Metastability

1 What is Metastability

Consider the hilly landscape shown in Figure 1. In the diagram x is the position in meters, and $h(x)$ the height in meters at position x. For simplicity we assumed a periodic landscape with $h(x) = \cos(x)$ that is symmetric around $x = 0$.

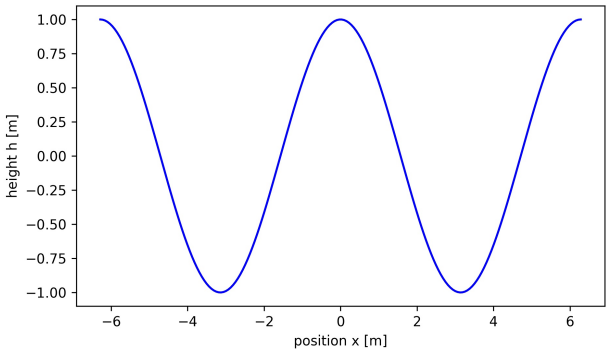

Figure 1: Landscape with valleys and hills.

We may now position a ball[1] at any position x_0 within this landscape and observe its trajectory over time. The result of this thought experiment for $x_0 = 2$ is shown in Figure 2.

As expected, the ball/particle follows a damped oscillation, approaching the right valley's center at π. Placing the particle at a higher initial position, but this time at the other side of the central hill, say at $x_0 = -1$, we observe a more accentuated oscillation in Figure 3.

Again the particle quickly approaches a valley's center; this time the left valley. Interestingly, with the naked eye, at time 25 s the particle seems to have almost come to rest in the valley like for $x_0 = 2$; the initial position where we let it go did not play a considerable role after some seconds.

This game can be played with several positions x_0 within $(-2\pi, 2\pi)$ with similar results: the particle seems to approach one of the valley centers rather quickly, and the concrete valley side depends on whether we position it left or

[1] Our ball will be a particle of negligible size in comparison to the landscape.

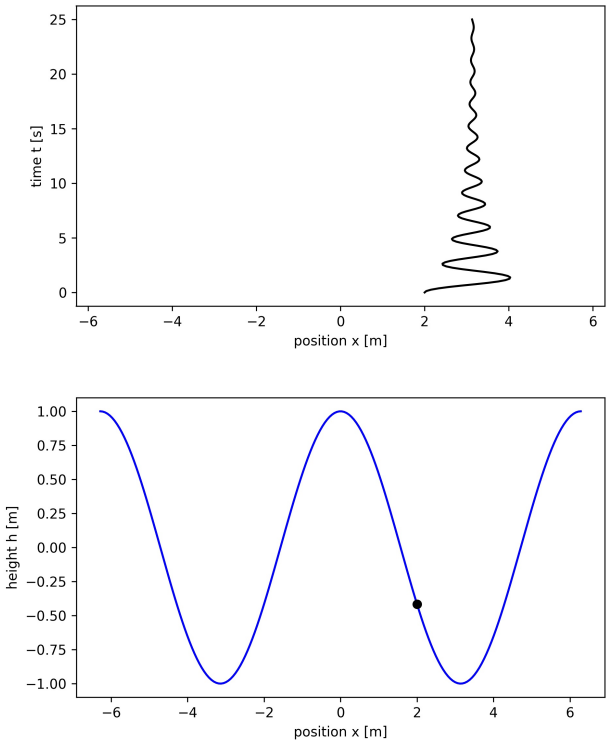

Figure 2: Particle moving in the landscape if initially positioned at $x_0 = 2$. At the bottom the landscape with the particle's initial position (in black) is shown. At the top the trajectory over the next 25 s is shown.

right to $x = 0$. There is one exception to this behavior: Figure 4 shows the case where the particle is placed at the top of the central hill at $x_0 = 0$. It seems to remain there forever.

While this behavior coincides with our physical intuition, our practical experience tells us that we would expect the particle to eventually fall to any side. Let us take a closer look in the following.

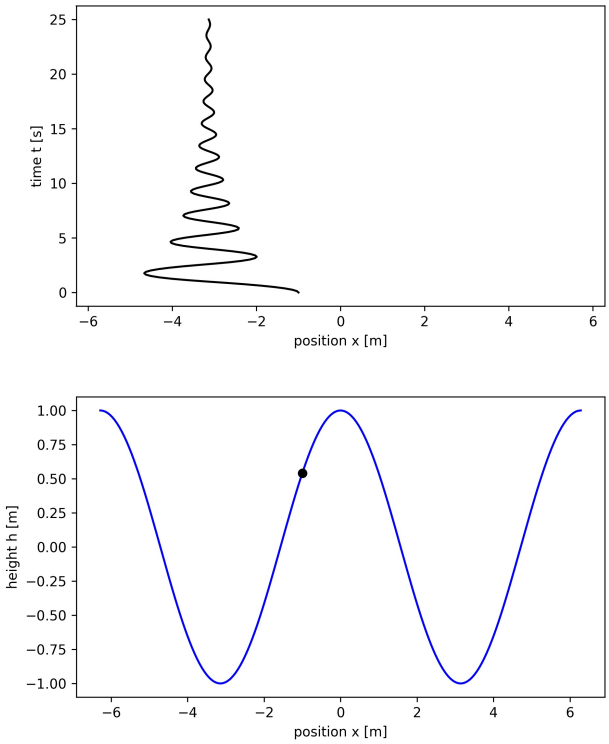

Figure 3: Particle moving in the landscape if initially positioned at $x_0 = -1$.

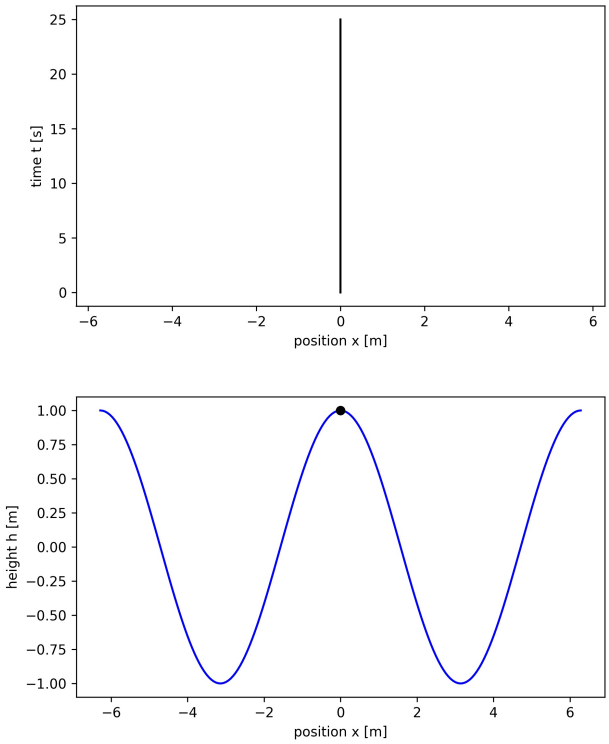

Figure 4: Particle not moving if initially positioned at $x_0 = 0$.

The initial position. In practical situations, one may arguably not be able to exclude all external forces, the tip of the hill will not be a single point, and static friction forces may play in our favor, but for the sake of the thought experiment let us exclude all these effects and check what happens if we are not able to perfectly place the particle at $x = 0$. Figure 5 shows the case where we position it slightly to the right at $x_0 = 10^{-13}$.

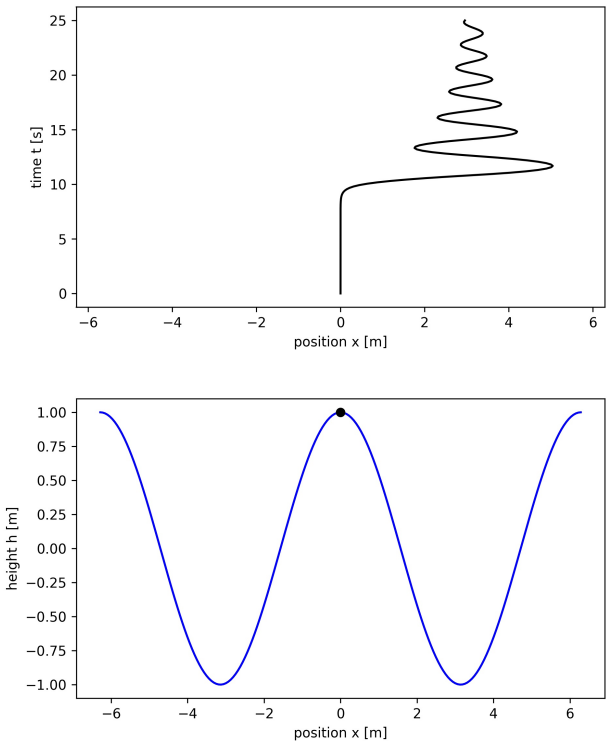

Figure 5: Particle moving delayed if initially positioned at $x_0 = 10^{-13}$.

