Lecture 6

Metastability

In the previous lecture, we've seen how to handle the maximum possible number of worst-case faults (with asymptotically optimal skew bounds!). Or have we? There are fault models that are worse than "just" Byzantine faults. One of the issues that may arise when dealing with low-level hardware implementations of synchronization algorithms is *metastability.* Metastability occurs when a storage element — e.g. a flip-flop — is brought into an unstable equilibrium state. Not binary 0 or binary 1 (low or high output voltage, respectively), but somewhere in between! With the "right" bad input, this is always possible. Figure 6.1 shows how a flip-flop's output responds to critical input signals.

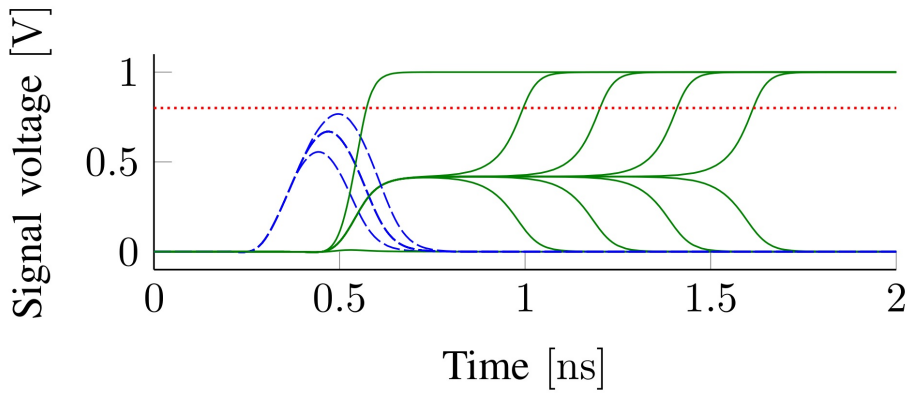

Figure 6.1: Several input (blue) and corresponding output (green) signal traces of a flip-flop. The dotted red line marks the threshold above which the output signal is reliably interpreted as a logical 1. The center blue line is actually not a single one, but many only slightly differing traces, which result in the various green outputs that remain metastable for some time.

Metastability breaks our standard Boolean abstraction, resulting in "faulty" behavior — or rather behavior that is unexpected if we neglect to account for the potential for metastability in our model. Worse, when a metastable flip-flop's output is used in computations and the result is stored in another flip-flop, the latter flip-flop may become metastable as well. So metastability can spread to

other parts of the computational logic and "infect" other storage elements.

The reason why we usually neglect metastability is because it's dealt with by electrical engineers. As unstable equilibrium state, the probability for sustained metastability decreases exponentially with time. Even the tiniest deviation from the "perfect balance" (e.g. due to thermal noise) gets amplified exponentially, resulting in quick stabilization of the storage element to one of its stable states. Whereever the danger of metastability exists, *synchronizers* are employed, i.e., storage elements specifically designed to resolve metastability as fast as possible, to reduce the probability of metastability sufficiently far before using the registers' content in computations.

Unfortunately, we want to synchronize clocks as accurately as possible, meaning that we cannot always afford to wait. Making things worse, our "worst-case" fault model of Byzantine nodes does not take into account that Byzantine nodes could try to "infect" correct nodes with metastability. Do we always need to wait for synchronizers to do their job? Can we only guarantee correct operation probabilistically?

Remarks:

- Don't mistake the synchronizers here with Awerbuch-Sipster network synchronizers. The former deal with metastability, the latter simulate a synchronous network on top of an asynchronous (fault-free) one.
- Using synchronizers is perfectly fine in most applications. Only if we have to respond very quickly (a few nanoseconds or less) to events, we need to look for alternatives.

6.1 Kleene Logic and Circuits

In order to capture how circuits behave in face of metastability, we need to understand how it propagates through logic gates. A very natural way of expressing worst-case behavior of standard circuits is Kleene logic. We extend the truth tables for Boolean logic by adding a third logic value M representing metastability and, in fact, *any* signal behavior that is not conform with what we consider a stable 0 or stable 1.

