Efficient Metastability-Containing Gray Code 2-Sort

Christoph Lenzen* Moti Medina*
*Max Planck Institute for Informatics, Saarbrücken, Germany
{clenzen,mmedina}@mpi-inf.mpg.de

***Abstract*—It is well-established that unsynchronized communication across clock domains can result in metastable upsets and that this cannot be avoided deterministically. This, however, does not preclude the possibility that metastability can be *contained* deterministically, in the sense that meaningful and precise computations can be performed despite metastability of some bits.**

In this work, we provide evidence that this is not only possible, but can also be done *efficiently*. We propose a circuit of size $\mathcal{O}(B^2)$ and depth $\mathcal{O}(B)$ that computes the minimum and maximum of two B-bit Gray code inputs, where each input may contain one metastable bit (introducing uncertainty regarding whether it encodes some value x or rather $x+1$). This is achieved by combining the results of a recursive call on the $(B-1)$-bit suffixes in a metastability-containing way. This overcomes the problem posed by possible metastability of the logic controlling the recursion, which must occur in some executions.

***Index Terms*—metastability worst-case propagation model; sorting networks; combinational circuits;**

I. Introduction

Metastability, the phenomenon of a bistable element entering an unstable third equilibrium state, is a well-known "problem child" in digital circuits. It is also well-known that no method of prevention is absolutely safe: *any* bistable circuit can be subjected to an input resulting in metastability of its output [1].

Fortunately, as metastability is a transient state by definition, simple waiting resolves the problem. The probability that metastability persists for t time falls exponentially in t [2]. If time is a critical resource, synchronizers are used to maximize the prefactor of t in the exponential decay; state-of-the-art synchronizers for 90 nm technology achieve a factor e^{-20} decay per nanosecond; however, there is evidence that the resolution time will *increase* with further miniaturization [3].

Why Synchronizers Are not Always Good Enough

Whenever communicating across clock domains in an unsynchronized fashion, synchronizers are used to ensure a sufficiently small probability of causing metastability of the receiver's logic. This is costly in terms of time (i.e., delay) and space (i.e., buffer size), and can be a limiting factor to performance.

Our main motivation in this paper is the cost in terms of time. This is a pivotal concern when implementing the fault-tolerant clock synchronization algorithm by Lynch and Welch [4], cf. Figure 1. A key step in this synchronization primitive is to measure the phase offsets between the n nodes that synchronize their local time references. Avoiding a fully analog design, this necessitates to resolve potential metastability of the digital subcircuits. However, the quality of synchronization deteriorates linearly in the time required for an individual synchronization step, i.e., in a classic synchronizer-based design the time used to resolve metastability negatively affects performance.

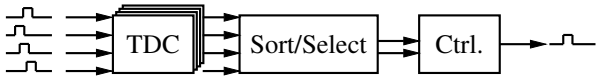

Fig. 1. Fault-tolerant clock synchronization algorithm in hardware. Each of n nodes (i) measures phase differences of each remote clock to its own, (ii) selects for $f < n/3$ the $(f+1)$-th and $(n-f)$-th largest among them, and (iii) derives a correction phase shift.

Challenging the Old Ways

Given that metastability cannot be avoided deterministically, one might conclude that this performance hit is obligatory. *This conclusion is wrong!* It turns out that one can carry out the involved computations in a metastability-containing way: while metastability might infect some of the output registers of a computation, it is nonetheless viable to have a meaningful result for which either interpretation of metastable outputs yields correct and accurate results.

As a concrete example, consider the outputs of two delay line TDCs. These are two unary encoded numbers, and if designed appropriately, each output has at most one metastable bit [5].[1] Taking this into account, one can assume that the (fixed-length) bit strings produced by the TDCs are of the form 1^*0^* or 1^*M0^*, where M denotes a metastable bit. Determining the maximum or minimum of two such values is a trivial task, by determining their bit-wise AND or OR, respectively. The logical masking properties of the AND and OR gates ensure that, in fact, the output is of the same form as the inputs – the precision of the original measurements is fully preserved! This means that we can build 2-sort modules and combine them into sorting networks. In the end, one translates the result of the computation back to an (analog) phase correction applied to the oscillator, so that no harm is done by the metastability incurred by the TDCs. Overall, the

[1] Decreasing the per-stage delay to the point where multiple bits can become metastable does not improve (worst-case) single-shot performance, as metastable registers may arbitrarily stabilize to either 0 or 1.

entire logic of the Lynch-Welch algorithm can be implemented such that metastability of the digital parts of the circuit has no negative impact on the algorithm's performance: the implementation is *metastability-containing*.

While this is exciting from a theoretical perspective, using delay line TDCs and unary encodings in this way does not result in compact circuits. The clock synchronization algorithm requires high-resolution TDCs to be accurate, which can easily result in above 1000 stages for a delay line TDC and the same number of bits in unary encodings of measurement results in this application scenario. The transistor count of a single sorting network would exceed 10^5, so a more efficient design is highly desirable.

However, there is no need to operate on unary encoded values. In concurrent work, we designed a TDC that outputs Gray code instead of unary encoded values. Recall that the defining property of a Gray code is that when counting, each up-count changes exactly one bit. Accordingly, the TDC design ensures that exactly one bit may become metastable, namely the one associated with the incomplete up-count corresponding to a metastable TDC stage. This marks the starting point and main motivation for the present paper, which explores whether one can efficiently perform the sorting operations on Gray code inputs with up to one metastable bit.

Our Contribution

We devise two combinational circuits for computing the maximum and minimum of two Gray code encoded inputs with one metastable bit (making the difference between encoded value x or $x+1$, respectively). The first follows a brute-force approach based on unary encodings and achieves the smaller delay. In accordance with the above discussion, the resulting sorting networks have very high transistor counts; for B-bit inputs, they use $\Theta(2^B)$ gates.

Our second circuit tackles this issue, reducing the number of gates to $\Theta(B^2)$. Its delay is larger, but only by a factor smaller than 3. We consider it highly surprising that this is possible, since any recursive approach must suffer from potential metastability of the logic controlling the recursion (this follows directly from Marino's result [1]).