Interestingly, the particle seems to reside at the initial position for around 10 s until when it rolls down the right hill. Also, we observe that at time 25 s it did not come to rest in the right valley's center as in the previous scenarios. A word of warning is due at this point: The figures were obtained by numerical integration with 64bit floats. So the trajectory may be off. However, one can indeed show that for all initial positions within $(-2\pi, 2\pi)$, the particle's position approaches $-\pi$ for strictly negative initial positions, π for strictly positive initial positions, and remains at 0 forever if started at 0. There are two other initial positions that result in the particle not moving over time: the valley centers $-\pi$ and π.

Before going into details, we introduce some notation. One can model the

dynamics of this system as a state $\mathbf{x}$ in $\mathbb{R}^n$ that evolves over time. To do so, we set

$$\frac{d}{dt}\mathbf{x} = f(\mathbf{x}) \ ,$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$.

Example 1. *In our case of the particle within the landscape, the dimension of the state space is $n = 2$ with $\mathbf{x} = (x, \dot{x})$, and we have*

$$\begin{aligned}\frac{d}{dt}x &= \dot{x} \\ \frac{d}{dt}\dot{x} &= -g\sin(\alpha(x))\cos(\alpha(x)) - \gamma\dot{x} \ ,\end{aligned}$$

where $\alpha(x) = \arctan(-\sin(x))$, the velocity dependent damping factor $\gamma = 0.3\,s^{-1}$, and gravitational acceleration $g = 9.809\,m\,s^{-2}$ as measured in Paris.

Definition 1 (Fixed point)**.** *A state $\mathbf{x} \in \mathbb{R}^n$ is a fixed point if $\frac{d}{dt}\mathbf{x} = \mathbf{0}$.*

Exercise 1. *Show that there are three fixed points whose first component is within $(-2\pi, 2\pi)$ and whose second component is 0.*

In summary, we may say that:

- There are three initial positions within $(-2\pi, 2\pi)$ that result in the particle not moving, i.e., $\frac{dx}{dt} = 0$: the two valley centers $-\pi$ and π, and the hill center 0.
- For all initial x_0 in $(-2\pi, 0)$, we have $x(t) \to -\pi$ for $t \to \infty$. For all initial x_0 in $(0, \pi)$, it is $x(t) \to \pi$ for $t \to \infty$. Both initial points are *attractors*. The regions $(-2\pi, 0)$ and $(0, 2\pi)$ are *basins of attraction* for these attractors: started within them, $x(t)$ will converge to the respective attractor.
- Initial value 0 is also an attractor, but its basin of attraction is the set $\{0\}$, only. Any small perturbation to this initial position will result $x(t)$ to converge to a different attractor. We thus call $-\pi$ and π *stable*, and 0 *metastable*.

Formally, we distinguish between stable and metastable, respectively, unstable fixed point as follows.

Definition 2 (Stable fixed point)**.** *A fixed point $\mathbf{x}'$ is* stable *if for all $\varepsilon > 0$, there exists a $\delta > 0$, such that,*

$$\begin{aligned}||\mathbf{x}(0) - \mathbf{x}'|| \leq \delta \Rightarrow \\ \forall t \geq 0 \ : ||\mathbf{x}(t) - \mathbf{x}'|| \leq \varepsilon \ .\end{aligned}$$

It is unstable *or* metastable*, otherwise.*

The definition requires a norm $||\cdot||$ on $\mathbb{R}^n$. Typically the Euclidean (2-norm) $||(x_1, x_2, \ldots, x_n)||_2 = \sqrt{\sum_{i=1}^n x_i^2}$ is used, if not stated otherwise. With this definition, stability requires more than we have discussed: it must tolerate a perturbation in all components and a combination thereof restricted only by the Euclidean distance to the fixed point.

Exercise 2. *Show that* $\mathbf{x} = (0, 0)$ *is a metastable fixed point in our setting.*

The time to resolve. Let us come back to the almost stable behavior at $x_0 = 10^{-13}$. From Figure 5 one may deduce that the particle resides steadily at x_0 until it diverges at around 10 s. For that purpose let us zoom into the first 5 s at a smaller scale of 10^{-6} m. The result is shown in Figure 6

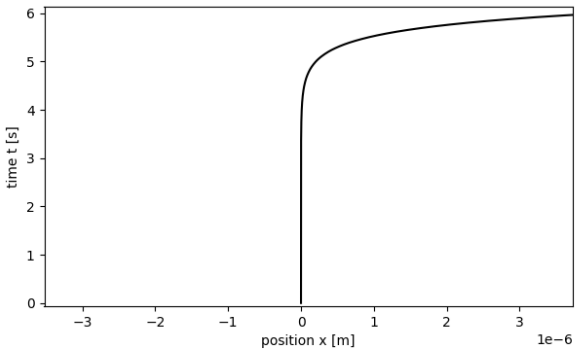

Figure 6: Particle starting at $x_0 = 10^{-13}$. Zoomed in view.

One observes a strong divergence to the right: the particle is not at all at rest, but diverges significantly. It rather takes another 5 s to observe this behavior at the meter scale.

We may run another simulation to see how long it takes for $x(t)$ to first cross 0.1 if we vary x_0 within $(0, 0.001)$, i.e., on the right side of the central hill near the metastable point at 0. The result is shown in Figure 7.

We observe a large increase of times when approaching the metastable point at $x = 0$. In fact one can show that the times approach ∞ as $x \to 0$; and for $x = 0$ the crossing time does not exist at all. By the symmetry of our setting, the same curve would be observed for small negative initial positions $x \to 0_-$; but this time for the crossing of $x = -0.1$. We will discuss this behavior in the following.

2 Metastability in Hardware

The picture of the landscape and the particle may seem far fetched for a book on hardware design at the first glance. We will, however, show that scenarios like this occur thousands of times on a single chip.

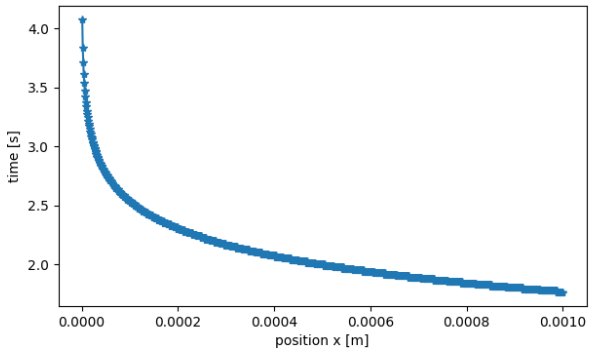

Figure 7: Time until $x(t)$ crosses 0.1 for the first time for initial positions from within $(0, 0.001)$.

For that purpose consider the Moore state machine from Figure **??** in Chapter **??**. Further assume that it has a single input signal $i(t)$ that is periodically sampled at the rising clock transition of clock clk. As a thought experiment, we will zoom into a single rising clock transition that occurs at, say, time 0, and observe the output Q of the flip-flop, depending on the input i driving the data input port D of the flip-flop. Figure 8 depicts this setup.

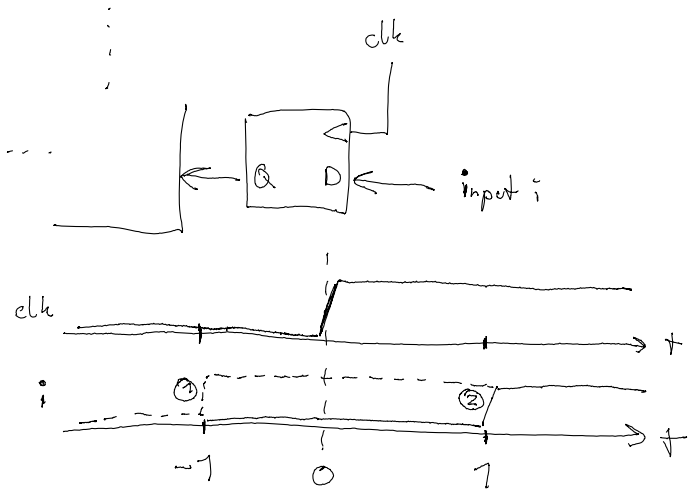

Figure 8: Input signal sampled for further processing by the synchronous state machine.

Let us assume that the input $i(t)$ is a step function at varying times x, i.e., it is 0 before time x, and 1 afterwards. Assume that setup/hold times are 0.1 ns. Now consider the following two cases:

1. Case $x = 1\,\text{ns}$. If so, the flip-flop will sample a 0 at its input and thus output a 0.

2. Case $x = -1\,\text{ns}$. If so, the flip-flop will sample a 1 at its input and thus output a 1.

The outcome is not far from the landscape: depending on the initial x, the flip-flop output will settle at 0 or 1; given that initial values respect the setup/hold window and are not within $[-0.1, 0.1]$. This is much like the two valley centers that are attractors in the landscape scenario.

What happens within this region? Interestingly, one also observes the same qualitative behavior as in the landscape for initial values near 0. If we plot the times until the flip-flop output first crosses 0.1, and in case x is strictly positive, we will likely obtain a behavior as in Figure 7.

Indeed, Figure 9 shows simulations for such times for a flip-flip in 15 nm technology.