AND	0	1
0	0	0
1	0	1

AND_M	0	1	M
0	0	0	0
1	0	1	M
M	0	M	M

OR	0	1
0	0	1
1	1	1

OR_M	0	1	M
0	0	1	M
1	1	1	1
M	M	1	M

Table 6.1: Gate behavior under metastability corresponds to Kleene's 3-valued logic. A NOT_M gate simply maps M to M.

Note carefully that if one stable input already determines the output of a gate, then the other input being M does not matter. This is called *logical masking,* and it is the best guarantee one can hope for: when changing inputs affect the output, one can always "adjust" the input precisely to hitting the spot where the output is neither a logical 0 nor a logical 1.

A definition that extends this desirable behavior to arbitrary Boolean functions is the following.

Definition 6.1 (Metastable Closure)**.** *For $x, y \in \{0,1,\mathrm{M}\}^n$, we say that $x \preceq y$ if and only if $x_i \neq \mathrm{M} \Rightarrow y_i = x_i$ for all $i \in \{1,\dots,n\}$, i.e., y* is a stabilization *of x. For any Boolean $f\colon \{0,1\}^n \to \{0,1\}^m$, define the* metastable closure f_M *of f by*

$$(f_\mathrm{M})_i(x) \coloneqq \begin{cases} 0 & \text{if } f(y) = 0 \text{ for all } x \preceq y \in \{0,1\}^n \\ 1 & \text{if } f(y) = 1 \text{ for all } x \preceq y \in \{0,1\}^n \\ \mathrm{M} & \text{else.} \end{cases}$$

Example 6.2 (MUX_M)**.** *A* multiplexer *(or short* MUX*) selects between two input bits based on a select bit (it's third input). Formally,*

$$\mathrm{MUX}\colon \{0,1\}^3 \to \{0,1\}$$
$$\mathrm{MUX}(a,b,s) \coloneqq \begin{cases} a & \text{if } s = 0 \\ b & \text{if } s = 1\,. \end{cases}$$

In Figure 6.2, a standard circuit implementation of a MUX *is shown — and why it does not implement* MUX_M*.*

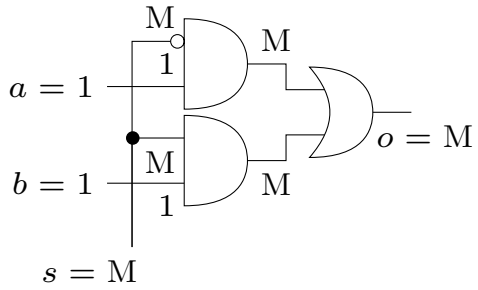

Figure 6.2: Standard MUX implementation. A black dot means that wires are joined (while regular crossings imply no contact). An empty dot is a negation, i.e., a NOT gate. AND gates are represented by the shapes that are approximately half circles, while the crescent-shaped symbol stands for an OR gate. The figure indicates in which gate inputs and outputs inputs $a = 1$, $b = 1$, and $s = \mathrm{M}$ to the circuit result; note that $\mathrm{MUX}_\mathrm{M}(1,1,\mathrm{M}) = 1$.

A metastability-containing multiplexer *(or short* CMUX*) has* MUX_M *as output function, i.e.,*

$$\mathrm{CMUX}\colon \{0,1,\mathrm{M}\}^3 \to \{0,1,\mathrm{M}\}$$
$$\mathrm{CMUX}(a,b,s) \coloneqq \mathrm{MUX}_\mathrm{M}(a,b,s) = \begin{cases} a & \text{if } s = 0 \text{ or } a = b = 0 \\ b & \text{if } s = 1 \text{ or } a = b = 1 \\ \mathrm{M} & \text{else.} \end{cases}$$

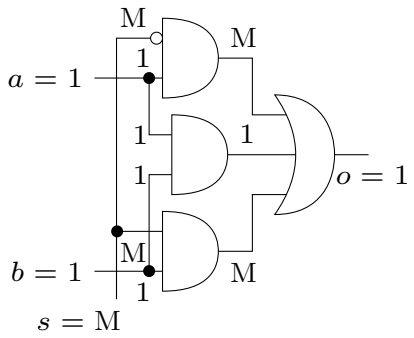

Figure 6.3: CMUX implementation. The figure indicates in which gate inputs and outputs inputs $a = 1$, $b = 1$, and $s = \mathrm{M}$ to the circuit result; the additional AND gate makes sure that the OR gate receives a stable 1 as third input if $a = b = 1$, guaranteeing a stable 1 as output.