We are able to overcome this obstacle by utilizing a metastability-containing multiplexer with 2 control bits, which also in case of a metastable control bit selecting between identical input yields stable output. By always executing both possible recursive calls and making sure that they return the same result if the control logic becomes metastable, the multiplexer can handle the case of metastable control inputs.

Concerning our practical goal of decreasing the size of the circuit in comparison to the brute-force approach, for 8-bit inputs and $n = 4$ we reduce the transistor count of the sorting network by almost a factor of 3 (to roughly $20,000$ transistors); for $B = 16$, this gap grows beyond factor 150.

A. Structure of this Article

In Section II, we introduce the used encodings and valid inputs, alongside a crucial characterization of valid Gray code inputs. Section III introduces our circuit model, which is a straightforward generalization of standard binary circuit models to worst-case propagation of metastability. This is followed by formalizing the problem in Section IV, i.e., extending the definition of $\max$ and $\min$ to metastable inputs in a meaningful and precision-preserving manner. Sections V and VI present and analyze the exponential- and polynomial-size solutions, respectively. In Section VII-A, we compare the scaling behavior of the two solutions with respect to B, the number of bits per input. We proceed to compare the scaling behavior when the solutions are employed to construct (optimal) sorting networks for $n \in \{4, 7, 10\}$. Finally, Section VIII concludes the paper.

II. Encodings and Valid Inputs

Due to the potential presence of metastability at the input, we need to carefully choose and make use of suitable encodings. In this section, we formalize the respective notation and summarize basic properties of the encodings.

Reflected Binary Gray Code & Unary Encoding

For $N \in \mathbb{N}$, we abbreviate $[N] := \{0, \dots, N-1\}$. For convenience, bit indices are one-based, e.g, the leftmost bit of a binary string g is $g[1]$, the second is $g[2]$, etc. For $1 \leq i \leq j \leq B$ (where g has B bits), let $g[i:j] := g[i]g[i+1]\dots g[j]$; in case $i > j$, $g[i:j]$ is the empty string.

Definition II.1 (Unary Encoding)**.** *For $x \in [N]$, define its* unary encoding

$$\mathrm{un}_{N-1}(x) := 1^x 0^{N-1-x}\,.$$

Fact II.2. *$\mathrm{un}_{N-1}(\cdot) : [N] \to \{0,1\}^{N-1}$ is injective.*

We denote by $\langle\cdot\rangle_{\mathrm{un}}$ the decoding function of a unary string, i.e., for $x \in [N]$, $\langle \mathrm{un}_{N-1}(x)\rangle_{\mathrm{un}} = x$.

Fact II.3. *For all $x \in [N-1]$, $\mathrm{un}_{N-1}(x)$ and $\mathrm{un}_{N-1}(x+1)$ differ in a single bit only.*

Definition II.4 (B-bit Reflected Binary Gray Code)**.** *Suppose $N = 2^B$ for a given $B \in \mathbb{N}$. For $x \in [N]$, we define the* reflected binary Gray code *of x recursively by*

$$\mathrm{rg}_B(x)[i] := \begin{cases} 0 & \text{if } i = 1 \text{ and } x < N/2 \\ 1 & \text{if } i = 1 \text{ and } x \geq N/2 \\ \mathrm{rg}_{B-1}(x)[i-1] & \text{if } i > 1 \text{ and } x < N/2 \\ \mathrm{rg}_{B-1}(2^B - x - 1)[i-1] & \text{if } i > 1 \text{ and } x \geq N/2 \end{cases}$$

for $i \in \{1, \dots, B\}$.

Fact II.5. *$\mathrm{rg}_B(\cdot) : [N] \to \{0,1\}^B$ is a bijection.*

We denote by $\langle\cdot\rangle_{\mathrm{rg}}$ the decoding function of a Gray code string, i.e., for $x \in [N]$, $\langle \mathrm{rg}_B(x)\rangle_{\mathrm{rg}} = x$.

Fact II.6. *For all $x \in [2^B]$, $\mathrm{rg}_B(x)$ and $\mathrm{rg}_B(x+1)$ differ in a single bit only.*

Metastability Characterization of Valid Strings

In this section we define the set of valid strings that serve as inputs to our combinational circuits. We begin by defining an operator that helps characterize this set of inputs.

Definition II.7 (The $*$ operator)**.** *Given $B \in \mathbb{N}$, define the operator $* : \{0,1,M\}^B \times \{0,1,M\}^B \to \{0,1,M\}^B$ by*

$$\forall i \in \{1,\dots,B\} : (x*y)[i] := \begin{cases} x[i] & \text{if } x[i] = y[i] \\ M & \text{else.} \end{cases}$$

We are now ready to define the set of valid strings in our context. Valid strings arise from the use of a time-to-digital converter (TDC); while it is impossible to avoid metastability completely [1], we assume that at most one bit of each input string is metastable.

Definition II.8 (Valid Strings)**.** *Let $B \in \mathbb{N}$ and $N = 2^B$. Then, the set of* valid unary strings of length $N-1$ *is*

$$\mathcal{S}_{\text{un}}^{N-1} := \text{un}_{N-1}([N]) \cup \bigcup_{x \in [N-1]} \{\text{un}_{N-1}(x) * \text{un}_{N-1}(x+1)\}.$$

Similarly, the set of valid Gray code strings of length B *is*

$$\mathcal{S}_{\text{rg}}^{B} := \text{rg}_B([N]) \cup \bigcup_{x \in [N-1]} \{\text{rg}_B(x) * \text{rg}_B(x+1)\}.$$

By Facts II.3 and II.6, valid strings contain at most one metastable bit, implying that once metastability resolves to either 0 or 1, the resulting string encodes either x or $x+1$. For unary encoding, $\text{un}_{N-1}(x) * \text{un}_{N-1}(x+1) = 1^x M 0^{N-x-2}$; for Gray code, things are more involved, cf. Lemma II.11.

For notational convenience, we extend $\langle\cdot\rangle_{\text{rg}}$ and $\langle\cdot\rangle_{\text{un}}$ so that they decode valid strings, as follows.