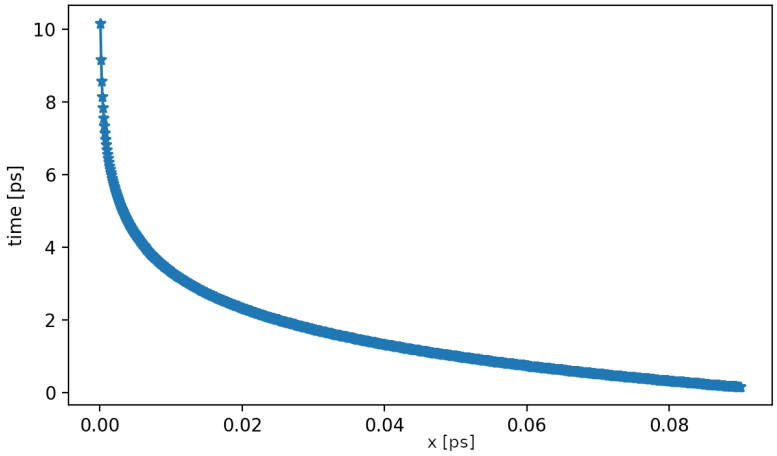

Figure 9: [TODO: redo this figure with real simulations of a gate. The current figure is a placeholder.] Resolution times for the flip-flop output u_2 and a threshold of 0.1 V for different initial values of x.

We will later see that an analogous behavior is obtained in a simplified model of the fip-flop.

In practical realizations of latches and flip flops the output of the actual storage loop (formed by the cross-coupled inverters) is not directly provided to the cell output. In order to protect the storage element from being flipped by coupling effects through the connected cells and interconnect, it is buffered by an inverter whose input is connected to the storage loop and whose output then supplies the cell output. In case of metastability this inverter will receive an

intermediate voltage (a voltage level somewhere in between a well defined HI and a well defined LO) and might convey that to its own output. This will happen when the inverter's threshold matches the voltage level of the metastable storage loop output. However, if the inverter's threshold is slightly higher (aka *high threshold inverter*) then the intermediate input voltage is below that threshold and will be regarded as a LO. Figure 10 (a-d) shows the possible outcomes of that: If the previous output was HI, then upon moving to metastability the output will definitely cross the high threshold, producing a HI output at the inverter output (mind the inversion!) immediately. When the metastability later resolves back to HI (Figure 10(a)), we see another threshold crossing on the way back. This will make the inverter output go back to LO – we have experienced a *glitch*. However, in the alternative case that the metastability resolves to LO (case (b)), no further threshold crossing occurs and it looks like the inverter's initial output transition had made the right guess. This case is called *early transition*. This is the most desirable case, as, from the view of the inverter output, it looks like no metastability ever happened.

For the remaining two cases we assume that the previous output had been LO. Then, on the way to the metastable state, no threshold crossing occurs yet. For a metastability resolving back to LO (case (c)) no threshold crossing occurs either, so the inverter output does not switch at all (*no transition*). This is another desired behavior, as it is perfectly fine to see no output reaction for an input sequence of LO followed by LO. Should, however, the metastable state resolve to HI (case (d)), then the threshold is crossed, but too late, namely only when metastability is resolving. This case is called *late transition*.

Alternatively, a low-threshold inverter can be used. As illustrated in Figure 10 on the right (cases (e) to (h)), we observe the same effects, albeit in different scenarios. So while the cases of no transition and early transition do not cause any negative effects for the subsequent logic, the glitch and the late transition are dangerous and hence undesired.

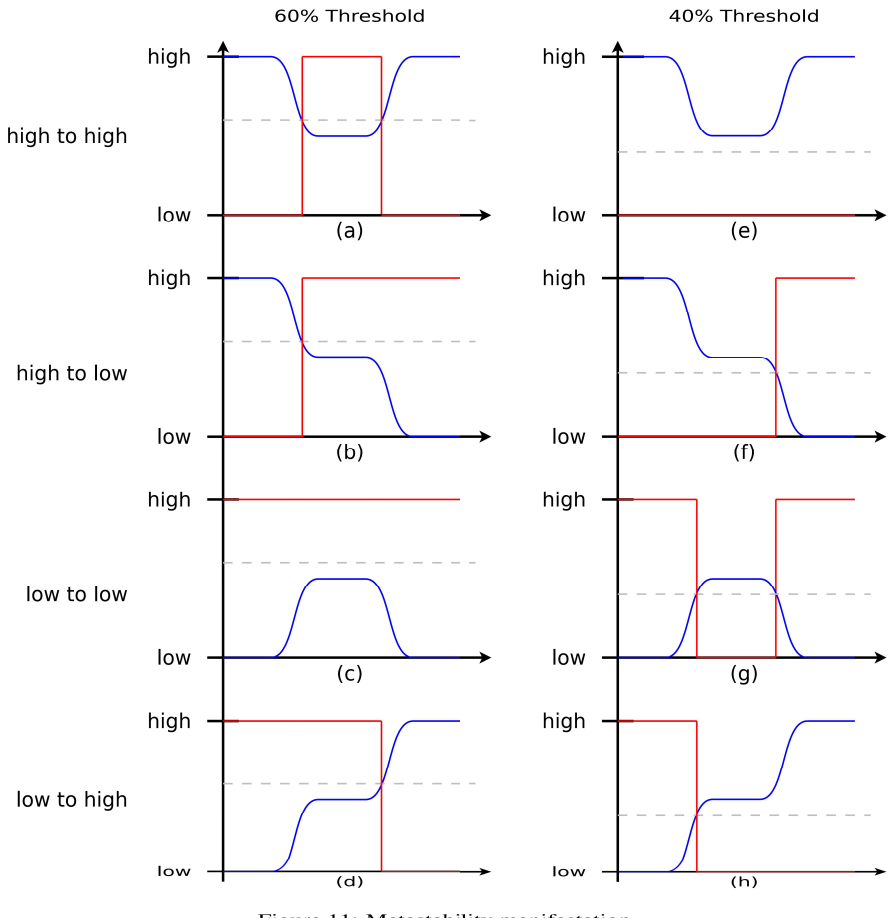

Figure 10: Manifestations of metastability after a high- or low-threshold inverter

3 Why is it inevitable?

One may argue that the fact that there is a metastable fixed point between the two stable fixed point may be specific to our setting of the landscape. We will next show that this is indeed a more general phenomenon. The precise shape of the hill in the middle of the two values is not needed to show such phenomena. But the result is also not specific to classical mechanics. It holds as well for electronic systems that are described by ODEs under only mild assumptions. We start this argument with a result in topology:

Theorem 1. *Let A and B be two topological spaces and $f : A \to B$ a continuous function. If $X \subseteq A$ is connected, then so is $f(X) \subseteq B$.*

For the landscape we choose $A = B$ to be the Euclidean topology on $\mathbb{R}$. The topology A will be the initial position x of the particle, and the topology B the position the particle will converge to. Accordingly, for a $T \geq 0$, we choose f_T to be the function that maps the initial position to the position of the particle at time T. By definition, f_0 is the identity. We further choose $X = (-2\pi.2\pi)$ as the subset of initial positions that we are interested in.

First, observe, that X is connected. Now choose an arbitrary $T \geq 0$, and let us take a closer look at function f_T. First, we may simplify the system's ODE to

$$\begin{aligned} \frac{d}{dt}x &= h_1(x, \dot{x}) = \dot{x} \\ \frac{d}{dt}\dot{x} &= h_2(x, \dot{x}) = \frac{g\sin(x)}{\sin^2(x)+1} - \gamma\dot{x} \ . \end{aligned}$$

Both functions h_1 and h_2 are Lipschitz on $\mathbb{R}^2$. For an initial value $\mathbf{x}(0) = (x_0, 0)$, with $x_0 \in (-2\pi, 2\pi)$, let us denote the corresponding solution $\mathbf{x}(t) = (x(t), \dot{x}(t))$ by $(x(t; x_0), \dot{x}(t; x_0))$ to highlight the dependency on x_0. It can be shown that from the fact that h_1 and h_2 are Lipschitz, the solution $(x(t; x_0), \dot{x}(t; x_0))$ is continuous in x_0; see, e.g., [**?**]. This finally implies that f_T as defined above is continuous.

We may now apply Theorem 1 and deduce that $f_T(X)$ is connected.

To see the concrete implications for the particle in the landscape, choose $T = 25\,\mathrm{s}$. We observed that for an initial position $x_0 = 2$, $f_T(x_0)$ was close to π; the center of the right valley. Also for an initial position $x_0 = -1$, $f_T(x_0)$ was close to $-\pi$; the center of the left valley. From the connectedness of $f_T(X)$ it follows that for any value between -1 and 2, e.g., 0.1, we may find an initial position x_0' such that the particle is at this value, here 0.1, at time $T = 25\,\mathrm{s}$.

We will later derive ODEs for the hardware setting and metastability therein. Analogous arguments, as the ones made for the landscape, will be seen to hold for them, too.

In fact, it was shown by Marino [**?**] that under mild assumptions on the ODEs that reflect constraints by Newtonian physics, for any system that has two stable fixed points, we can find an input that results in the system's output

attaining any output value between the two stable outputs, at an arbitrarily late time. In short: there exists no shape of the hill and no implementation of a flip-flop that does not have such behavior.

