Digital Circuit Design for Biological and Silicon Computers

Matthias Függer, Manish Kushwaha, Thomas Nowak

Digital Circuit Design for Biological and Silicon Computers

Matthias Függer, Manish Kushwaha, and Thomas Nowak*

1 Introduction

Evolving for the past 4 billion years [3], life on Earth today is highly diverse with an estimated 8.7 million species [21]. Despite this enormous diversity, one key characteristic that differentiates all living cells from non-living matter is their property of response to stimuli [14]. Living cells receive information from their environment, process that information, and then effect a response. Therefore, in the simplest "information processing" sense of computing [30], cells can be seen as tiny computing machines. This notion has sustained itself through Monod to today as the successes of molecular and cell biology have continued to reveal insights into the mechanistic bases of cellular functioning [18, 23]. Conversely, the similarities in information flow between living systems and computing devices have also led to the design and construction of synthetic biological circuits inspired from electronic circuitry [29].

In this chapter, we explore the similarities and differences between computation by biological and silicon computers. We first describe the nature of information in the two system types. Next, we outline how the information flow can be represented as chemical reaction networks or communicating hardware processes. Finally, we define the kind of computational problems and the circuits that can be used to implement their solutions.

Matthias Függer
CNRS, LSV, ENS Paris-Saclay, Université Paris-Saclay, Inria, e-mail: mfuegger@lsv.fr

Manish Kushwaha
Micalis Institute, INRA, AgroParisTech, Université Paris-Saclay, Jouy-en-Josas, France, e-mail: manish.kushwaha@inra.fr

Thomas Nowak
Université Paris-Sud, CNRS, e-mail: thomas.nowak@lri.fr

* All authors contributed equally. Names are listed in alphabetical order.

1.1 Analog versus digital computation

Most information in the physical world is analog in nature, varying between low and high values on a continuous scale [27]. This is true not just of biologically relevant signals like metabolite concentration, heat, light, pH, and osmotic pressure, but also of electrical signals that run through human-made electronic circuits. Current and voltage in electronic circuits can vary across their different parts depending on other circuit properties like material (doping profiles) and geometry [1]. For both biological and electronic circuits special processing of analog signals can be used to generate their digital abstractions [33], using analog components that favor a binary all-or-nothing behavior. Many such circuits have been implemented over the past two decades inside living cells [20].

Before being applied to biological circuitry, however, this special treatment of analog signals was exploited for building digital computer architectures [1]. The main reason for the success of the digital abstraction in computer circuit designs is its tolerance to noise. By introducing a threshold voltage that separates two voltage zones corresponding to the binary digits 0 and 1, some noise in the signal can be tolerated as long as it does not force the signal outside of the correct zone. In contrast, it is impossible to build an analog adder circuit that outputs the exact sum of two voltages and that tolerates *any* substantial amount of noise. The digital abstraction also significantly simplifies the specifications of circuit interfaces and thus the reusability of components. Additionally, it enables the use of techniques and results from Boolean logic to further speed up and simplify the design process and the development of design automation tools.

1.2 Natural biological computation

Natural biological systems process several continuous signals in their extra- and intracellular environments to make decisions [33]. Bacterial cells sense gradients of chemical signals to modulate their movements to swim towards chemo-attractants and away from chemo-repellants [31]. Similarly, they make auxotrophic decisions on upregulating and downregulating different parts of their metabolism depending on the available nutrients [19]. Some bacteria use environmental cues to switch between different life cycle stages [25]. In developmental pathways, cells of multicellular organisms make differentiation decisions based on gradients of mRNA molecules across their body length [32]. In summary, information processing in natural biological systems seems to use hybrid approaches that combine aspects of analog and digital circuitry. While the analog parts of the circuitry process and output continuous signals, digital parts are used for decision-making between discrete options [4], often for steps that require substantial downstream commitment. This hybrid architecture appears to be much more powerful and has been applied in bio-inspired design of computational methods, for example in neural computing

where perceptrons use analog logic to generate weighted sums and subsequently a digital activation function to reach a classification decision [26].