As we will see shortly, asking for implementing the metastable closure of a Boolean function is the best we can hope for in a mathematically precise sense. However, we first need to clarify what we mean by "implement." It's exactly what one might expect (see the example above), but a somewhat wordy formalization is necessary to correctly describe the process.

Definition 6.3 (Circuit Behavior)**.** *A combinational circuit C is described as a directed acyclic graph (DAG), where n nodes are marked as inputs and m nodes are marked as outputs. Input nodes have indegree 0, output nodes have indegree 1, and all remaining nodes are gates. A gate implements $f_{\mathrm{M}} \colon \{0,1,\mathrm{M}\}^k \to \{0,1,\mathrm{M}\}$ for a Boolean function $f \colon \{0,1\}^k \to \{0,1\}$. The basic available gates are OR_{M}, $\mathrm{AND}_{\mathrm{M}}$, $\mathrm{NOT}_{\mathrm{M}}$, and the constant gates (i.e., no-input gates that provide outputs 0 or 1, respectively). In the DAG, input nodes and gates may have any number of outgoing edges, while output nodes have none. Gates have the number of inputs prescribed by their gate function.*

For a given input $x \in \{0,1,\mathrm{M}\}^n$, the evaluation $C(x)$ of C on x is determined by structural induction as follows. The i^{th} input node evaluates to x_i. As the circuit is described as a DAG, there must be a node for which all incoming edges come from nodes whose evaluation is already determined. If the node is a gate, we apply the gate function to determine its evaluation. If it is an ouput node, it evaluates to the evaluation of the unique node at which its incoming edge originates. This process is iterated until all nodes' evaluation is determined. The output of the circuit is then given by the output nodes' evaluation. We say that C implements *$g : \{0,1,\mathrm{M}\}^n \to \{0,1,\mathrm{M}\}^m$ if $g(x) \preceq C(x)$ for all $x \in \{0,1,\mathrm{M}\}^m$, i.e., $C(x)$ is a stabilization of $g(x)$.*

Remarks:

- The model does not allow for M to stabilize to 0 or 1. This is a worst-case assumption — stable values are never worse than M.
- Accordingly, accepting stabilizations of the desired output from the circuit is fine — the circuit then does better than we ask it to.

6.2 The Limits of Metastability-Containment

Theorem 6.4. *Suppose for a circuit C and a Boolean function f it holds that $C(x) = f(x)$ for all $x \in \{0,1\}^n$. Then $C(x) \preceq f_{\mathrm{M}}(x)$ for all $x \in \{0,1,\mathrm{M}\}^n$.*

Proof. Assume w.l.o.g. that $m = 1$ (otherwise, simply repeat the reasoning for each output bit). We need to show that $C(x) = b \in \{0,1\}$ implies that $f_{\mathrm{M}}(x) = b$. Hence, assume for contradiction that $C(x) = b \in \{0,1\}$, but $f_{\mathrm{M}}(x) \neq b$, for a minimal circuit C with this property. Thus, there is $y \in \{0,1\}^n$ so that $x \preceq y$ and $f(y) \neq b = C(x)$.

Consider the node connecting to the output node in C. If it is an input node, say the i^{th} input node, then $f(y) \neq x_i = C(x)$. However, as $y \in \{0,1\}^n$, we also have that $y_i = C(y) = f(y)$, so $y_i \neq x_i$. As also $x \preceq y$, this entails that $x_i = \mathrm{M}$, yielding the contradiction that $b = C(x) = x_i = \mathrm{M}$.