4 Calculating the MTBU

We have seen that often metastability cannot be completely avoided, so we have to cope with it. However, if we still want to build reliable systems, we need to be able to quantify the risk of metastability-induced failures. To this end, we must first specify more precisely the notion of a "metastability-induced failure". Consider a flip flop FF_m whose output is metastable – when would that be a problem? In fact, the problem starts when one (or more) flip flops (FF_v) connected to FF'_m s output, either directly or through some combinational gates, capture an incorrect or metastable value as a consequence of FF_m being metastable. We call this a *metastable upset*. Notice that FF_m becoming metastable alone does not yet constitute a metastable upset; if the metastability resolves before FF_v captures the output, then no harm was done. So apparently whether a metastable output of FF_m causes a metastable upset depends on how much time we give FF_m to resolve its metastability. We call the *available* time margin between the input change at FF_v in the regular case (i.e. without metastability) and the instant of FF_v actually capturing its input (typically with the rising clock edge) the *resolution time* t_{res}. This is the safety margin available for a potential metastability at FF_m to resolve, and it is clearly a design choice. We have already seen that the time *required* to resolve actually depends on how deep FF_m has been driven into metastability, and clearly, the more resolution time we allow, the deeper cases of metastability we can still tolerate. So actually there are other reasons to provide such a margin, like tolerances in delay parameters, as well as their temperature and voltage dependence; but the resolution time is the actual margin that remains in the given operation. So practically, the resolution time depends on how aggressively we optimize the clock period of our system – being more conservative about tolerances also directly increases t_{res}.

In order to approach a risk quantification, in a next step let us develop a model of the storage element under examination, especially its behavior in the metastable state. To this end we need to decompose the flip flop into its two constituent latches, as discussed in Section 2. As was already laid out there, each of these latches, in turn contains a pair of cross-coupled inverters that form the actual storage element.

To be able to model analog behavior like metastable voltage, we need to leave the digital abstraction and consider the analog function of the inverter. Figure 11 shows the transfer characteristic (output voltage as a function of input voltage) of the inverter. Since we are interested in the metastable behavior, we can concentrate on the middle range of the characteristics (around half the voltage). Fortunately, the characteristics is a straight line there, which means the inverter acts as an analog amplifier $V_{out} = -A \cdot V_{in} + \frac{V_{DD}}{2}$ (since the slope is

negative we put the minus sign in front to get a positive value for the gain A). To get rid of the constant term, we will move our zero potential to $\frac{V_{DD}}{2}$, which, mathematically speaking, is a simple coordinate transformation. So from now on "0" corresponds to the perfect metastable point, while a perfect LO (HI) equals $-\frac{V_{DD}}{2}$ ($+\frac{V_{DD}}{2}$). This is the simple abstraction we will use in the following. For the original paper proposing this model see [**?**].

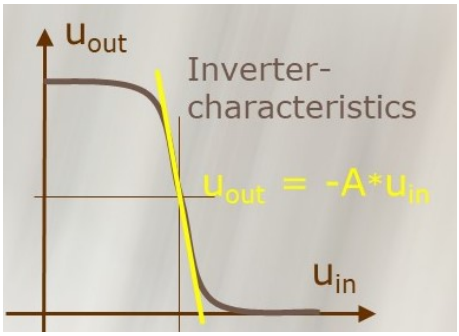

Figure 11: Inverter characteristics and its linear approximation

In addition we need to model the dynamic behavior as well. As a rough approximation (which turns out quite valid in practice), let us assume this is a first order dynamic system, which, in terms of electrical engineering, is a low-pass constituted by a so called RC element (a series resistor R followed by a parallel capacitor C, as shown in Figure 12). By applying circuit analysis techniques (Kirchhoff's Laws), Ohm's Law ($U = R \cdot I$) and the U/I characteristics of a capacitor ($i_C = C \cdot \dot{u}_C$), and considering the above amplifier behavior, the analog behavior of the inverter can finally be expressed by

$$u_{out}(t) = -A \cdot u_{in} - RC \cdot \dot{u}_{out} \tag{1}$$

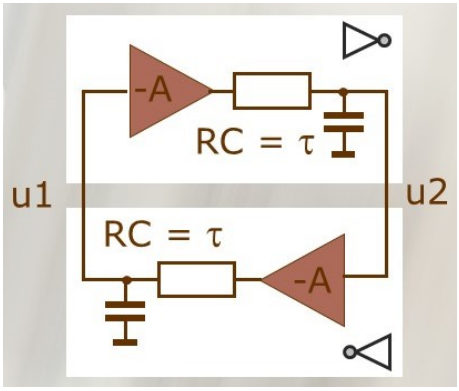

Figure 12: Circuit model for the inverter loop

Now, with this model, we are in the position to form a storage element by cross-coupling two such inverters, where u_{out} of one becomes u_{in} of the other. Denoting the output voltages of the two inverters by u_1 and u_2, the resulting ODEs are

$$\begin{aligned}\frac{d}{dt}u_1 &= \frac{-Au_2 + u_1}{RC} \\ \frac{d}{dt}u_2 &= \frac{-Au_1 + u_2}{RC}\end{aligned} \quad .$$

By solving the resulting system of two first-order ODEs we obtain a solution for how the storage cell will leave its metastable state. Note that for this purpose we consider the latch in opaque mode (D input decoupled), which, conveniently, yielded a homogeneous ODE. This solution is

$$u_2(t) = \frac{{U_2}^0 - {U_1}^0}{2} \cdot e^{\frac{A-l}{RC}t} + \frac{{U_2}^0 + {U_1}^0}{2} \cdot e^{-\frac{A+l}{RC}t} \tag{2}$$

Here u_1 is the voltage at the input end of the cell, and u_2 at the output, which is the relevant one for us as it represents the voltage trajectory along which the cell leaves the metastable state. Furthermore, ${U_1}^0$ and ${U_2}^0$ are the respective starting values (voltages across the capacitors) at the moment when the resolution starts.

Exercise 3. *The case ${U_1}^0 = {U_2}^0 = 0$ represents the perfect metastable state (recall the offset we introduced). How does the output trajectory look like in this case? Is this as expected?*

The left term in Equation (2) has a positive exponent in the exponential function, so it will increase over time. It describes how the *difference* in the initial voltages U_1 and U_2 evolves over time, if the inverter loop is left on its own. The right term, in contrast, describes the evolution of the initial *common mode voltage* (the average of U_1 and U_2) over time. As this term has a negative exponent, it will decay. Since, moreover, in regular operation we will always have $u_2(t) \approx -u_1(t)$ and therefore do not expect a significant common mode component, it is quite safe to ignore the right term in the following. In that case we end up with a purely exponential trajectory with time constant $\frac{RC}{A-1}$ and starting value $\frac{{U_2}^0 - {U_1}^0}{2}$, which we will further call U_d.

$$u_2(t) = U_d \cdot e^{\frac{A-l}{RC}t} \tag{3}$$

Clearly, this trajectory will saturate once it approaches V_{DD} or GND, but for the area around the metastable point our amplifier approximation without saturation will be sufficiently precise.

We can now use this trajectory of u_2 to predict the resolution time from a given metastable state (see Figures 13 and 14): We simply have to define a threshold U_{th}, beyond which the state is considered "resolved". Considering that the resolution can go both in upward or downward direction, we demand $|u_2(t_{res})| > U_{th}$. For a given resolution time t_{res} we can therefore calculate the

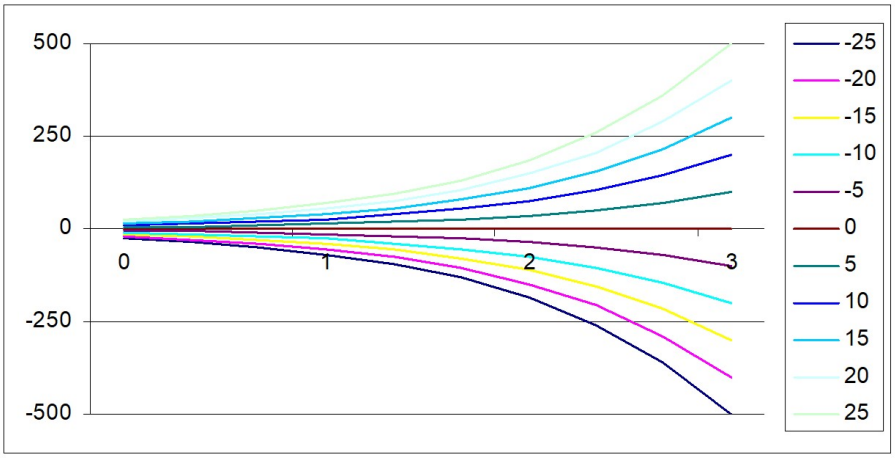

Figure 13: Trajectories of u_2 for different initial values of U_d

smallest difference voltage U'_d required to still cross the threshold U_{th} and thus reach a resolved state by rephrasing Equation (3) as

$$|U'_d| > U_{th} \cdot e^{-\frac{A-l}{RC} t_{res}} \tag{4}$$

Figure 15 depicts the resolution times for $U_{th} = 0.1\,\mathrm{V}$ for the two inverters parametrized as in Figure 14.

In practice, Equation (4) gives us the condition for the inverter loop to still resolve, if left on its own (i.e. latch switched to opaque) with an initial difference voltage U_d. Intuitively it is clear that for smaller (absolute) U_d the exponential trajectory will be too flat to reach the threshold in time. So for any U_d with $-U'_d < U_d < +U'_d$ we cannot obtain resolution within the available resolution time t_{res}.