1.3 Synthetic biological computation

In contrast to natural systems that have optimized analog-digital hybrid architectures over evolutionary time scales, rational design of synthetic biological circuits in the past two decades has focused mainly on building digital logic circuitry due to its relative ease of implementation [33]. This has involved building one or more levels of NOT, AND, OR, NAND, and NOR gates in single cells or across multicellular consortia. Implementation has been done at different layers of biological information processing: the gene expression layer of transcription and translation, the metabolic layer of enzymatic reactions, and the signal transduction layer of small molecules and their sensors [24, 11, 13]. Applications of these circuits have ranged from detection of pollutants and diagnostic biomarkers, to smart therapeutics and metabolic engineering [12]. Despite the extensive work in biological circuit design, the process remains quite *ad hoc* with many manual steps needed for prototyping and testing. However, new tools for computer-aided design are beginning to show promise for the future of design automation [24].

2 Representation

This section discusses two different symbolic representations of circuits, one from microbiology (chemical reaction networks) and one from electrical circuits (CHP). Symbolic representations, even when similar at a superficial level, can have a profound impact not only for the computer tools that manipulate these representations but also for the ease of understanding and expression for the end user. The persistent community discussions to identify the "best" programming language bears witness to the immense value of symbolic representation. The importance of precise and workable representations is well-understood in the synthetic biology community. Consequently, a number of (visual) representational standards are already being tested [10, 17].

2.1 Chemical Reaction Networks: A Language for Biochemical Reactions

A chemical reaction network specifies a set of reactions involving a set of molecules. Each reaction defines which molecules, and in which quantities, it takes as its input and which molecules, and in which quantities, it produces as its output. Additionally,

each reaction has a rate, which determines how often the reaction occurs in a given time interval.

For instance, the reaction $A + B \xrightarrow{\alpha} C + D + E$ takes one unit of A, one unit of B, and produces one unit each of C, D, and E at the rate α. The frequency at which the reaction happens is determined not only by the rate α, but also by the concentrations of A and B. Denoting the concentrations of A and B by $[A]$ and $[B]$, respectively, the propensity of the above reaction is equal to $\alpha \cdot [A] \cdot [B]$.

Different operational interpretations of chemical reaction networks exist. The two most common ones are the ordinary differential equation (ODE) model and the stochastic model. In the ordinary differential equation model, each molecule has a real-valued continuous concentration whose derivative is equal to the sum of the propensities of reactions that produce the molecule minus the sum of the propensities of reactions that consume it. In the stochastic model, concentrations of molecules are measured as integer-valued counts (normalized to volume), and a reaction's next occurrence time is randomly chosen according to an exponential random variable with parameter equal to the reaction's propensity [8]. The deterministic ODE model can be interpreted as the mean field solution of the stochastic model, that is, it approximates the behavior of the expected value. Figure 3 (right) shows ODE and stochastic traces with two different molecule numbers: the higher the number of involved molecules the less likely it becomes for the stochastic traces to differ greatly from the expected ODE trace.

2.2 CHP: A Language for Digital Circuit Design

Hardware description languages like Verilog and VHDL provide means to specify the behavior of a circuit not only as a structural composition of basic gates, but also by abstract semantics as they are known from programming languages for software (comprising constructs like loops, conditional execution, etc.). As such, Verilog and VHDL have evolved as standards for design entry and exchange format for circuits *in silico*, but recently also for microbiological combinational circuits [24].

Throughout this chapter, we will make use of a semantically simpler hardware description language that has recently gained attention, the CHP (communicating hardware processes) language. Its use in a comparative article like this is motivated by two facts: (a) the language emphasizes a circuit's nature as a collection of simple but highly concurrently operating components and (b) we will argue in Section 4 that it is closely related to, although being an abstraction of, the reaction based notation used for the design and modeling of microbiological pathways.

CHP programs. Introduced by Martin in 1989 [16], CHP is a language that adapts the concept of communicating sequential processes [9], originally developed for sequential software modules that are executed in parallel and communicate with each other, for hardware processes. In this notation a circuit component is one of the following terms that indicates (i) a *variable assignment* $x := x'$ with x being a variable and x' being its assigned value (0 or 1, or any Boolean formula which may

include variables), (ii) a *guard* $[G]$, where G is a Boolean expression that evaluates to 0 or 1, (iii) a *sequential composition* $a; b$ of terms a and b such that b starts only after a is complete, (iv) a *parallel composition* $a \parallel b$ of terms a and b such that a and b run concurrently, or (v) a *repeat execution* $*[[a]]$ of term a. Other brackets are used in straightforward way to group terms[2]. Besides its CHP term, a circuit has an initial value (0 or 1) for all its variables.