Now consider the case that the node connecting to the output node in C is a gate. Consider the subcircuits $C_1, \ldots, C_k$ computing the inputs to the gate and denote by g the gate function. We have that $g(C_1(y), \ldots, C_k(y)) = C(y) = f(y) \neq C(x) = g_{\mathrm{M}}(C_1(x), \ldots, C_k(x))$, where again we used that $C(y) = f(y)$ because $y \in \{0,1\}^n$.

For $i \in \{1, \ldots, k\}$, consider the Boolean function $c_i \colon \{0,1\}^n \to \{0,1\}$ given by $c_i(z) = C_i(z)$. As C was minimal, we have that $C_i(x) \preceq (c_i)_{\mathrm{M}}(x)$. As this holds for all i and using that $x \preceq y$, it follows that

$$\begin{aligned} C(x) &= g_{\mathrm{M}}(C_1(x), \ldots, C_k(x)) \\ &\preceq g_{\mathrm{M}}((c_1)_{\mathrm{M}}(x), \ldots, (c_k)_{\mathrm{M}}(x)) \\ &\preceq g_{\mathrm{M}}(c_1(y), \ldots, c_k(y)) \\ &= g(C_1(y), \ldots, C_k(y)) = C(y)\,. \end{aligned}$$

However, $C(x), C(y) \in \{0,1\}$ and $C(x) \neq C(y)$, implying the contradiction that $C(x) \not\preceq C(y)$. □

This theorem shows that we cannot do better than computing the metastable closure. We now show that the closure can also be implemented, using a generalized CMUX as key ingredient.

Lemma 6.5. *Let* $\mathrm{MUX} \colon \{0,1\}^{2^k} \times \{0,1\}^k \to \{0,1\}$ *be a the generalized* MUX *function, i.e.,* $\mathrm{MUX}(x,s) = x_s$*, where $s \in \{0,1\}^k$ is interpreted as (the binary encoding of) an index. It holds that*

$$\mathrm{MUX}_{\mathrm{M}}(x,s) = b \in \{0,1\} \Leftrightarrow \forall s \preceq s' \in \{0,1\}^k \colon x_{s'} = b$$

and there is a circuit of size $\mathcal{O}(2^k)$ implementing $\mathrm{MUX}_{\mathrm{M}}$*.*

Proof. Exercise. □

Theorem 6.6. *For any $f \colon \{0,1\}^n \to \{0,1\}^m$, a circuit implementing f_{M} exists.*

Proof. W.l.o.g., assume that $m = 1$ (otherwise, perform the construction for each output bit of f separately). Let $f \colon y \mapsto f(y)$. By Lemma 6.5, we can implement $\mathrm{MUX}_{\mathrm{M}}$. Take such a circuit for $k = n$ and feed it inputs $x_s = f(s)$ and $s = y$, see Figure 6.4. If $f(z) = b \in \{0,1\}$ for all $y \preceq z \in \{0,1\}^n$, then by Lemma 6.5 the resulting circuit C outputs b. Thus, $f_{\mathrm{M}}(y) \preceq C(y)$ for all $y \in \{0,1,\mathrm{M}\}^n$, i.e., C implements f_{M}. □

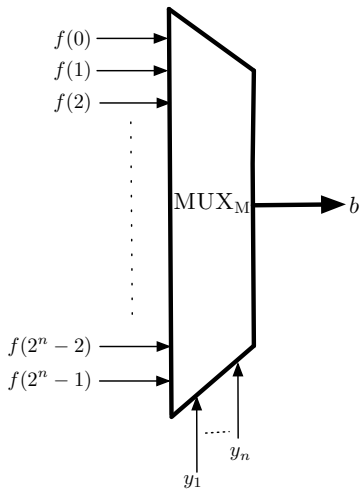

Figure 6.4: We first implement $\text{MUX}_\text{M}\colon \{0,1\}^{2^n} \times \{0,1,\text{M}\}^n \to \{0,1,\text{M}\}$ according to Lemma 6.5. Then we set the input domain to be $\{f(0),\dots,f(2^n-1)\} \times \{0,1,\text{M}\}^n$ as depicted in the figure.