In a next step we will elaborate a mapping from input timing to difference voltage. To this end, let us assume we have a linear voltage ramp at the input with slope S (in V/s) as illustrated in Figure 16. For this signal to cross the critical interval $[-U'_d, +U'_d]$ it will take a time $T_{crit} = \frac{2U'_d}{S}$. This is the width of the critical time window during which the inverter loop should not be decoupled, i.e. the latch be switched to opaque. Using U'_d from Equation (4) we get

$$T_{crit} = \frac{2U_{th}}{S} \cdot e^{-\frac{A-l}{RC} t_{res}} \tag{5}$$

Note that the size of the critical window can be scaled with the resolution time, so by increasing t_{res}, we can make T_{crit} smaller.

If we use our latch as part of a flip-flop in a synchronous circuit with a clock period $T_{clk} = \frac{1}{f_{clk}}$, we can turn this argument around and postulate, that around the rising clock edge (at which we switch the master latch to opaque) we must keep an interval of width T_{crit} free from input transitions, because otherwise

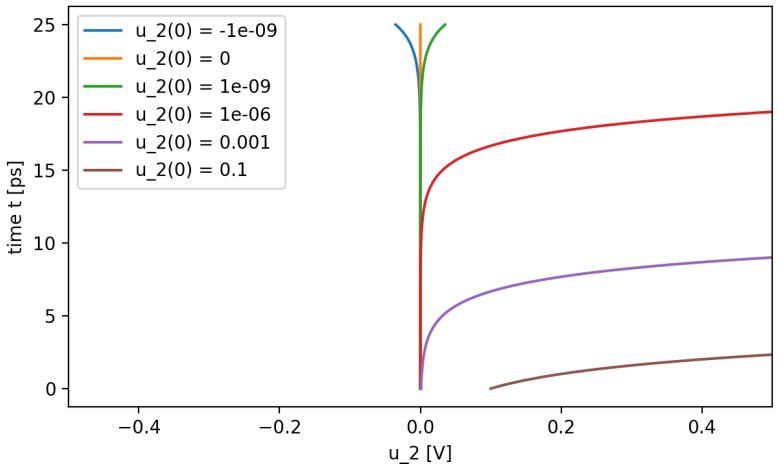

Figure 14: Trajectories of u_2 for different initial values of $u_2(0)$ plotted as for the landscape scenario in Figure 2. We set $u_1(0) = -u_2(0)$. Here, we assume an inverter with delay of 2 ps, accordingly choosing $RC = 2/0.69$ ps, and an amplification of $A = 3$.

the distance between the input transition and the clock transition will violate Equation (5). In consequence this means, that for an input transition that arrives completely uncorrelated to the clock (mind this assumption!), we have a probability of $P_{crit} = \frac{T_{crit}}{T_{clk}}$ for it to occur within the window T_{crit}, hence violate Equation (5), thus create a too low difference voltage U_d, and in final consequence cause a metastable state that does not resolve within the available resolution time t'_{res}. This is what we had called a metastable upset. With such uncorrelated transitions on the data input occurring at rate λ_d, we obtain a rate of metastable upsets given by $\lambda_{upset} = \lambda_d \cdot P_{crit}$. Putting that all together, we obtain

$$\lambda_{upset} = \lambda_d \cdot P_{crit} = \lambda_d \cdot f_{clk} \cdot T_{crit} = \lambda_d \cdot f_{clk} \frac{2U_{th}}{S} \cdot e^{-\frac{A-l}{RC} t_{res}} \tag{6}$$

If we now substitute the circuit parameters $\frac{2U_{th}}{S} = T_W$ (aperture window) and $\frac{RC}{A-1} = \tau_c$ (resolution time constant), we finally obtain the commonly used equation for estimating the mean time between upsets MTBU $= \frac{1}{\lambda_{upset}}$:

$$\text{MTBU} = \frac{1}{\lambda_d \cdot f_{clk} \cdot T_W} \cdot e^{\frac{t_{res}}{\tau_c}} \tag{7}$$

The MTBU is the expected interval between two successive metastable upsets. This is a statistical value that does not allow any direct conclusion on a specific case. Typically one will design a system for an MTBU of over 10 years,

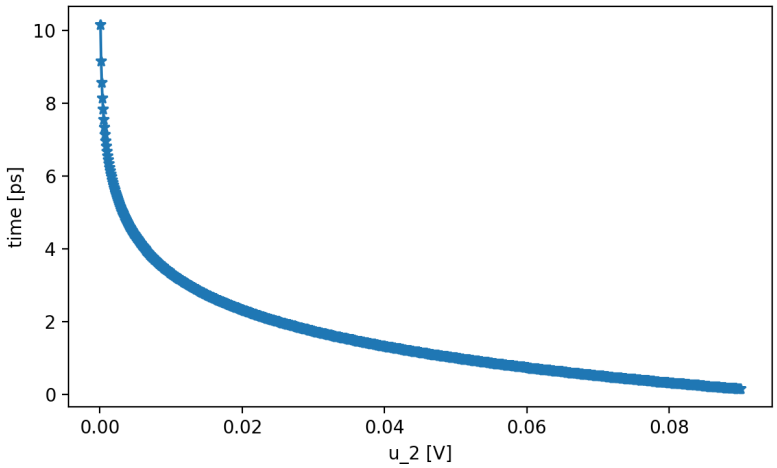

Figure 15: Resolution times for u_2 if the threshold is 0.1 V. Different initial values of $u_2(0)$ are plotted as for the landscape scenario in Figure 7. Again, we set $u_1(0) = -u_2(0)$. An inverter with $RC = 2/0.69\,\text{ps}$ and an amplification of $A = 3$ is assumed.

so the probability of seeing a metastable upset in the lifetime of the system is very low (but still one cannot safely exclude such upsets). In this sense the MTBU is a very important design parameter.

In essence, Equation (7) shows an inverse proportionality of MTBU to the rate λ_d of input events as well as to the clock frequency, which is both intuitive, since the more transitions we have (both on clock and data), the higher the threat of hitting the critical window. Note that the resolution time t_{res} is in the exponent, which means higher resolution time has a dramatic positive effect on MTBU.

5 Determination of τ_c and T_W through measurement

If we draw Equation (7) in a semi-logartithmic diagram with t_{res} on the x-axis and ln(MTBU) on the y-axis, we obtain a straight line:

$$\ln(\text{MTBU}(t_{res})) = \frac{1}{\tau_c} \cdot t_{res} - \ln(T_W) - \ln(\lambda_d \cdot f_{clk}) \tag{8}$$

The slope is given by $\frac{1}{\tau_c}$ and the offset determined by the logarithms of clock frequency f_{clk}, rate λ_d of input transitions, and a parameter T_W. Recall from the previous section that both, τ_c and T_W are circuit parameters; the former characterizing the dynamic properties of the inverter, and the latter being

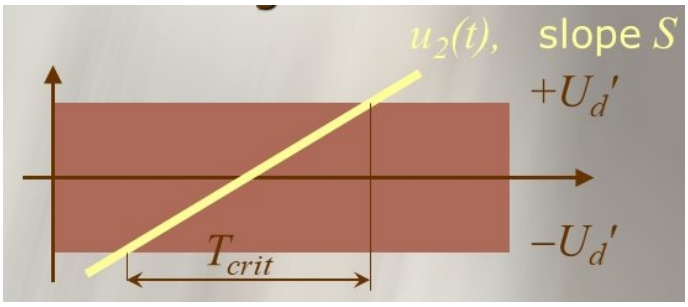

Figure 16: Mapping from critical voltage to critical time window

determined by threshold voltage and signal slope. Clearly, if we want to estimate MTBU through Equation (7), we need to have these parameters available. Looking at the linear graph described by Equation (8), these parameters are relatively easy to measure: For a given setting of f_{clk} and λ_d we can experimentally determine the graph by measuring the MTBU for different choices of t_{res}. The slope and the offset of the resulting straight line will allow us to determine the two circuit parameters. For integrated circuits, of course the manufacturers will do this characterization and provide the required parameters (even though they are often hard to find in the data sheets) – but they typically use the described measurement principle.

The good thing about this approach is that is does not suffer from the substantial simplifications that have been made in the derivation of Equation (7), such as assuming a first-order dynamic system, assuming equal parameters for forward and backward inverter, dropping the decaying term in Equation (6), etc. In fact it only uses the insight that the dependence between resolution time and MTBU is exponential, which is well confirmed by experiments in the literature. The rest is in some sense "calibrated away" through the experimental measurement.

On the downside, the practical implementation of the experimental measurement is quite challenging: To keep the measurement duration within reasonable limits, short MTBU must be targeted, as for each setting of t_{res} numerous upsets must be collected to obtain valid statistics (recall that MTBU is a statistical value, and the upsets do not occur regularly spaced). This requires short resolution times with precise control of their temporal spacing (in the order of 10 ps for modern technologies).