Behavior of CHP programs. We will describe the behavior of such a circuit with the use of a simple example with initial values $A = B = 0$ and CHP term

$$*[[[A = 0]; (A := 1 \parallel B := 1)]]$$

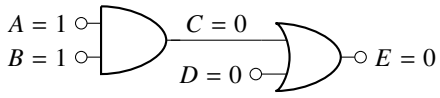
It specifies a circuit that waits until guard $A = 0$ becomes true and then, in parallel, sets the value of variables A and B to 1 (typically with a certain delay). Immediately after that, it repeats and waits until $A = 0$. Since initially $A = 0$, both A and B will indeed be set to 1.

Translation to gates. While the above CHP term might look sufficiently simple to directly derive a gate-level implementation from it, more complex terms may not be that easily derivable. CHP terms are thus iteratively translated into terms of specific structure: a list of production rules. A *production rule* is a term $*[[[G]; x := x']]$, also denoted as $G \rightarrow x := x'$ for simplicity, that can be read as: when G becomes true assign x' to x, where x' is either the constant 0 or the constant 1. All production rules are executed in parallel. The behavior, in accordance with the semantics of CHP, is as follows: whenever a guard is true its assignment is executed (with a certain delay). CHP notation can be distinguished from chemical reaction network notation by a rate that is indicated above the arrow when describing a chemical reaction.

For example, the circuit with initial values $A = B = 1$ and $C = D = E = 0$ with production rules

$$A \wedge B \rightarrow C := 1 \tag{1}$$
$$\neg(A \wedge B) \rightarrow C := 0 \tag{2}$$
$$C \vee D \rightarrow E := 1 \tag{3}$$
$$\neg(C \vee D) \rightarrow E := 0 \tag{4}$$

is now easily seen to be equivalent to the following gate-level implementation (initial values are indicated)



where the first two production rules are the AND gate and the latter two the OR gate. Note that initially rule (1) is executed since $A \wedge B = 1 \wedge 1 = 1$, resulting in an update

[2] In favor of a more concise presentation we present a simplified form of CHP in this chapter. For a full description we refer the reader to Martin's original paper [16].

of C to 1 after some delay. Then rule (3) is executed, resulting in an update of E to 1.

While in circuits for *in silico* implementation it is forbidden to have two production rules whose guards are both true at the same time and that set a variable to different values, such rules are plausible in microbiological settings (see Section 4.2). Depending on the intended semantics such conflicts can be resolved by defining a hierarchy among rules or by resolving conflicts by taking into consideration the time since the guard is activated (or deactivated) and a strength of the rule.

3 Computational Problems

From a formal point of view, computational problems can be distinguished into single-shot problems and those problems that require reactive behavior.

Single-shot problems have classically been the main focus of research in computer science. A problem of this class is stated as a function that maps elements of an input space to elements of an output space according to some specification. Classically, elements of the input and output spaces are finite words of the Boolean alphabet $\{0, 1\}$. For example, the problem to compute the logical OR satisfies the specification $\mathrm{OR}(x_1, \ldots, x_n) = x_1 \vee \cdots \vee x_n$. In computer science, solutions to such problems are classically searched for in terms of Turing machines and Boolean circuits, briefly described below.

Turing machines capture an abstract notion of computing architectures where computation is driven by a single processing unit that manipulates strings, one symbol at a time, starting from the input string and finishing at the desired output string. It therein assumes that functions like reliable non-volatile storage of (arbitrarily long) strings and reliable reading and manipulation of a string at a certain position are provided by the concrete architecture, an assumption that is valid for most *in silico* architectures but not yet for synthetic biological ones.

In *Boolean circuits* computation is driven by Boolean gates, which all operate in parallel, simply transforming binary input signals to output signals and which *a priori* lack any storage elements. These solutions are closer to the microbiological systems that we are discussing here. While searches for solutions in classical computer science are restricted to so called combinational Boolean circuits (cf. Section 3.1.1) that are stateless, we will consider both stateful and stateless Boolean circuits for their practical relevance in real world implementations.

Importantly, for single-shot problems, it is implicitly assumed that a Boolean circuit that represents a history-less solution can be repeatedly evaluated with changing inputs and its result does not depend on that of a former evaluation.

Reactive systems, by contrast, are systems whose behavior can depend on past inputs. A reactive system continuously receives inputs and produces corresponding outputs that may depend on its history. Given a behavior of the environment which provides the controller's inputs and reacts to its outputs, reactive systems' goal is

to provide appropriate outputs such that the combined system, together with the environment, satisfies a given specification.