Remarks:

- By Theorem 6.4, the circuit C from Theorem 6.6 satisfies that $C = f_\text{M}$.
- The construction has, unfortunately, exponential size in n. Can we do better?

6.3 Hardness of Containment

Recall that for any language in NP and word x, if x is in the language, there is a polynomially checkable proof w that this is the case. On the other hand, if x is not in the language, no proof w will work. If we had a small circuit implementing the metastable closure of the checker, we could exploit this to determine membership in the language efficiently.

Theorem 6.7. *Let $V_n(x,w)$, $n \in \mathbb{N}$, denote the family of verifier functions for 3SAT with n clauses, i.e., x encodes a 3SAT instance with n clauses, $w \in \{0,1\}^n$ is an assignment of these variables, and $V_n(x,w) = 1$ if and only if the instance is satisfied with the assignment given by w. If there is a family of circuits C_n, $n \in \mathbb{N}$, of size $n^{\mathcal{O}(1)}$ such that C_n implements $(V_n)_\text{M}$, then there is a family of circuits of size $n^{\mathcal{O}(1)}$ deciding 3SAT instances on n clauses.*

Proof. We construct a circuit simulating $(V_n)_\text{M}$. That is, we encode 0, 1, and M using two bits, e.g., 00 for 0, 11 for 1, and 01 for M. Then we construct (constant-size) circuits implementing the closure of basic gates (i.e., OR_M, AND_M, and NOT_M) in this encoding and replace all gates in the circuit implementing $(V_n)_\text{M}$ accordingly.

Now we use the simulating circuit as follows. For any instance x, compute $(V_n)_\text{M}(x,\text{M}^{3n})$ ($3n$ is the maximum number of variables in n clauses). If $(V_n)_\text{M}(x,\text{M}^{3n}) = 0$, output 0, otherwise output 1. We claim that this circuit,

which is of size $n^{\mathcal{O}(1)}$, decides 3SAT with n clauses. To see this, assume that x is a "no" instance first. Then for any assignment w, it holds that $V_n(x, w) = 0$, implying that $(V_n)_{\mathrm{M}}(x, \mathrm{M}^{3n}) = 0$ and the output is correct. On the other hand, if x is a "yes" instance, there must be at least one witness w so that $V_n(x, w) = 1$. As $\mathrm{M}^{3n} \preceq w'$ for any $w' \in \{0,1\}^{3n}$, in particular $\mathrm{M}^{3n} \preceq w$. Accordingly, $(V_n)_{\mathrm{M}}(x, \mathrm{M}^{3n}) \preceq (V_n)_{\mathrm{M}}(x, w) = 1$, implying that $(V_n)_{\mathrm{M}}(x, \mathrm{M}^{3n}) \neq 0$ and the output of the circuit is also correct. □

Remarks:

- The same argument applies to any problem in NP, implying that any verifier function of an NP-complete problem is unlikely to admit a small metastability-containing implementation.
- In the simulation, it is straightforward to properly "compute" with M. We're not actually providing bad inputs, we're only simulating the behavior of the containing circuits in our worst-case model, which is deterministic!
- One can even show *unconditional* exponential separations between the minimum size of non-containing and containing circuits for some explicit functions!
- So, should we accept our fate and give up? No, we still got some tricks up our sleeves! In many settings it's way to pessimistic that arbitrarily many metastable inputs can appear. We'll find small circuits that have output $f_{\mathrm{M}}(x)$ for inputs x with few Ms.

6.4 Containing a Bounded Number of Metastable Inputs

As the base case of our construction, we construct circuits handling only fixed positions for the (up to) k unstable bits. We take 2^k copies of a circuit computing f. For the i^{th} copy, we fix the k considered bits to the binary representation of i. Now we use a CMUX to select one of these 2^k outputs, where the original k input bits that we replaced are used as the select bits.