Definition II.9 (Extensions of the Decoding Functions)**.** *Let $N = 2^B$ for some $B \in \mathbb{N}$. Denote*

$$[N]_M := \left\{ \frac{z}{2} \,\middle|\, z \in [2N-1] \right\},$$

i.e., the half-integers from the range $[0, N-1]$. For $x \in [N-1]$, we extend $\langle\cdot\rangle_{un}$ to $\mathcal{S}_{\text{un}}^{N-1}$ and $\langle\cdot\rangle_{rg}$ to $\mathcal{S}_{\text{rg}}^{B}$ by

$$\langle \text{un}_{N-1}(x) * \text{un}_{N-1}(x+1) \rangle_{un} := x + \frac{1}{2} \quad \text{and}$$
$$\langle \text{rg}_B(x) * \text{rg}_B(x+1) \rangle_{rg} := x + \frac{1}{2}.$$

Accordingly, we extend un_{N-1} and rg_B to $[N]_M$ by

$$\text{un}_{N-1}\left(x + \frac{1}{2}\right) := \text{un}_{N-1}(x) * \text{un}_{N-1}(x+1) \quad \text{and}$$
$$\text{rg}_B\left(x + \frac{1}{2}\right) := \text{rg}_B(x) * \text{rg}_B(x+1).$$

We stress that these definitions are not meant to indicate that actually half-integers are encoded. Rather, they express that, since resolving metastability may result in either of the two bit strings encoding the adjacent integer values, the string with metastable bit "is in between" its stabilized counterparts.

We make the following observation.

Fact II.10. *$\text{un}_{N-1} : [N]_M \to \mathcal{S}_{\text{un}}^{N-1}$ is bijective with inverse $\langle\cdot\rangle_{un}$ and $\text{rg}_B : [N]_M \to \mathcal{S}_{\text{rg}}^{B}$ is bijective with inverse $\langle\cdot\rangle_{rg}$.*

We already discussed how metastability manifests in unary encodings. For Gray code strings, Definition II.4 generalizes to $[N]_M$ as follows.

Lemma II.11. *Let $x \in [N]_M$. Then*

$$\text{rg}_B(x)[i] = \begin{cases} 0 & \text{if } i = 1 \text{ and } x < (N-1)/2 \\ M & \text{if } i = 1 \text{ and } x = (N-1)/2 \\ 1 & \text{if } i = 1 \text{ and } x > (N-1)/2 \\ \text{rg}_{B-1}(x)[i-1] & \text{if } i > 1 \text{ and } x < (N-1)/2 \\ \text{rg}_{B-1}(2^B - x - 1)[i-1] & \text{if } i > 1 \text{ and } x \geq (N-1)/2. \end{cases}$$

Proof. We prove the statement by induction on B. Both for the base case $B = 1$ and the induction step, for index $i = 1$ the claim follows directly from the definitions. For $i > 1$, the base case is trivial, so consider the step from $B-1$ to B. Observe that if $\text{rg}_B(x)[1] \neq M$, the claim readily follows from the recursive definition of the Gray code and the induction hypothesis.

Hence, suppose $\text{rg}_B(x)[1] = M$. It follows that

$$\text{rg}_B(x)[1] = M \overset{II.7, II.8}{\Leftrightarrow} \text{rg}_B(\lfloor x \rfloor)[1] \neq \text{rg}_B(\lceil x \rceil)[1] \overset{II.4}{\Leftrightarrow} x = \frac{N-1}{2}.$$

We conclude that

$$\text{rg}_B(x)[2:B] \overset{II.8}{=} \text{rg}_B(\lfloor x \rfloor)[2:B] * \text{rg}_B(\lceil x \rceil)[2:B] \overset{II.4, II.7}{=} \text{rg}_{B-1}\left(\frac{N}{2} - 1\right). \qquad \square$$

III. Model

In this paper, we seek to design combinational circuits that accept valid strings as inputs, i.e., we need to specify circuit behavior in face of metastability. In our model, gates propagate metastability to their outputs in a worst-case manner, but taking into account logical masking: an AND gates always outputs 0 if one of its inputs is stable 0, and an OR gate always outputs 1 if one of its inputs is stable 1.

TABLE I
Logical extensions to metastable inputs of an AND gate (left), an OR gate (center), and an inverter (right).

a \ b	0	1	M
0	0	0	0
1	0	1	M
M	0	M	M

a \ b	0	1	M
0	0	1	M
1	1	1	1
M	M	1	M

a	$\bar{a}$
0	1
1	0
M	M

A. Complexity Measures: Cost and Delay

We use standard cost and delay complexity measures. Concretely, the *cost* of a combinational circuit is the sum of the costs of its basic gates, and the *delay* of a combinational circuit is the heaviest path from an input to an output, where the weight of a path is the sum of the weights of its gates. We

TABLE II
COST AND DELAY OF BASIC GATES.

	Gate cost	# Trans.	Unit delay	Norm. Delay
NOT	1	2	1	1
OR	1	6	1	3
AND	1	6	1	3
XOR	1	8	1	2
OR(3)	1	8	1	4

assume the values given in Table II for cost and delay of basic gates, but remark that the results will not change substantially for different libraries.

IV. Problem Definition

Our goal is to compute the maximum or minimum of two valid strings, where we extend the definition of $\max$ and $\min$ to valid strings in the following natural way.