Analogously to the example of the single-shot OR problem, one may consider the problem of designing a controller that operates in discrete time steps $1, 2, 3, \ldots$: In each time step, it receives an input bit from the environment and has to produce an output that is the OR operation over all input bits received thus far. Such a device could be useful in settings where a controller has to raise an alarm signal once it senses a trigger condition as its input.

3.1 Implementations

Reactive problems inherently require implementations that store an internal state while single-shot problems do not. There may, however, be good reasons for using solutions that use internal state for single-shot problems, e.g., smaller or more efficient implementations: think of the single-shot OR problem with thousands of inputs, or at design time *a priori* unknown number of inputs. One would probably favor a small stateful solution that iteratively solves the problem, remembering intermediate results, over a stateless solution that has to grow with the number of inputs as it cannot store intermediate results from a computation.

3.1.1 Implementations without internal state

A *combinational Boolean circuit* is a circuit that consists only of stateless gates, like AND, OR and NOT, and that has no feedback loops: signals strictly propagate from the input to the output. Thus, if a Boolean input is applied to a combinational circuit, its outputs will settle at binary values solely determined by the input values and not by the order in which they are applied, or previously applied input values. It is thus considered stateless.

For the previously discussed single-shot problem to compute the OR of n bit strings, a natural solution as a combinational Boolean circuit is a tree of OR gates; see Fig. 1.

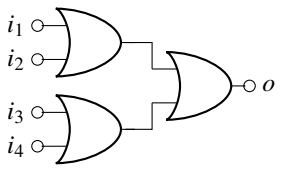


Fig. 1 Combinational circuit that solves the OR problem for 4 input bits i_1, i_2, i_3, i_4.

3.1.2 Implementations with internal state

A deterministic state machine is a set of states S, one of which is the initial state s_0, an input alphabet I, an output alphabet O, a transition function $\delta : S \times I \to S$, and an output function $o : S \to O$. The behavior of this abstract machine in the presence of an input stream $i_1, i_2, i_3, \ldots$ is as follows: starting from the initial state s_0 where it produces output $o(s_0)$, the machine transitions to state $s_1 = \delta(s_0, i_1)$ producing output $o(s_1)$. Next, it transitions to state $s_2 = \delta(s_1, i_2)$ producing output $o(s_2)$, and so on.

Figure 2 depicts a state machine that solves both the single-shot OR problem and the reactive OR problem. Its input and output alphabet is $I = O = \{0, 1\}$, its set of states is $S = \{S_0, S_1\}$ with initial state S_0, its transition function is

$$\delta(S_0, i) = \begin{cases} S_0 & \text{if } i = 0 \\ S_1 & \text{if } i = 1 \end{cases} \quad \text{and} \quad \delta(S_1, i) = \begin{cases} S_1 & \text{if } i = 0 \\ S_1 & \text{if } i = 1 \end{cases}$$

and its output function is $o(S_0) = 0$ and $o(S_1) = 1$.

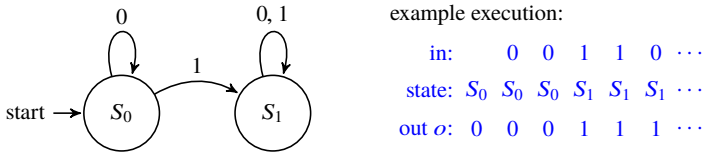


Fig. 2 State machine that solves the OR problem. Transitions are depicted as arrows labeled with inputs. Outputs of the states are $o(S_0) = 0$ and $o(S_1) = 1$. An example execution is shown in blue.

4 Circuits

The combinational circuits and state machines discussed in Section 3.1 are agnostic to specific target technologies or target machines. In the current section we will discuss more concrete implementations targeted for low-level implementation *in silico*, that is, in VLSI circuits, and in synthetic biological circuits.

4.1 Wires

Conceptually, wires or channels provide means to geometrically route information, to provide separation between different signals, and to translate an input species into an output species. Media and transported species vary greatly: while wires (interconnect) in circuits are made out of metal or polysilicon, insulated from each other,

and transport charge (in some cases on-chip fibers transport photons), the situation is far more diverse in microbiological circuits where signals can be metabolites, small molecules, peptides, phages, DNA or RNA. For all such wires orthogonality of signals, error rates, transport delay, etc., are important questions that need to be addressed.

From a computational point of view, a wire is the simplest computation: a gate that computes the identity, typically with a certain delay. Therefore, in terms of production rules, the specification of a wire can be expressed as:

$$I \to O := 1 \quad \text{and} \quad \neg I \to O := 0 \tag{5}$$

When the input is 1 the output becomes 1, and when the input is 0 the output becomes 0.