Lemma 6.8. *Let C be a circuit implementing $f : \{0,1\}^n \to \{0,1\}$ and $S \subseteq [n]$ with $|S| = k$. Denote by $|C|$ the size (i.e., number of gates) of C. Then there is a circuit of size at most $2^k(|C| + \mathcal{O}(1))$ that computes $f_{\mathrm{M}}(x)$ for any $x \in \{0,1,\mathrm{M}\}$ satisfying that $x_i = \mathrm{M} \Rightarrow i \in S$.*

Proof. For every assignment $a \in \{0,1\}^{|S|}$ of stable values to the indices of x that are in S, compute $g_a = f(x|_{S \leftarrow a})$, where $x|_{S \leftarrow a}$ is the bit string obtained by replacing in x the bits at the positions S by the bits of vector a. We feed the results and the actual input bits from indices in S into the the k-bit $\mathrm{MUX}_{\mathrm{M}}$ given by Lemma 6.5, such that for stable values the correct output is determined. The correctness of the construction is now immediate from the properties of $\mathrm{MUX}_{\mathrm{M}}$. Concerning the size bound, for each $a \in \{0,1\}^{|S|}$ we can use C with some fixed inputs to compute g_a. Using the size bound for the MUX from Lemma 6.5, the construction thus has size $2^k(|C| + \mathcal{O}(1))$. □

Using this construction as the base case, we increase the number of sets (i.e., possible positions of the k unstable bits) our circuits can handle.

Theorem 6.9. *Let C be a circuit implementing $f : \{0,1\}^n \to \{0,1\}$. There is a circuit of size at most $(ne/k)^{2k}(|C| + \mathcal{O}(1))$ that computes $f_{\mathrm{M}}(x)$ for any $x \in \{0,1,\mathrm{M}\}$ satisfying that $|S| \leq k$ for $S \coloneqq \{i \in \{1,\ldots,n\} \mid x_i = \mathrm{M}\}$.*

Proof. Choose an order of all k-bit subsets of $\{1,\ldots,n\}$ and let S_i, $i \in \{1,\ldots,I\}$, be the i^{th} element. Denote by C_{ij}, $1 \leq i,j \leq I$, a circuit whose outputs coincide with f_{M} whenever all unstable bits are from $S_i \cup S_j$. Set $a_i := \textsc{And}_{\mathrm{M}}(C_{i1},\ldots,C_{iI})$ ($\textsc{And}_{\mathrm{M}}$ with fan-in I is implemented by a binary tree of fan-in 2 $\textsc{And}_{\mathrm{M}}$ gates of minimum depth). We claim that $o := \textsc{Or}_{\mathrm{M}}(a_1,\ldots,a_I)$ (implemented by a tree of fan-in 2 $\textsc{Or}_{\mathrm{M}}$ gates) coincides with f_{M} whenever there are at most k unstable bits.

To show the claim, assume that $x \in \{0,1,\mathrm{M}\}^n$ is stable except at indices from some $S_i \in \binom{[n]}{k}$. Assume first that $f_{\mathrm{M}}(x) = 1$. Thus, we get that $a_i = \textsc{And}_{\mathrm{M}}(1,\ldots,1) = 1$. This implies $o = 1$, because the I-bit $\textsc{Or}_{\mathrm{M}}$ has a stable 1 at one of its inputs. Next, suppose that $f_{\mathrm{M}}(x) = 0$. Then, for each $1 \leq i' \leq I$, $C_{i'i}(x) = 0$. Hence $a_{i'} = 0$, because the I-bit $\textsc{And}_{\mathrm{M}}$ has a stable 0 at one of its inputs. It follows that $o = \textsc{Or}_{\mathrm{M}}(0,\ldots,0) = 0$. The case that $f_{\mathrm{M}}(x) = \mathrm{M}$ is trivial; hence the claim holds.