Definition IV.1. *Let $N = 2^B$ for some $B \in \mathbb{N}$. For $u, v \in \mathcal{S}_{\mathrm{un}}^{N-1}$, define*

$$\max_{\mathrm{un}}\{u, v\} := \mathrm{un}_{N-1}(\max\{\langle u\rangle_{un}, \langle v\rangle_{un}\})$$
$$\min_{\mathrm{un}}\{u, v\} := \mathrm{un}_{N-1}(\min\{\langle u\rangle_{un}, \langle v\rangle_{un}\})\,.$$

For $g, h \in \mathcal{S}_{\mathrm{rg}}^B$, define

$$\max_{\mathrm{rg}}\{g, h\} := \mathrm{rg}_B(\max\{\langle g\rangle_{rg}, \langle h\rangle_{rg}\})$$
$$\min_{\mathrm{rg}}\{g, h\} := \mathrm{rg}_B(\min\{\langle g\rangle_{rg}, \langle h\rangle_{rg}\})\,.$$

Note that this definition means that the results are valid strings as well. Moreover, the input has precision 1, in the following sense. Suppose a valid string "encodes" $x+1/2$ for some $x \in [N]$, i.e., the string contains a metastable bit that makes it uncertain whether the represented value is x or $x+1$. In this case the TDC generating this string has measured a time period corresponding to at least x and at most $x+1$ stages. The string will stabilize to either x or $x+1$. The stabilized string is thus off by at most 1 w.r.t. the time period it represents. Observe that we impose the same constraint on the extensions of $\max$ and $\min$ to valid inputs. Therefore, these definitions require to fully "contain" metastability, i.e., to not lose any precision due to metastable upsets.

Observe that computing $\max_{\mathrm{un}}$ and $\min_{\mathrm{un}}$ is trivial, because of the masking properties provided by the basic gates readily match our requirements, formalized in the following fact.

Fact IV.2. *For $u, v \in \mathcal{S}_{\mathrm{un}}^{N-1}$, $\max_{\mathrm{un}}\{u, v\} = u + v$ and $\min_{\mathrm{un}}\{u, v\} = u \cdot v$, where $+$ and $\cdot$ denote the bitwise* OR *and* AND*, respectively.*

Our goal is to find circuits computing $\max_{\mathrm{rg}}$ and $\min_{\mathrm{rg}}$.

Definition IV.3. *For $B \in \mathbb{N}$, a metastability-containing Gray code $2\text{-sort}(B)$ combinational circuit is defined as follows.*

- **Input:** $g, h \in \mathcal{S}_{\mathrm{rg}}^B$,
- **Output:** $g', h' \in \mathcal{S}_{\mathrm{rg}}^B$,
- **Functionality:** $g' = \max_{\mathrm{rg}}\{g, h\}$ *and* $h' = \min_{\mathrm{rg}}\{g, h\}$.

V. Exponential Cost, Linear Delay Circuit

Our first solution follows a brute-force approach:

(1) Determine the unary encodings of each input.
(2) Take the bitwise OR or AND, respectively (cf. Fact IV.2).
(3) Determine the Gray code of the results.

The third step is straightforward, exploiting that each bit of the unary encoding affects only a single bit of the Gray code. Hence, we do not need to worry about containing metastability and may simply use XOR trees (one for each output bit) for the conversion, resulting in fewer than N additional gates for $\max_{\mathrm{rg}}$ and $\min_{\mathrm{rg}}$ each; the depth is smaller than $\log N$.

Hence, it remains to design a circuit that determines the unary encoding of a given valid B-bit Gray code string. Our goal in this section is to implement such a circuit with $\mathcal{O}(N) = \mathcal{O}(2^B)$ gates and combinational logic of depth $\mathcal{O}(B)$ in a metastable-containing fashion. The formal specification of the required circuit is as follows:

Definition V.1. *Let $N = 2^B$, where $B \in \mathbb{N}$. A metastability-containing $\mathrm{rg2un}(B)$ circuit meets the following specification.*

- **Input:** $g \in \mathcal{S}_{\mathrm{rg}}^B$,
- **Output:** $u \in \mathcal{S}_{\mathrm{un}}^{N-1}$,
- **Functionality:** $u = \mathrm{un}_{N-1}(\langle g\rangle_{rg})$

The following structural lemma will give rise to an efficient $\mathrm{rg2un}(B)$ circuit.

Lemma V.2. *For $g \in \mathcal{S}_{\mathrm{rg}}^B$ and $i \in \{1, \ldots, N-1\}$,*

$$\mathrm{un}_{N-1}(\langle g\rangle_{rg})[i] =$$
$$\begin{cases} g[1] + \mathrm{un}_{N/2-1}(\langle g[2:B]\rangle_{rg})[i] & \text{if } i < \frac{N}{2} \\ g[1] & \text{if } i = \frac{N}{2} \\ g[1] \cdot \left(\neg\,\mathrm{rev}(\mathrm{un}_{N/2-1}(\langle g[2:B]\rangle_{rg}))\left[i - \frac{N}{2}\right]\right) & \text{if } i > \frac{N}{2}\,, \end{cases}$$

where

$$\mathrm{rev}(x) := x[N-1]x[N-2]\ldots x[1] \quad \textit{and}$$
$$\neg x := (1 - x[1])(1 - x[2])\ldots(1 - x[N-1])\,.$$

Proof. Let $x := \langle g\rangle_{\mathrm{rg}}$. We make a case distinction by $g[1]$.
Case $g[1] = 0$**:** By Lemma II.11, $x \leq N/2 - 1$ and $g[2:B] = \mathrm{rg}_{B-1}(x)$. The claim follows, as $\mathrm{un}_{N-1}(x) = \mathrm{un}_{N/2-1}(x)0^{N/2}$ for $x \leq N/2 - 1$.
Case $g[1] = 1$**:** By Lemma II.11, $x \geq N/2$ and $g[2:B] = \mathrm{rg}_{B-1}(N - x - 1)$. Thus,

$$\begin{aligned} &\neg\,\mathrm{rev}(\mathrm{un}_{N/2-1}(\langle g[2:B]\rangle_{\mathrm{rg}})) \\ \overset{II.4}{=}\ &\neg\,\mathrm{rev}\left(\mathrm{un}_{N/2-1}(\langle \mathrm{rg}_{B-1}(N - x - 1)\rangle_{\mathrm{rg}})\right) \\ =\ &\neg\,\mathrm{rev}\left(\mathrm{un}_{N/2-1}(N - x - 1)\right) \\ =\ &\mathrm{un}_{N/2-1}(x - N/2)\,. \end{aligned}$$