4.1.1 Electrical Wires

The simplest non-trivial model for an electrical wire is an *RC element*. It consists of a resistor and a capacitor in series. This allows the modeling of not only the non-zero resistance of real wires, but also the various loading effects that any real circuit element invariably displays.

Denoting by R the resistance of the resistor and by C the capacitance of the capacitor, the output voltage O of the RC element reacting to an input voltage I is given by the ordinary differential equation

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \frac{I}{RC} - \frac{O}{RC}$$

For example, when starting from an initial voltage of 0 V (= ground, logical 0), the time behavior of an RC wire reacting to a positive input pulse of voltage V_{DD} (= power supply, logical 1) can be calculated as

$$O(t) = V_{\mathrm{DD}} \cdot \left(1 - e^{-t/\tau}\right)$$

where $\tau = R \cdot C$ is the RC constant. Figure 3 (left) depicts this function.

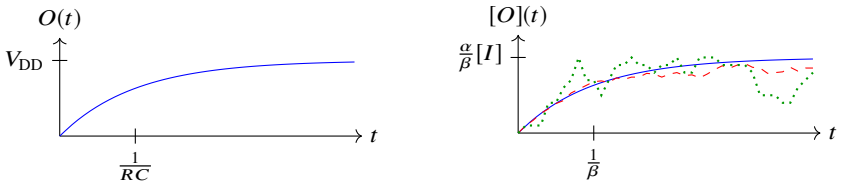


Fig. 3 Behavior of electrical (left) and microbiological (right) wires reacting to a constant input signal. The microbiological signal is shown for three settings: stochastic models with $I = 10$ (dotted green) and $I = 100$ (dashed red), and the ordinary differential equation model (solid blue).

When combining a wire with a Boolean gate, the RC constant of the physical wire is dominated by loading effects inside the gate. In fact, on the scale of gates the RC constant of a physical wire is negligible. In this setting, the derivations from this section are more appropriately applied to an *identity gate*, i.e., a single-input gate whose output is equal to its input.

4.1.2 Microbiological Wires

A very simple implementation of a microbiological wire, or identity gate, can be constructed by having the input molecule I be an activator for the gene encoding output molecule O. Likewise considerations, however, hold for other microbiological wires. After adding a decay reaction, which is necessary for the stability of the model, the simple microbiological wire can be described by the two reactions

$$I + S \xrightarrow{\gamma} I + O \quad \text{and} \quad O \xrightarrow{\beta} \emptyset \tag{6}$$

where β and γ are some rate constants and S is a substrate assumed to be present in abundance (i.e., its concentration remains constant). By applying mass-action kinetics to the system of reactions (6), we get the following ordinary differential equation for the time behavior of the concentration of the output molecule O:

$$\frac{\mathrm{d}[O]}{\mathrm{d}t} = \alpha[I] - \beta[O]$$

where $\alpha = \gamma \cdot [S]$. An immediate consequence of this equation is that the output is in a steady state only if the input is constant and $[O] = \frac{\alpha}{\beta} \cdot [I]$.

The concentration of output molecules O over time, when given a constant concentration of input molecules I, is equal to

$$[O](t) = \frac{\alpha}{\beta} \cdot [I] \cdot \left(1 - e^{-\beta t}\right) \tag{7}$$

when starting at an initial output concentration of zero. Figure 3 (right) depicts this function.

Expressed in the CHP language, the reactions (6) can be interpreted as the following two production rules:

$$I \to O := 1 \quad \text{and} \quad O \to O := 0$$

Note that this CHP formulation is different from (5). In fact, modern electrical implementations directly[3] implement (5) by using complementary stacks (explained in Section 4.2.1), while classical microbiological implementations rely on decay reactions which can lead to a simpler set-up. However, this distinction is not a peculiarity of microbiological settings. In principle, one could also use single-stack

[3] To be precise, an identity gate is usually implemented by two successive inverters.

implementations in electrical circuits and complementary-stack implementations in microbiological circuits.