The above circuit contains the circuits C_{ij} and additionally $I^2 - 1$ many gates (a binary tree of $\textsc{And}_{\mathrm{M}}$ and $\textsc{Or}_{\mathrm{M}}$ gates). By Lemma 6.8, each C_{ij} can be implemented with size $2^{2k}(|C| + \mathcal{O}(1))$, as $|S_i \cup S_j| \leq 2k$. Moreover, using exactly all subsets of size $2k$, we use at most $\binom{n}{2k} \leq (en/2k)^{2k}$ different such circuits. This results in at most

$$\left(\frac{en}{k}\right)^{2k}(|C| + \mathcal{O}(1)) + \binom{n}{k}^2 - 1 = \left(\frac{en}{k}\right)^{2k}(|C| + \mathcal{O}(1))$$

gates. □

Bibliographic Notes

In the context of switching networks, hazards — when changing inputs to a circuit affects the output, even though the stable values are the same regardless of the values the changing bits take — were studied even before modern computers existed [Got49], in the context of relay networks. This early Japanese work remained unnoticed in the western world, and Huffman studied the same issue, also developing a CMUX [Huf57]. Huffman noted that his design principle is sufficiently general to construct hazard-free circuits for any Boolean function. Yoeli and Rinon formalized the connection to Kleene logic [YR64] (which Goto also had done before!) or, more precisely, Kleene's strong logic of indeterminancy K_3 [Kle52, §64]. Our worst-case model for metastability (propagation) presented here results in the same logic, i.e., hazard-free circuits are the same as metastability-containing circuits. In [FFL18], a more general model for clocked circuits is presented. However, so long as only standard registers are used, the computational power is the same as that of the combinational circuits introduced in this lecture. This changes when employing masking registers, which "mask" internal metastability to the outside world by outputting a stable value;

the result is that metastability may result in late transitions only, which in the worst case may result in M being read from the register in a single round.

The fact that metastability cannot be avoided in general is, in essence, a topological statement: As the output of a bistable element like a flip-flop is, due to physics, a continuous function of its input, the fact that there are two distinct stable states necessitates at least one unstable third equilibrium state. This was shown by Marino [**?**]. This impossibility holds also in the abstract model given here — no circuit can reliably detect or resolve metastability of its inputs [FFL18].

All of these works leave aside the complexity question, namely how large containing circuits must be. The lower bound given here is very simple, but to the best of our knowledge was first formalized by Ikenmeyer et al. [IKL$^+$18], who also show unconditional exponential separations between containing and standard circuits based on monotone circuit complexity. The same work also gives a construction for circuits containing k bits, which is slightly weaker than the one given here. A number of works provides small circuits whose output coincides with f_{M} for specific f and certain inputs. Most notably, this is the case for sorting [BLM18], which we will study in the next lecture.

Bibliography

[BLM18] Johannes Bund, Christoph Lenzen, and Moti Medina. Optimal Metastability-Containing Sorting Networks. In *Design, Automation and Test in Europe (DATE)*, 2018. To appear. Preliminary version available at https://arxiv.org/abs/1801.07549.

[FFL18] Stephan Friedrichs, Matthias Függer, and Christoph Lenzen. Metastability-Containing Circuits. *IEEE Transactions on Computers*, 2018. To appear, online first.

[Got49] M. Goto. Application of Logical Mathematics to the Theory of Relay Networks (in Japanese). *J. Inst. Elec. Eng. of Japan*, 64(726):125–130, 1949.

[Huf57] David A. Huffman. The Design and Use of Hazard-Free Switching Networks. *J. ACM*, 4(1):47–62, 1957.

[IKL$^+$18] Christian Ikenmeyer, Balagopal Komarath, Christoph Lenzen, Vladimir Lysikov, Andrey Mokhov, and Karteek Sreenivasaiah. On the complexity of hazard-free circuits. In *Symposium on the Theory of Computing (STOC)*, 2018. To appear. Preprint available on arxiv: https://arxiv.org/abs/1711.01904.

[Kle52] Stephen Cole Kleene. *Introduction to Metamathematics*. North Holland, 1952.

[YR64] Michael Yoeli and Shlomo Rinon. Application of Ternary Algebra to the Study of Static Hazards. *J. ACM*, 11(1):84–97, 1964.