The claim follows, as $\mathrm{un}_{N-1}(x) = 1^{N/2}\,\mathrm{un}_{N/2-1}(x - N/2)$ for $x \geq N/2$.
Case $g[1] = M$**:** $x = (N-1)/2$ and $g[2:B] = \mathrm{rg}_{B-1}(N/2 - 1)$ by Lemma II.11. Thus, $\mathrm{un}_{N/2-1}(\langle g[2:B]\rangle_{\mathrm{rg}}) = 1^{N/2-1}$. The claim follows, as $\mathrm{un}_{N-1}(x) = 1^{N/2-1}M0^{N/2-1}$. □

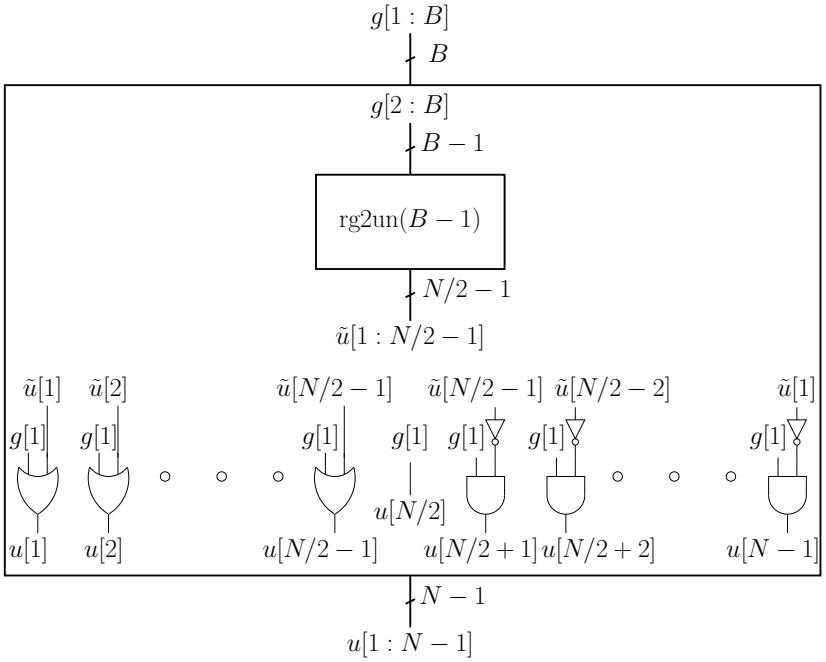

Fig. 2. Recursive implementation of $\mathrm{rg2un}(B)$ derived from Lemma V.2.

Lemma V.2 yields a straightforward recursive circuit of size $\mathcal{O}(N)$ that constructs $\mathrm{un}_{N-1}(\langle g \rangle_{\mathrm{rg}})$ from B-bit Gray code inputs g, depicted in Figure 2.

Corollary V.3. *The combinational circuit depicted in Figure 2 implements the specification of* $\mathrm{rg2un}(B)$ *from Definition V.1. The delay and cost of this circuit are:*

$$\mathrm{delay}(\mathrm{rg2un}(B)) = B \cdot (\mathrm{delay}(\mathrm{AND}) + \mathrm{delay}(\mathrm{NOT})) \quad (1)$$

$$\mathrm{cost}(\mathrm{rg2un}(B)) = \left(2^B - B - 1\right)(2\,\mathrm{cost}(\mathrm{AND}) + \mathrm{cost}(\mathrm{NOT})) \quad (2)$$

Proof. Correctness follows from Lemma V.2. As delay and cost of AND and OR gates are identical, delay and cost admit the following recurrences:

$$\begin{aligned}
\mathrm{delay}(\mathrm{rg2un}(0)) &= 0\\
\mathrm{delay}(\mathrm{rg2un}(B)) &= \mathrm{delay}(\mathrm{rg2un}(B-1)) +\\
&\qquad \mathrm{delay}(\mathrm{AND}) + \mathrm{delay}(\mathrm{NOT})\\
\mathrm{cost}(\mathrm{rg2un}(0)) &= 0\,,\\
\mathrm{cost}(\mathrm{rg2un}(B)) &= \mathrm{cost}(\mathrm{rg2un}(B-1)) +\\
&\qquad \left(2^{B-1} - 1\right)(2\,\mathrm{cost}(\mathrm{AND}) + \mathrm{cost}(\mathrm{NOT}))
\end{aligned}$$

□

Remark V.4. *The fan-out of the input bit* $g[i]$ *is* $N/2^{i-1} - 1$*, where* $1 \le i \le \log N = B$*. This can be reduced to a constant fan-out by standard techniques, e.g., buffer/inverter trees. This increases the cost of the circuit by less than a factor of* 2*, without affecting the depth.*

Overall, we arrive at an implementation of $2\text{-sort}(B)$ with the following properties.

Theorem V.5 (Brute-force Approach)**.** *There is a combinational circuit implementing the specification of* $2\text{-sort}(B)$ *given in Definition IV.3. Its delay and cost are:*

$$\mathrm{delay}(2\text{-sort}(B)) = \mathrm{delay}(\mathrm{rg2un}(B)) + \mathrm{delay}(\mathrm{AND}) + \mathrm{delay}(\mathrm{XOR}) \cdot (B-1) \quad (3)$$

$$\mathrm{cost}(2\text{-sort}(B)) = 2 \cdot (\mathrm{cost}(\mathrm{rg2un}(B)) + (2^B - 1) \cdot \mathrm{cost}(\mathrm{AND}) + (2^B - B - 1) \cdot \mathrm{cost}(\mathrm{XOR})) \quad (4)$$

VI. Polynomial Cost, Linear Delay Circuit

In the previous section, we saw that a metastability-containing comparator circuit of size $\mathcal{O}(2^B)$ exists for B-bit reflected binary Gray code inputs. However, exponential-size circuits are not practical for more than a few input bits. Also, this solution seems fairly pointless: What good does it do for the TDC to produce an efficient representation like a Gray code, if we return to unary encoding for meaningful computation? It would be preferable to stick to unary encodings, for which computing $\max_{\mathrm{un}}$ and $\min_{\mathrm{un}}$ is trivial, even when facing metastability.

In this section, we improve on the brute-force approach and obtain a circuit of size $\mathcal{O}(B^2)$ and depth $\mathcal{O}(B)$.