4.2 Gates

A single output gate is described by two production rules, one that specifies when its output is set to 1, and another that specifies when it is set to 0:

$$P_{\text{up}} \rightarrow O := 1 \quad \text{and} \quad P_{\text{down}} \rightarrow O := 0 \tag{8}$$

Typically P_{up} and P_{down} are required to be mutually exclusive, so that the gate never forces its output to 0 and 1 at the same time. If further P_{up} and P_{down} are negations of each other, that is exactly one of them holds, then the gate is *stateless*. Otherwise, it is *stateful*. An example stateless gate is the OR gate with $P_{\text{up}} = A \vee B$ and $P_{\text{down}} = \neg(A \vee B)$. An example of a stateful gate is the *RS-latch*, a storage element that can be set with $S = 1$ and reset with $R = 1$, with $P_{\text{up}} = S \wedge \neg R$ and $P_{\text{down}} = \neg S \wedge R$. Note that, if $R = S = 0$, then the RS-latch holds its previous output; hence, it is stateful.

4.2.1 CMOS Gate Implementations

Once the circuit has been rewritten in the form of production rules, a standard complementary metal–oxide–semiconductor (CMOS) implementation using transistors is readily obtainable. Consider the two rules in (8) and assume that they represent a stateless gate. For an efficient CMOS implementation, we also require that P_{up} consists only of negated variables and their combination via AND and OR, e.g., $\neg A \wedge \neg B$. Likewise, P_{down} must consist only of positive variables and their combination via AND and OR, e.g., $A \vee B$. Then, rule $P_{\text{up}} \rightarrow O := 1$ is implemented by a *stack of p-type transistors* (one transistor per negated variable) that are responsible to pull the gate's output to V_{DD} (= logical 1) in case P_{up} is true. Rule $P_{\text{down}} \rightarrow O := 0$ is implemented by a *stack of n-type transistors* (one transistor per positive variable) that pull the gate's output towards the ground (= logical 0) if P_{down} is true. Figure 4 shows a CMOS implementation for an inverter, i.e., a gate with $P_{\text{up}} = \neg A$ and $P_{\text{down}} = A$. The p-stack (green) is the above transistor that connects V_{DD} with O if $A = 0$. The n-stack (magenta) is the bottom transistor that connects ground with O if $A = 1$.

Figure 5 shows the two cases for input A: If $A = 0$, the p-stack connects and the n-stack is open. Thus the load at O is charged and pulled up to V_{DD} (= logical 1) via the connecting p-stack. Conversely, if $A = 1$, the n-stack connects and the p-stack is open, thus discharging the load at O through the n-stack to ground (= logical 0).

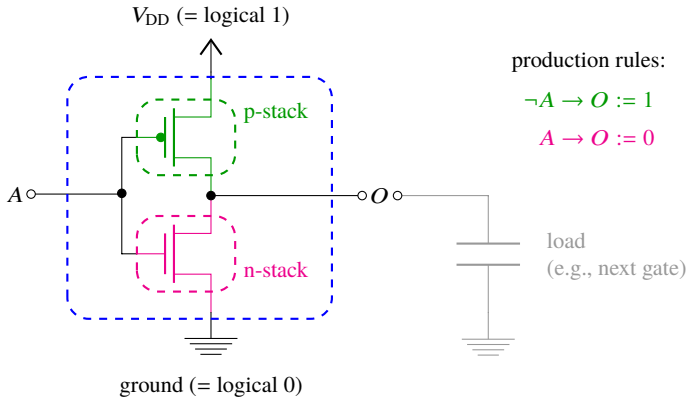


Fig. 4 CMOS implementation of an inverter (left) driving the next gate (right) represented by a capacitive load. Note that the capacitive load is a model for the inertia of the next circuit and is not deliberately built into the circuit.

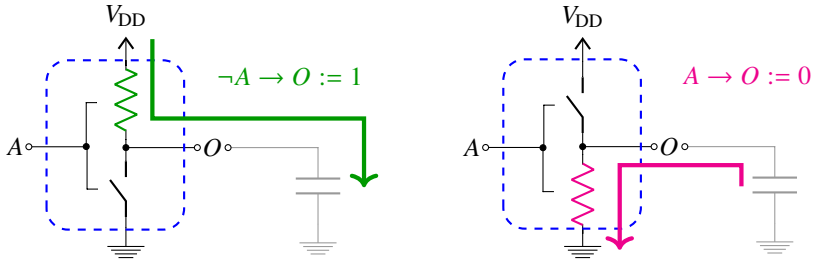


Fig. 5 Case $A = 0$: the load at output O is charged (left). Case $A = 1$: the load at output O is discharged (right).

4.2.2 Microbiological Gate Implementations

Figure 5 shows that the inverter gate does not use charge from input A to charge the output O (so as not to reduce the input charge). Instead, the charge comes from an independent and undepletable source, the power supply V_{DD}.