Figure 17 shows an example circuit from [**?**]. Here the target flip flop (termed DUT for "device under test") is provided with two independent clocks, one (osc 1) at the clock input and one (osc 2) at the data input. This creates exactly the equally distributed phase relation that was assumed in the derivation of Equation (7). A controllable delay element (var Δ) is used to determine the point in time at which the output of the DUT is captured by a successor flip flop (FF_1). Obviously this determines the resolution time, and the controllable delay element is the one that requires specific care in its implementation. Another

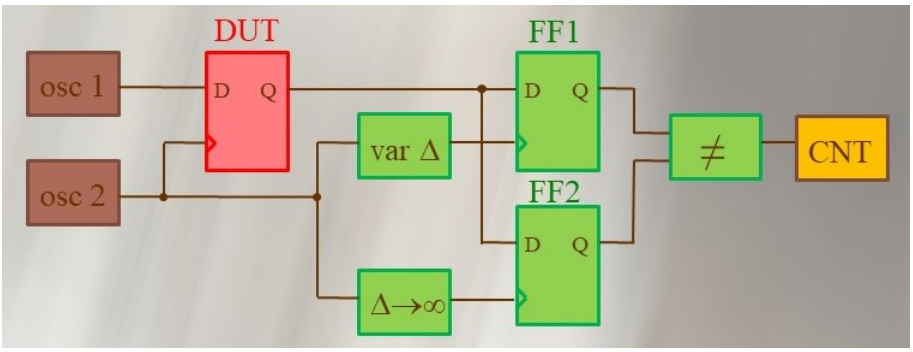

Figure 17: Block diagram for metastability characterization circuit

flip flop, FF_2, is used to capture the output of the DUT at a significantly later point in time, when it can be safely assumed that any metastability in the DUT has already resolved[2]. By comparing the outputs of FF_1 and FF_2 we can determine whether the value captured by FF_1 had already been the final, resolved value. In case of a mismatch we can conclude that the output of the DUT had not yet been resolved at the time FF_1 captured it, so the resolution took longer than the resolution time allowed through the controllable delay element. That indicates an upset. By counting these upsets and relating them to the observation period, a fault rate, and its inverse, the MTBU can be calculated. The pair of values (t_{res}, MTBU) thus obtained constitutes one point of our linear graph. To determine a next point a new value of t_{res} is chosen through the controllable delay element, and the process for measuring the associated MTBU is repeated. Knowing that the graph is linear, one could essentially come along with measuring two points only; however in practice there is some tolerance for both, t_{res} as well as MTBU, and so it makes sense to collect several points and do a linear regression fitting as shown in Figure 18.

The relevant slope is the one in the center of the figure (from 0 to 400ps). The parallel lines in the lower right corner characterize the metastability of the slave latch. As can be seen, the slave latch becomes metastable only when the metastability (of the master) persists longer than the clock half period, which occurs very rarely.

[2]Note that, since the duration of the metastable state is essentially unbounded, there is always a residual probability for FF_2 to still capture an unresolved state, but with an appropriate dimensioning the related probability becomes negligible

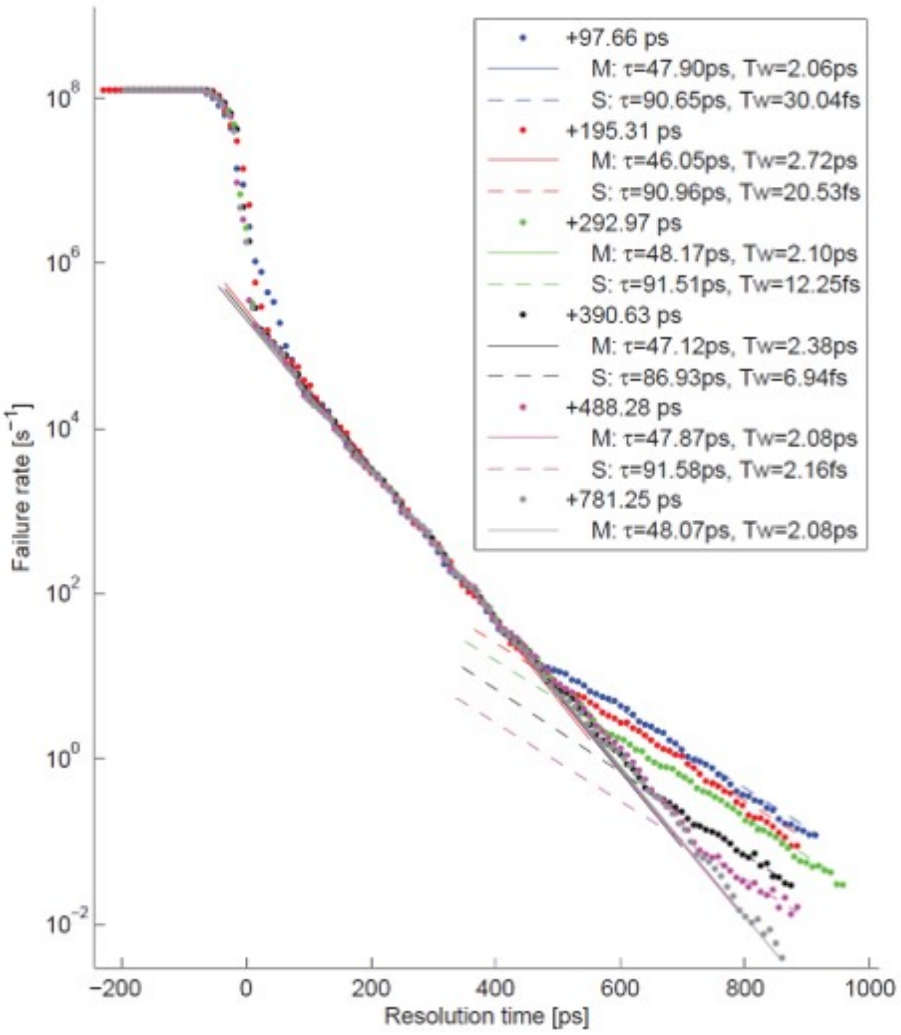

Figure 18: Measurement result from metastability characterization

6 Mitigating metastability

Equation (7) not only allows us to calculate the MTBU for a given setting – it is also a good foundation for studying which knobs to turn to maximize the MTBU. Let us use its inverse for this purpose in the following:

$$UR = \frac{1}{MTBU} = \lambda_d \cdot f_{clk} \cdot T_W \cdot e^{-\frac{t_{res}}{\tau_c}} \tag{9}$$

An obvious measure to decrease the upset rate UR is to minimize the clock frequency f_{clk} or the data rate λ_d. While this is most often impossible or at least undesired for a given application, it is still good to keep in mind that unnecessarily high values there will worsen the upset rate. Another important insight is that setting either f_{clk} or λ_d to zero, i.e. completely avoiding the sampling of an asynchronous input, is the only way to get a zero upset rate. Neither the exponential function can become zero, nor can we expect a circuit to exhibit $T_W = 0$. As for the circuit parameters, a low value of T_W is desirable, corresponding to a small sensitive window; as well as a low metastability resolution constant τ_c, allowing fast resolution. These parameters can only be influenced by choosing the most modern technology available, as they get better at roughly the same pace as the propagation delay of a technology decreases.

Finally we have the resolution time as the most effective design parameter, since it has an exponential influence on the upset rate. Unsurprisingly, larger t_{res} results in lower FR. This is because, clearly, allowing more time for resolution increases the chances of sampling a resolved value. In a strictly synchronous system the resolution time is determined by the choice of the clock frequency as follows:

$$T_{clk} = t_{co} + t_{comb} + t_{su} + t_{res} \tag{10}$$

In the relation given in Equation (10) the clock period is determined such that all delays involved in the propagation of a signal edge for the output of a producer flip flop to the input of a consumer flip flop can be accommodated (in the worst case). More specifically, t_{co}, the nominal clock-to-output delay, is the delay it takes for a flip flop to present at its output the input value it just captured with a clock edge (i.e. the time from the clock edge to the point in time where the output is valid; in the non-metastable case); t_{comb} is the propagation delay of combinational elements that process the output on the way to the receiver flip flop (including interconnect delays), and t_{su} is the setup time required by the receiving flip flop, i.e. the time the input value needs to be stable before the clock edge in order to safely prevent metastability. Obviously, the clock period must be chosen large enough to accommodate these parameters to make the system work at all. Any choice of T_{clk} larger than that minimum will leave room for a non-zero resolution time t_{res} which, as explained previously, can be useful as an extra margin for allowing metastability to resolve, should it occur. In this sense, increasing t_{res} implies a reduction of the attainable clock frequency, which is again undesired. In the following section we will see how to work around that.

For the moment let us conclude that an aggressive choice of the clock frequency leaves no room for metastability to resolve. This is, however, generally unproblematic for flip flops that are never exposed to asynchronous inputs (like those well within a synchronous environment), as these are not supposed to ever get metastable.