High-Level Description

We devise a recursive scheme that can directly operate on the Gray code representation. To avoid "amplifying" metastability by "infecting" bits of the output that can be correctly computed, it cannot readily branch on the first bit of the code: if the control logic becomes metastable, the branches may produce metastable outputs without need, which ultimately would corrupt the output irreversibly.

We avoid this problem by considering all possible stable values of the first bits of the two inputs g and h (00, 01, 10, and 00), determining the respective outcomes, and then selecting from these results in a safe way. Roughly speaking, if, say, $h[1] = M$, then Lemma II.11 shows that $h[2 : G] = \mathrm{rg}_{B-1}(N/2 - 1)$, and we can exploit this to show that the recursive call computes the same results both for the $h[1] = 0$ and the $h[1] = 1$ branch. Hence, "all" we need to do is to feed the results into a multiplexer that uses $g[1]$ and $h[1]$ as control bits and has the property that if the inputs between which a metastable control bit selects are identical, the output is stable.

Standard multiplexer circuits do not have this property, so the first step is to devise such a *metastability-containing multiplexer (*CMUX*).*

A. Metastability-containing Multiplexers

The logic table and an implementing circuit of a $(2 : 1) -$ CMUX (i.e., selecting between two inputs using one control bit) are depicted in Figure 3. The vital difference to a standard multiplexer is that it outputs $a * b$ in case the control bit s is metastable; a standard multiplexer may yield metastable output if s is metastable, even if a and b agree, i.e., the selection does not actually matter. We use the convention that a CMUX that selects among inputs of length B each is denoted by CMUX(B) (as it is simply B copies of a CMUX).

s / ab	0	1	M
00	0	0	0
01	0	1	M
10	1	0	M
11	1	1	1
0M	0	M	M
M0	M	0	M
1M	1	M	M
M1	M	1	M
MM	M	M	M

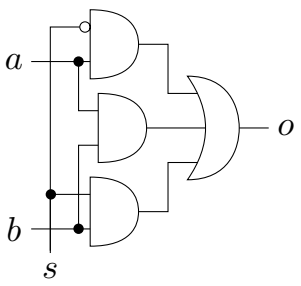

Fig. 3. Top: Output of a metastability-containing $(2 : 1) -$ CMUX, whose single control bit s is used to select between two inputs a and b. Bottom: Implementation of a $(2 : 1) -$ CMUX using AND gates and an OR(3) gate.

We generalize this principle to 2 control bits, i.e., selection between 4 inputs. This is done in a straightforward way, as depicted in Figure 4.

Lemma VI.1. *The circuit shown in Figure 4 implements the specification of a metastability-containing* $(4 : 1) -$ CMUX *given in the logic table.*

Proof. By checking all cases, first for the 1-bit multiplexer given in Figure 3, and then for the $(4 : 1) -$ CMUX based on the specification of a 1-bit multiplexer. □

B. 2-sort(B) Implementation

We are now ready to implement the high-level idea outlined earlier. We show that using the first bits of the two Gray code inputs g and h as control bits, we can feed suitable inputs to the CMUX to determine the correct output recursively.

s / t	0	1	M
0	a	b	a ∗ b
1	c	d	c ∗ d
M	a ∗ c	b ∗ d	a ∗ b ∗ c ∗ d

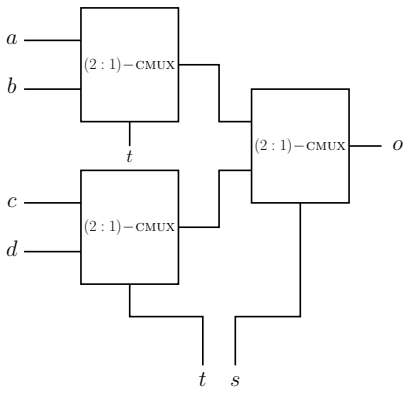

Fig. 4. Top: output of a $(4 : 1) -$ CMUX. The control bits are s and t, the selectable inputs are a, b, c, d. Bottom: Implementation using $(2 : 1) -$ CMUX.

Lemma VI.2. *Let* $g, h \in \{0, 1\}^B$. *Then, for* $i \in \{1, \ldots, B\}$,

$$\max_{\mathrm{rg}}\{g, h\}[i] = \begin{cases} g[1] + h[1] & \text{if } i = 1 \\ f_{B-1}(g, h)[i] & \text{otherwise} , \end{cases}$$

where $f_{B-1}(g, h)$ *is the output of a* $(4 : 1) -$ CMUX *with inputs*

$$\begin{aligned} a &= \max_{\mathrm{rg}}\{g[2 : B], h[2 : B]\} \\ b &= h[2 : B] \\ c &= g[2 : B] \\ d &= \min_{\mathrm{rg}}\{g[2 : B], h[2 : B]\} \\ s &= g[1] \\ t &= h[1] . \end{aligned}$$

Proof. From Lemma II.11 and Definition IV.1, we see that $\max_{\mathrm{rg}}\{g, h\}[1] = g[1] + h[1]$. Hence, we need to show that $f_{B-1}(g, h) = \max_{\mathrm{rg}}\{g, h\}[2 : B]$, which we show by induction on B. The base case $B = 1$ being trivial, consider the step from $B - 1$ to $B > 1$. If $g[1], h[1] \in \{0, 1\}$, checking the cases using Lemma II.11, Definition IV.1, and the induction hypothesis, the claim readily follows.

Hence, assume w.l.o.g. that $h[1] = M$. By Lemma II.11 it follows that $h[2 : B] = \mathrm{rg}_{B-1}(N/2-1)$, i.e., $h[2 : B]$ encodes the maximum value that can be represented by a $(B - 1)$-bit Gray code. Therefore,

$$\max_{\mathrm{rg}}\{g[2 : B], h[2 : B]\} = h[2 : B] \quad \text{and} \tag{5}$$

$$\min_{\mathrm{rg}}\{g[2 : B], h[2 : B]\} = g[2 : B] . \tag{6}$$

We distinguish three cases, based on $g[1]$.