This concept of decoupling input and output is also observed in naturally occurring reactions and synthetic biological designs. While in the enzymatic reaction

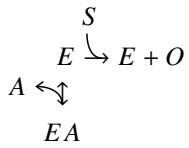
$$A + E \rightleftharpoons AE \to E + O$$

the input molecule A is consumed to create output molecule O via enzyme E, i.e., there is no decoupling, in a reaction like

$$A + E \overset{S}{\rightleftharpoons} AE \hookrightarrow AE + O$$

with a sufficiently available source molecule S, playing the role of V_{DD}, provides decoupling analogous to the CMOS inverter. It resembles the charging via the p-stack in Fig. 5 (left) in that it is enabled if A is present and disabled if A is absent, analogously but negated to its silicon counterpart. In terms of production rules, we may express its behavior as $A \to O := 1$. Typically, the resetting of O is not actively driven, but effected implicitly by the decay of O. Generalizations of this scheme to two input species have, e.g., been used to build AND gates [28, Fig. 1D–F]. Inputs are two complementary parts of the phage T7 RNA polymerase, E is a promoter, and S is the pool of nucleotides to transcribe into O.

By contrast, the scheme



is decoupled and negating, analogous to the p-stack. In terms of production rules, we may write $\neg A \to O := 1$. Again, resetting of O typically is by decay of O. Figure 6 shows the reactions of an inverter with "charging" (in green) and "discharging" (in magenta) reactions corresponding to Fig. 5. Generalizations of this to two input species have been used to implement NOR gates [2, Fig. 2c]. It works by repressing expression of O via the CRISPR/dCas9 machinery. Inputs are guide RNAs, E is a promoter, and S is the pool of nucleotides and dCas9.

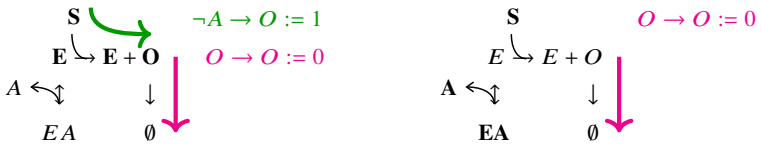


Fig. 6 Case $A = 0$: output molecule O is produced (left). Case $A = 1$: output molecule O is decayed if present (right). Molecules with high concentrations are printed in bold.

4.3 Stateful Circuits

In this section, we turn our attention to non-combinational circuits, i.e., circuits that contain some internal state. We do this by discussing implementations of the RS-latch with CMOS circuits and microbiological circuits.

The capacity to store bits, as is possible with the RS-latch, is prototypical for stateful circuits. In fact, the latch is used as a component in the so-called flip-flop circuits that are widely used to store states in CMOS circuits. For example, the reactive OR problem from Section 3.1.2 can be implemented with an RS-latch.

4.3.1 CMOS Circuits

Most CMOS circuits possess some internal state. The RS-latch in CMOS circuits is most commonly implemented with two NOR gates, as shown in Fig. 7. The

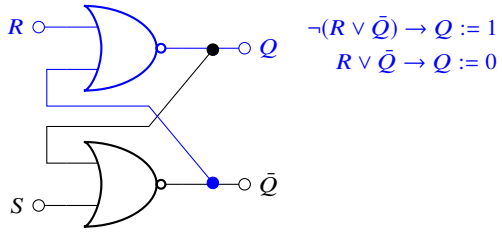


Fig. 7 Implementation of RS-latch with two NOR gates. The gate driving Q is shown in blue and its production rules are stated.

implementation's main idea is to translate the RS-latch representation in the CHP language

$$*[[[S = 1 \wedge R = 0]; Q := 1]] \parallel *[[[S = 0 \wedge R = 1]; Q := 0]]$$

syntactically into the production rules

$$(S \wedge \neg R) \to Q := 1 \quad \text{and} \quad (\neg S \wedge R) \to Q := 0$$

and then to introduce the variable $\bar{Q} = \neg Q$ that allows rewriting the production rules into:

$$\neg(S \vee Q) \to \bar{Q} := 1 \quad \text{and} \quad (S \vee Q) \to \bar{Q} := 0$$
$$\neg(R \vee \bar{Q}) \to Q := 1 \quad \text{and} \quad (R \vee \bar{Q}) \to Q := 0$$

Given this CHP representation, the possible implementation with two NOR gates is clearly visible.