7 (Waiting) Synchronizer circuits

In the derivation of Equation (7) we have seen that metastability occurs when a flip flop has to turn a marginal initial voltage difference U_d into a clear logic state at its output in limited time. If it does not succeed in doing so, the subsequent flip flop will again capture a marginal U_d and propagate the metastability. However, let us analyze this case for a cascade of n flip flops, with $\mathrm{FF}_{(i+1)}$'s input directly connected to FF_i's output, without any combinational gates in between (see Figure 19). Now assume FF_1 starts with a marginal voltage difference $U_{d,1}$ and does not get its output across the threshold voltage within the available time (which is $T_{clk} - t_{co} - t_{su}$). So FF_2 will be confronted with an undefined input voltage. However, during the available resolution time the output voltage of FF_1 has at least undergone an exponential increase (positive or negative), so FF_2 will start from a higher voltage difference $U_{d,2} > U_{d,1}$. Starting from that point, FF_2 will further increase the output voltage during its resolution time and either already present a clean logic level to its successor FF_3, or at least cause a further increased initial voltage difference $U_{d,3}$, and so on.

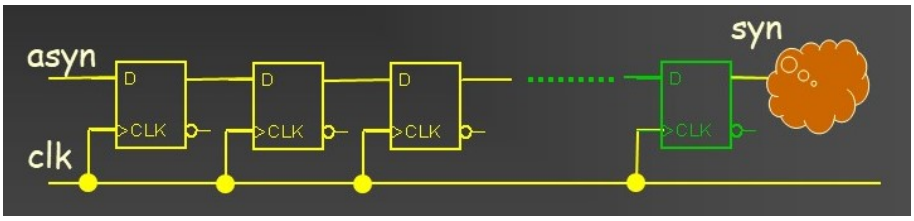

Figure 19: Flip flop cascade forming a multistage synchronizer

So in fact, the further one goes downstream along the chain of flip flops, the more likely it becomes that metastability has resolved, as increasingly more resolution time is being aggregated. A usual assumption is that this aggregation occurs in the form of a summation, i.e. the resolution times between the individual flip flops can be summed up. This is an excellent way of increasing the effective resolution time without sacrificing performance by reducing the clock period

Exercise 4. *Start from Equation* (3) *and show that for a multistage synchronizer summing up the resolution times is indeed justified.*

So a common way of mitigating metastability at asynchronous inputs is to use such a chain of flip flops. This is called a *synchronizer*. In its simplest form

it just comprises two flip flops, a first one that is exposed to the asynchronous input and hence likely to become metastable, followed by a second one who only samples its output after (nearly) a full clock cycle of resolution time. It is important that no combinational logic is inserted in between, as that would reduce the available t_{res} and hence drastically (exponentially) increase the upset rate. Also, no fork is allowed in the connection between the flip flops: In fact, due to tolerances in the thresholds, it is not sure that FF_2 will actually consider FF_1's undefined output as metastable, it might well happen to interpret it as "above threshold" (logic HI) or "below threshold" (LO), which is perfectly fine. However, if the different receiving flip flops at the ends of a fork would make different interpretations (one going for HI and the other going for LO), that would be a problem.

The synchronizer constituted by just two flip flops is called *one-stage synchronizer* or *two-flop synchronizer* (note that a single flip flop is not yet a synchronizer!). Its MTBU can be calculated by using Equation (7) with a resolution time calculated from Equation (10), using $t_{comb} = 0$. Should the MTBU thus obtained not be sufficient, another flip flop stage can be appended, yielding a two-stage (3-flop) synchronizer with twice the resolution time. In the general case of n flip flops cascaded, we obtain a $(n-1)$-stage (n-flop) synchronizer with $(n-1)$-fold resolution time. Due to the exponential impact of t_{res} on MTBU, each extra stage substantially improves the situation, and most often MTBU values of hundred years are easily achieved with only a few stages.

So even though it is not possible to completely rule out metastable upsets in the general case, their occurrence can be made arbitrarily improbable through the use of synchronizers. The price, though, is a higher latency: A transition arriving at the input of an n-stage synchronizer takes n extra clock cycles before it reaches the output. This can be a substantial performance penalty in the communication among clock domains. However, at least a reduction of the clock frequency for the whole synchronous block can be avoided.

The synchronizer circuit described here is the most simple and popular one, but it should be mentioned here that other approaches exist as well. One uses just a chain of 3 flip flops and increases MTBU by simply scaling the clock of the two upstream flip flops, while having the third flip flop in the row run at full clock. With a scaling factor of k for the clock, this results in an overall resolution time of (approximately) $(k+1)$ times that of a single stage. Yet another approach is using n flip flops in parallel and alternating in supplying clock edges to them as well in reading their data in such a way that a "resting time‘" of n clock cycles is obtained for each individual flip flop. More details can be found in [**?**].

The principle of all these synchronizers is to trade latency for a better MTBU, by increasing t_{res} in the exponential term of Equation (7). These types of synchronizers are generally called *waiting synchronizers*.

8 Eliminating metastability

Let us again have a look at Equation (7). We have already discussed that the right, exponential term describes the chances of having metastability resolved within the available time t_{res}. This becomes relevant once the flip flop has actually become metastable. The probability for the latter is represented by the left term in the equation. For the waiting synchronizers we have so far addressed the exponential term, which obviously cannot yield a zero upset rate.

With respect to the right term recall a central assumption in the derivation of Equation (7): The transitions on the data input arrive completely uncorrelated to the clock. This assumption was instrumental in deriving the probability of getting into metastability. Note that this is consequently a fundamental prerequisite for the validity of Equation (7) for calculating the MTBU of the waiting synchronizers presented above. There are, however, cases where clock and data *are* temporally correlated, like in case of a clock domain crossing with the clocks on both sides being derived from the same source by division or a PLL. In such a setting one may easily encounter a periodic sequence of distinct phase values. Just imagine the case of a 200MHz clock (5ns period) and a 250MHz clock (4ns period) as a simple example: Starting with a rising transition on both sides, the 250 MHZ clock will, relative to its next rising edge, observe the 200MHz clock to lag behind by 1ns, then by 2ns and 3ns at its next rising edges, and finally they will be in phase again at the 5th rising edge (which is actually the 4th rising edge of the 200MHz clock). No other values will ever be seen – which is obviously far from the assumption of uncorrelated transitions.

However, such a setting does not represent the "general case" for which Marino's formal proof stated that metastability is inevitable [**?**]. In fact the input space for the decision (the phase) is now not continuous anymore; it just comprises 5 discrete values. In the above example e.g., one could simply move the clocks by 0.5 ns relative to each other and then no data transition will end up closer than 0.5 ns to any clock edge, which may, depending on the specific parameters, allow to stay out of the relevant setup/hold windows – for sure, as these phase values occur deterministically. However, note that without this phase shift an upset would occur every 200 ns, while Equation (7) would completely mispredict the MTBU. This time shift is the principle of other synchronizers, like the time delay synchronizer, the data delay synchronizer. While it is probably not justified to call the insertion of appropriate static delay elements during the design process a synchronizer already, the task becomes more challenging, if the appropriate setting of these delay elements has to be calibrated upon each power-up or even re-calibrated during operation through phase estimators. Such synchronizers are capable of handling jitter between the clocks from the same source, and they can even accommodate clocks that have the same (nominal) frequency but come from different sources.

Even knowing that both, the clock and the input data are periodic (without having the same source or being correlated otherwise), is an a-priori knowledge that can be leveraged to build a synchronizer: In [**?**] presents an "Even/Odd Synchronizer" that continuously observes the phase between clock and input

data, and, by knowing (or learning) their ratio, can predict those cases where a setup/hold violation might trigger metastability. If the prediction identifies such a case, the phase is changed by 180 degree and metastability hence avoided. However, it should be noted, that the metastability problem now has been moved to the phase estimator – where, fortunately, its appropriate handling through waiting synchronizers incurs lower performance penalty. Knowing that clock and data are uncorrelated here, we have a continuous input space (every phase relation will occur with the same probability), so clearly the metastability problem cannot be completely removed. What is done, in fact, is to change the probability distribution of the phase for the data path to one that does not comprise the critical ones anymore.

A completely different attempt of eliminating metastability is voting: The hope is that by having n flip flops sample the same asynchronous input concurrently, only a minority of them will get metastable (thanks to subtle variations in timing and parameters), and so a m-of-n majority vote (with $m = \frac{n+1}{2}$) can mask the metastable outputs. Unfortunately this does not work: In case one of the flip flops actually gets metastable, this implies, that the data transition is indeed close to the clock transition. In that case, however, it may well be that some (say r) of the non-metastable flip flops captured their input before the critical transition occurred, while the others (due to the variations) captured their input after the data transition. In the unfortunate case of $r = \frac{n-1}{2}$ we have one metastable flip flop, while half of the remaining ones disagrees with the other half. So it is exactly up to the metastable one to decide, in which case the voter output becomes metastable. One can play this with other choices of m, but the bottom line is that there is always a residual constellation where the metastable flip flop decides the outcome – so metastability cannot be reliably masked by a voter.

There have also been attempts to detect metastability through various provisions. While this is not impossible, the problem here is that, strangely enough, the decision whether metastability occurred, in itself carries the risk of ending up in metastability: After all it is a mapping from a continuous space (voltage) to a binary one. So this cannot be a reliable measure to safely eliminate metastability.

A more promising approach is to simply accept the risk of metastable upsets, but limit the error they introduce to the absolutely necessary amount (like the LSB in a digital value). In a sensor/actuator system where both, inputs and outputs are analog this may lead to minute changes in the output that are absolutely acceptable. More on that will follow in Section **??**.