Case $g[1] = 0$**:** By Lemma II.11, $\langle g \rangle_{\mathrm{rg}} \leq N/2-1 < \langle h \rangle_{\mathrm{rg}}$, i.e., $\max_{\mathrm{rg}}\{g, h\} = h$. By the specification of the $(4 : 1) -$ CMUX, $f_{B-1}(g, h) = a * b = h[2 : B]$, as $a = b = h[2 : B]$ by (5).

Case $g[1] = 1$**:** By Lemma II.11, $\langle g \rangle_{\mathrm{rg}} \geq N/2 > \langle h \rangle_{\mathrm{rg}}$, i.e., $\max_{\mathrm{rg}}\{g, h\} = g$. By the specification of the $(4 : 1) -$ CMUX, $f_{B-1}(g, h) = c * d = g[2 : B]$, as $c = d = g[2 : B]$ by (6).

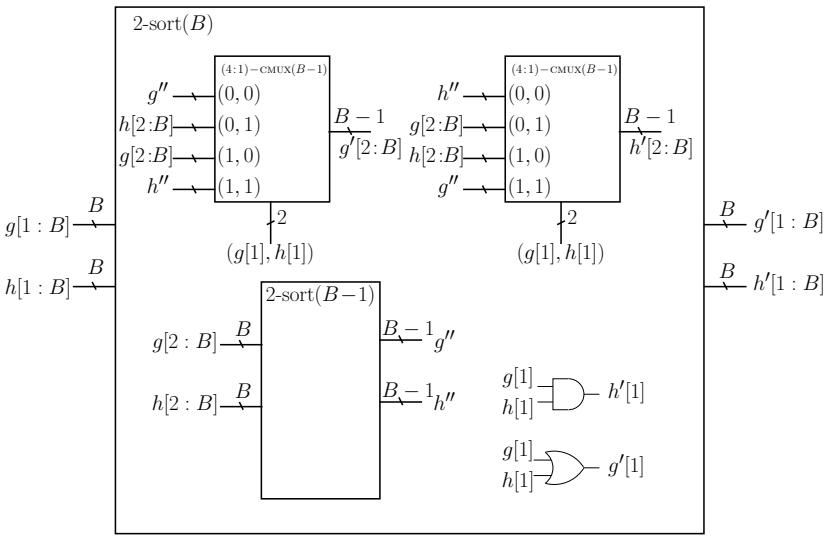

Fig. 5. Recursive implementation of 2-sort(B) derived from Lemmas VI.2 and VI.3.

Case $g[1] = M$**:** By Lemma II.11, $\langle g \rangle_{\mathrm{rg}} = (N-1)/2 = \langle h \rangle_{\mathrm{rg}}$, i.e., $g = h$. In particular, $\max_{\mathrm{rg}}\{g[2:B], h[2:B]\} = \min_{\mathrm{rg}}\{g[2:B], h[2:B]\} = g[2:B] = h[2:B]$, i.e., we have that $a = b = c = d$. By the specification of the $(4:1)-$CMUX, $f_{B-1}(g,h) = a * b * c * d = \max_{\mathrm{rg}}\{g,h\}[2:B]$. □

An analogous statement applies to $\min_{\mathrm{rg}}$.

Lemma VI.3. *Let $g, h \in \{0,1\}^B$. Then, for $i \in \{1, \ldots, B\}$,*

$$\min_{\mathrm{rg}}\{g,h\}[i] = \begin{cases} g[1] \cdot h[1] & \textit{if } i = 1 \\ f'_{B-1}(g,h)[i] & \textit{otherwise}\,, \end{cases}$$

*where $f'_{B-1}(g,h)$ is the output of a $(4:1)-$*CMUX *with inputs*

$$\begin{aligned} a &= \min_{\mathrm{rg}}\{g[2:B], h[2:B]\} \\ b &= g[2:B] \\ c &= h[2:B] \\ d &= \max_{\mathrm{rg}}\{g[2:B], h[2:B]\} \\ s &= g[1] \\ t &= h[1]\,. \end{aligned}$$

Leveraging this recursive structure, we obtain a different 2-sort(B) implementation (see Figure 5), whose properties are summarized in the following theorem.

Theorem VI.4. *The circuit depicted in Figure 5 implements the specification of* 2-sort(B) *given in Definition IV.3. The delay and cost of this circuit are:*

$$\begin{aligned} \mathrm{delay}(\text{2-sort}(B)) &= (B-1) \cdot (\mathrm{delay}(\mathrm{AND}) + \\ &\qquad 2 \cdot \mathrm{delay}(\textsc{cmux})) + \mathrm{delay}(\mathrm{AND})\,, \quad (7) \\ \mathrm{cost}(\text{2-sort}(B)) &= 3 \cdot (B-1)^2 \cdot \mathrm{cost}(\textsc{cmux}) + \\ &\qquad B \cdot (\mathrm{cost}(\mathrm{AND}) + \mathrm{cost}(\mathrm{OR}))\,. \quad (8) \end{aligned}$$

TABLE III
SUMMARY OF COST AND DELAY OF THE IMPLEMENTATIONS OF 2-sort(B) GIVEN IN THEOREM V.5 (EXP.) AND THM. VI.4 (POLY.).

B	Circuit	# Gates	# Trans.	Delay
$B=2$	Exp.	14	80	13
	Poly.	19	108	22
$B=4$	Exp.	118	664	25
	Poly.	143	804	60
$B=8$	Exp.	2486	13928	49
	Poly.	751	4212	136
$B=16$	Exp.	655222	3669256	87
	Poly.	3407	19092	288

Proof. Correctness follows from Lemmas VI.2 and VI.3. Cost and delay admit the following recurrences:

$$\begin{aligned} \mathrm{delay}(1) &= \mathrm{delay}(\mathrm{AND})\,, \\ \mathrm{delay}(B) &= \mathrm{delay}(B-1) + \\ &\qquad 2 \cdot \mathrm{delay}(\textsc{cmux}) + \mathrm{delay}(\mathrm{AND}) \\ \mathrm{cost}(1) &= 2 \cdot \mathrm{cost}(\mathrm{AND})\,, \\ \mathrm{cost}(B) &= \mathrm{cost}(B-1) + 2 \cdot 3 \cdot (B-1) \cdot \mathrm{cost}(\textsc{cmux}) + \\ &\qquad 2 \cdot \mathrm{cost}(\mathrm{AND})\,. \end{aligned}$$

