemma actually shows that there must be $\Omega(k)$ adjacent grid nodes that are in distance $\Omega(k)$ in the tree. More generally, one can show that $\Omega(2^i k)$ adjacent grid nodes are in distance $\Omega(k/2^i)$ in the tree.
- An H-tree matches this bound, too: Cutting the square in half horizontally and vertically, one gets four subsquares, each of which hosts a smaller H-tree. Where we cut the grid, we have $2k$ adjacent node pairs that end up being in distance (almost) $2k$ in the tree. All other adjacent node pairs are in the same of one of the four sub-H-trees, so they are by factor 2 closer to each other.

2.7 Beyond a Single Clock

For some applications a single clock source, driving a clock tree with balanced nodes is too costly in terms of design effort or required resources, or simply not possible because of the limitations we saw in the last section. In this case, a possible strategy is to give up on a single clock source and replicate the clock source. The replication may be analytically rather than physically, though: the independent clock trees may be fed by the same clock source, but be two subtrees that are not balanced with respect to each other. Figure 2.11 shows the step from a single clock domain to two clock domains, each running its own finite state machine, but state machine FSM_1 communicating its output to state machine FSM_2.

While the figure shows one-directional communication for simplicity, the following principles are easily translated to two-direction communication. First

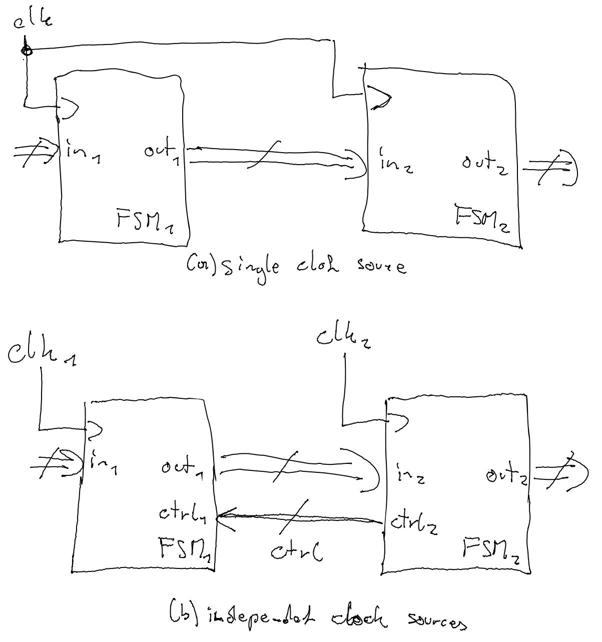

Figure 2.11
Communicating state machines: (a) within a single clock domain, (b) within two different clock domains.

observe that unrelated clocks, potentially of different nominal frequencies, necessitates a feedback mechanism for flow control from FSM_2 back to FSM_1 in order to acknowledge when the data has been readily processed. Clearly, though, certain scenarios of two clock domains, may not require such feedback control.

E2.9 When don't you need feedback? Think of an example. What if clocks are derived from the same source but with unbalanced / unknown skew?

E2.10 For class and not for pre-reading: What flow control signals would you use? Explain and draw a timing diagram of a typical communication from FSM_1 to FSM_2. What are critical scenarios? Think of setup/hold time constraints.

Flow control is not the sole problem that has to be solved, though: remember the setup and hold time constraints that had to hold for flip-flops. We will discuss this in the following Chapter **??**.

Bibliography

[1] Boksberger, Philipp, Fabian Kuhn, and Roger Wattenhofer. 203. On the Approximation of the Minimum Maximum Stretch Tree Problem, Technical Report 409, ETH Zurich.

[2] Even, Guy. 2006. On teaching fast adder designs: Revisiting Ladner & Fischer. In *Theoretical computer science*, 313–347. Springer.

[3] Even, Guy, and Moti Medina. 2012. *Digital logic design: a rigorous approach*. Cambridge University Press. http://hyde.eng.tau.ac.il/Even-Medina/index.html.

[4] Even, Shimon. 2011. *Graph algorithms*. Cambridge University Press.

[5] Fisher, Allan L., and Hsiang-Tsung Kung. 1985. Synchronizing Large VLSI Processor Arrays. *IEEE Transactions on Computers* C-34 (8): 734–740.

[6] Mealy, George H. 1955. A method for synthesizing sequential circuits. *The Bell System Technical Journal* 34 (5): 1045–1079.