9 How to build a GALS system

A final word about the proper use of synchronizers for parallel data: Synchronizing all bits of a multi-bit data bus leads to consistency problems, due to the skew (temporal displacement) of the transitions involved when changing from one data word to a next one. To illustrate that, let us assume we have an 8 bit

data bus on which we transmit the following sequence of data words "00110011" and "00110000". Since, due to inevitable delay mismatches the two transition in the rightmost bist will not arrive at the same time, the receiver will see the following sequence (assuming the rightmost bit is faster): "00110011", "00110010" "00110000". It depends on the amount af the skew, how long the inconsistent data word in the middle, that just results from the switching process, will be visible for the receiver. Still there is a chance that the receiver will capture that invalid word, and a waiting synchronizer cannot prevent that – after all there is no metastability involved here.

The correct way of building such an interface is to encapsulate the actual data transfer into a handshake that is executed by two extra signals, namely a *req* (request) signal with which the sender indicates to the receiver that the data is valid and should be captured, and an *ack* (acknowledge) signal used by the receiver to confirm the receipt of the data. More specifically, the handshake will comprise the following steps:

1. send data
2. activate *req* (after some delay to make sure all data transitions are over)
3. capture data
4. activate *ack*

Note that in this process synchronizers for the data bits are obsolete: The protocol implies that data are stable when *req* is activated, so data transition during the sampling will not occur. However, the handshake signals *req* and *ack* need to be synchronized, but with just one signal per direction consistency problems cannot occur either. Note that having several synchronizer stages for *req* and *ack* may painfully reduce the transfer speed of the interface (the handshake requires both, the synchronizer delay at *req* and that at *ack* to be lined up). Therefore, sometimes FIFOs are used for the data to allow for pipelined transmissions and hence increase the throughput even if the latency is constrained by the synchronizers. Figure 20 shows a typical solution based on this principle (without a FIFO).

10 Metastability trends

In the early days of synchronous digital design with low clock frequencies, few interfaces and ample timing margins, metastability has not even been noticed. However, over time problems (in fact metastable upsets) were experienced, and though the seminal works of Molnar, Chaney [**?**] and Kinniment [**?**] an understanding and related theory of metastability in digital circuits started to be developed. Unfortunately, the tremendous progress in CMOS technology that helped boost clock frequencies by orders of magnitude over the past decades, did not mitigate the metastability issues: In fact, the relevant circuit parameters, namely τ_c and T_W, improved with the same pace as the propagation

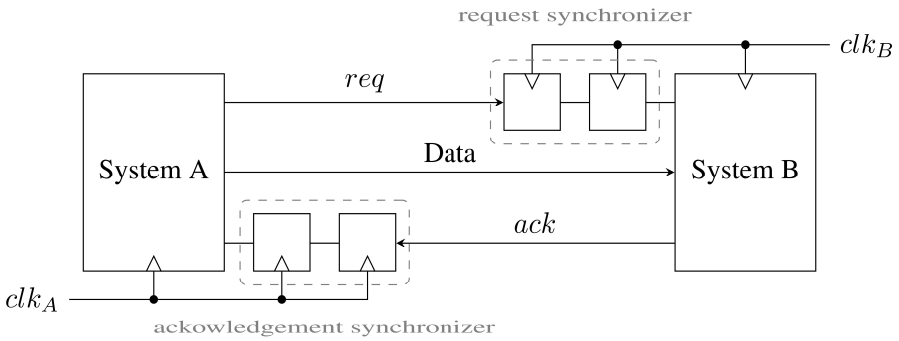

Figure 20: A typical example for a multi-bit clock domain crossing in GALS

delays in a technology, and hence, simply put, through scaling all parameters in Equation (7), the MTBU remained roughly constant.

While this is true for a single filp flop, the number of flip flops interfacing clock domain crossings has increased enormously. This is due to the increasing number of clock domains seen in modern system-on-chip architectures (recall the GALS architectures from Section **??**, e.g.). Consequently, to keep the overall MTBU constant, the upset rate at a single flip flop must be drastically reduced – by increasing the number of stages in the synchronizer. The latter, of course, introduces performance penalties.

On top of that, the more sophisticated technological processes that enable single-digit nanoscale feature sizes, incur higher timing tolerances, sometimes in the order of tens of percent. Considering that these tolerances directly impact the resolution time (recall Equation (10)) their effect on MTBU is exponential. This can make the difference between considering the best case and the worst case as large as several extra synchronizer stages [**?**].

11 Difference between Mutex and Synchronizer

Let us briefly recall the mission of a flip flop with an asynchronous data input: Its purpose is to align data input transitions to the temporal grid imposed by the clock. In other words, it has to uniquely associate each data transition to an interval between two specific clock edges such that the data values associated with those clock edges are well defined. To that end, it needs to decided whether a data transition occurred before of after a specific clock transition. This becomes difficult when the data transition coincides with a clock edge, and then synchronizers come to the rescue to mitigate the resulting metastability issues.

A different problem of ordering transitions occurs with the access to shared resources: Here a mutual exclusion mechanism is required to prevent two (or more) clients from accessing the same resource simultaneously (as the resource can only service one client at a time). The usual approach here is to allow access for that client first that also requested the resource first. While this is relatively easily done in a fully synchronous environment, metastability issues arise as soon as the requests can arrive in continuous time (i.e. not synchronized with each other). In that case, again, a discrete decision about a winner must be made, based on relative arrival times on a continuous scale. And obviously the problem becomes tough for coincident requests. The circuit element normally used for this function is the *mutual exclusion element (mutex)*. In some sense its mission is similar to that of the flip flop in that it needs to decide about the precedence of its two inputs, however with one important difference: It has two outputs, namely the so called "grant" signals, one for each client. So while being metastable, it is perfectly legal for the mutex not to activate (i.e. set to HI) any of its outputs; and only after resolution has finished, one of the requesting clients gets its grant activated. This can be accomplished by using low-threshold inverters at both grant outputs, and so no metastable upsets need be feared (but unbounded decision time, instead).

The mutex's mode of operation is called *value safe*, because it is considered its most important property to deliver a correct result (namely never activating both grant outputs at the same time), and it has a choice of delivering "no decision" by activating none of the grant outputs for as long as it takes. The flip flop (and synchronizer), in contrast, operates in *time safe* mode: Here the clock dictates the point in time when the output has to be ready, no matter whether it is already valid/resolved at that time. Moreover, with its single output, it lacks an option to express not being ready.

12 Summary

Starting from some thought experiments about a particle moving within a hilly landscape, we have observed stable and metastable fixed points. We then argued why the landscape scenario is related to scenarios in hardware components that occur thousands of times in larger circuits. With respect to digital hard-

ware we have seen that metastability issues emerge through the mapping from the continuous space formed by the arrival time of a data transition at a flip flop input relative to the time of the relevant clock edge. With uncorrelated timing sources for data and clock no magic solution exists to completely avoid metastable upsets. However, the mean time between upsets (MTBU) can be made arbitrarily large by just allowing enough time for the metastability to resolve before the flip flop output is used. This is the basic principle underlying the commonly used waiting synchronizers. By spreading the resolution time over multiple clock periods, multistage synchronizers can attain resolution times even substantially larger than a clock cycle. If additional knowledge about about input transitions is available, like periodicity, metastability can also be moved away from the data path, like into a phase predictor, where it can be more easily accommodated.

Complete elimination of metastability is only possible with correlated timing of clock and input data – which implies that both stem from the same timing source[3]. In that case the relative position of the input transitions is no more continuous and just needs to be appropriately aligned against the clock grid through delays – either statically, or through continuous adaptation.

Through the increasing number of clock domains and clock domain crossings in modern VLSI architectures, as well as increasing timing tolerances, the careful consideration of metastability along with the selection of appropriate synchronizers has become more important than ever.

A Exercises

1. Build a communication system of state machines.

2. Basic Synchronizer Design & Effect of Tolerances. You are given a clock domain crossing with parameters xyz. Design a synchronizer that obtains an MTBU of at least xx. Now assume that the parameters have tolerance of yy

3. Restrictions of the "General Case". You are given a clock domain crossing and need to design a synchronizer. Are there alternatives to the waiting synchronizer for the following cases: both domains have the same clock, with a known/unknown stable/unstable phase relation less/more than 1 cycle?

4. Clock domain crossing as a special case. You are given a clock domain crossing and need to design a synchronizer. Clock A has a stable frequency of x MHz, clock B has a stable frequency of y MHz. Is there a way to do better than building a waiting synchronizer? (Hint: Consider using a phase estimator)

[3]Or they have been synchronized, which just moves the metastablity problem to that synchronizer.

5. MTBU for ratiochronous clocks. You are given a clock domain crossing, with period of A is k times period of B. Discuss how you can make a safe synchronization. Can you use a waiting synchronizer here? How do you calculate the MTBU?

6. Transformations of metastability. Using the output voltage shape of a metastable flip-flop as an input, discuss the possible output shapes of a comparator/ a high threshold buffer / a low threshold buffer / a Schmitt Trigger

7. Oscillatory metastability. Use an inverter model that just exhibits a fixed pure delay from input to output. Can you make a storage loop built from such inverters metastable? Can you make it do any other unexpected thing?