□

VII. COMPARISON BETWEEN OUR TWO IMPLEMENTATIONS

Table III shows the delay and cost for the two implementations for the 2-sort(B) presented in this paper, neglecting the issue of large fanout for the exponentially sized solution. Note that the effect of reducing the fanout is small for the polynomially sized solution: each bit has fanout $O(B)$, and no large fanouts appear on the critical paths determining the delay. Hence, the comparison is in favor of the exponentially sized circuit w.r.t. gate and transistor counts, while it slightly favors the polynomially sized solution w.r.t. delay. This makes the results simple to interpret: the exponentially sized solution

TABLE IV
GATE AND TRANSISTOR COUNTS FOR METASTABILITY-CONTAINING SORTING NETWORKS WITH $n \in \{4, 7, 10\}$ B-BIT INPUTS DERIVED FROM OUR IMPLEMENTATIONS OF 2-sort(B).

B	Circuit	4-sort		7-sort		10-sort	
		#Gates	#Trans.	#Gates	#Trans.	#Gates	#Trans.
$B = 2$	Exp.	65	370	179	1010	311	1750
	Poly.	95	540	304	1728	551	3132
$B = 4$	Exp.	535	2990	1393	7654	2377	12986
	Poly.	715	4020	2288	12864	4147	23316
$B = 8$	Exp.	11195	62230	28661	156158	48629	263122
	Poly.	3755	21060	12016	67392	21779	122148
$B = 16$	Exp.	2948515	16380710	7535197	41017966	12777133	69062594
	Poly.	17035	95460	54512	305472	98803	553668

has a factor 2 to 3 smaller delay, but the polynomially sized solution has dramatically smaller transistor counts even for moderate $B = 8$.

Note that for, e.g., $B = 4$, the polynomial-size solution provides little gain over the exponential-size solution (even after taking into account that fanout needs to be reduced). Thus, it can be beneficial to use hybrid solutions, where a few bits are handled by the brute-force solution. This will reduce the depth of the circuit without increasing transistor counts noticeably.

A. Application: Sorting Networks of Valid Gray Code Strings

We now consider the application of these two implementations in the context of sorting networks. Suppose the sorting network has n channels, i.e., we sort n strings. The inputs are valid Gray code strings of length B. The output of the sorting network are the n input strings, sorted according to the order induced by $g < h \Leftrightarrow \langle g \rangle_{\mathrm{rg}} < \langle h \rangle_{\mathrm{rg}}$.

Codish et. al [6] showed the optimality of previous designs of sorting networks [7], [8]. Hence, together with the known optimality of Knuth's sorting networks for networks with up to 8 channels [9], the optimality of sorting networks with up to 10 channels is known.

In our context, we are specifically interested in sorting networks with $n = 3f + 1$ channels for some $f \in \mathbb{N}$, as this is the minimum number of nodes required to tolerate f faulty nodes in the clock synchronization by Lynch and Welch [4]. In Table IV, we list sorting networks with 4, 7, and 10 channels, for which the optimal implementation uses $5, 16$ and 29 modules of 2-sort(B) circuits, respectively.

Although the brute-force implementation is more costly in terms of transistors, we note that the translation from Gray code to unary encoding and vice versa needs to be done only once for each input. This reduces the transistor count for the brute-force solution notably, which we took into account in Table IV. Thus, in particular for larger values of n, the polynomial size solution is outperformed for small values of B of roughly up to 4. For $B \geq 8$, the asymptotics clearly kick in and result in a dramatic gap for all considered values of n.

VIII. DISCUSSION

In this paper we considered metastability-containing combinational circuits sorting Gray code inputs. Our input specification aligns with the outputs generated by a TDC, allowing for computations to take place *before* metastability is resolved. In fact, this enables to perform all computations required by the Lynch-Welch synchronization algorithm [4] without the need for synchronizers. The resulting circuit *deterministically* contains metastability, without any loss of precision.

We devised a rigorous formalism that enabled us to prove the correctness of our designs in the context of metastability. We believe that this theoretical basis will prove useful beyond the applications considered in this paper.

The main open questions this paper raises are:

- What is the optimum cost of the 2-sort(B) primitive?
- What is the minimum delay of the 2-sort(B) primitive?
- Are better trade-offs feasible than achieved in this work?
- Are there other powerful basic primitives that can be efficiently realized in a metastability-containing way?

ACKNOWLEDGEMENTS

We thank Matthias Függer for proposing the metastability-containing $(2:1) -$ CMUX given in Figure 3.

REFERENCES

[1] L. Marino, "General Theory of Metastable Operation," *IEEE Transactions on Computers*, vol. C-30, no. 2, pp. 107–115, 1981.

[2] C. Dike and E. Burton, "Miller and Noise Effects in a Synchronizing Flip-Flop," *Solid-State Circuits*, vol. 34, no. 6, pp. 849–855, 1999.

[3] S. Beer, R. Ginosar, M. Priel, R. R. Dobkin, and A. Kolodny, "The Devolution of Synchronizers," in *Proc. Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2010, pp. 94–103.

[4] J. L. Welch and N. A. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, 1988.

[5] R. B. Staszewski, S. Vemulapalli, P. Vallur, J. Wallberg, and P. T. Balsara, "1.3 V 20 ps Time-to-Digital Converter for Frequency Synthesis in 90-nm CMOS," *IEEE Trans. on Circuits and Systems*, vol. 53, no. 3, pp. 220–224, 2006.

[6] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp, "Twenty-five comparators is optimal when sorting nine inputs (and twenty-nine for ten)," in *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*. IEEE, 2014, pp. 186–193.

[7] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. ACM, 1968, pp. 307–314.

[8] D. E. Knuth, "The art of computer programming vol. 3: Sorting and searching," 1998.

[9] R. W. Floyd and D. E. Knuth, "The bose-nelson sorting problem," *A survey of combinatorial theory*, pp. 163–172, 1973.