4.3.2 Microbiological Circuits

In synthetic biology, circuits with internal state are less common than in CMOS circuits. While there is a considerable number of microbiological circuits without an internal state [24], one of the first microbiological circuits with an internal state was the bacterial toggle switch [7]. Recently, theoretical microbiological asynchronous circuit designs with internal states have appeared [22]. Figure 8 (left) shows the implementation of the toggle switch as a genetic circuit. Interestingly, due to different implementation constraints and the implementation's single-stacked nature, the implementation of the toggle switch does not decompose into two NOR gates,

as is the case for the CMOS implementation. We provided the corresponding CHP production rules in Fig. 8 (right).

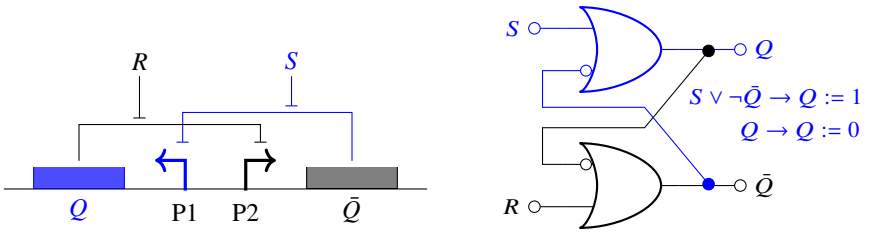


Fig. 8 Toggle switch design from [7]. The components driving Q are shown in blue (left). Corresponding production rules and gate-level description (right). By shifting the input negation to the output observe that signal $\neg\bar{Q}$ in the microbiological circuit corresponds to Q in the circuit in Fig. 7 and $\neg Q$ to $\bar{Q}$. Since $Q = \neg\bar{Q}$ during normal operation (R and S are not issued at the same time), the CMOS and the microbiological circuit behave equivalently.

4.4 Challenges

In previous sub-sections we discussed an idealized view on CMOS and microbiological circuits. In the following, we point to the limitations of such a view: circuit components may fail, both at production and mission time, and apparently simple operations like setting an output to 1 are, in fact, continuous processes with a range of implications.

4.4.1 CMOS Circuit Implementations

The typical production faults that may cause a permanent unintended behavior of the circuit are open and bridge faults at the transistor level [5]. In an *open fault* a connection is interrupted, while in a *bridge fault* two points are connected via an unintended (low) resistance. Possible reasons are friction, electrical wear-out, impurities during fabrication, etc., and the effects are typically permanent once they occur. Open and bridge faults within a gate may have different effects on the gate output that depend on the gate, the resistance of the bridge, etc. For simplicity, typically two types of behaviors are assumed, which the circuit is tested for: in a *stuck-at*-0 or *stuck-at*-1 fault the gate output is simply stuck to the logical value of 0 or 1. An example for a stuck-at-1 fault is a low resistance bridge from O to V_{DD} in Fig. 4. The second common manifestation at gate-level is the *delay-fault* where logical gate behavior is as expected, but with pronounced delays under certain inputs.

By contrast, a transient fault is a temporary misbehavior of the circuit. In this class falls sudden deposition of charge caused by an ionizing particle hit. Effects are

short pulses created within the logic and bit-flips within the memory [6]. Related to transient faults is signal noise, e.g., induced by environmental fluctuations (e.g., temperature) and voltage drops in the power line due to simultaneous switching activity of numerous gates in a circuit. Such noise is an issue in aggressively timed or low-power CMOS circuits.

Unrelated to the above phenomena is metastability that may occur in a circuit in the absence of any external faults. Consider a circuit that has an internal state whose value can be set to at least two different values, say 0 and 1, e.g., an RS-latch. If there are two scenarios, one where the environment can drive the circuit into state 0 and one where it can drive it into state 1, and the scenarios can be continuously transformed into each other, then there also exists a scenario in which the state is in-between where it resides for an arbitrarily long period [15], the *metastable state*. This is problematic as a circuit in this state may produce in-between voltage outputs that are neither logical 0 nor logical 1. These can corrupt downstream gates that expect clear voltage inputs.

4.4.2 Microbiological Circuit Implementations

Several of the aforementioned faults have corresponding faults in synthetic biological implementations. Brophy and Voigt [2] provided a survey of common permanent faults that have been observed in synthetic biology designs using the example of an AND gate and an oscillator. A central source of the faults described were mismatched components (e.g., dynamic range and offset mismatches) that essentially lead to a loss of the digital abstraction. In the AND gate the faults led to dynamic range and offset mismatches at its output. While these range mismatches do not play a prominent role in modern CMOS designs, inaccessible ribosome binding sites (RBS) because of RBS context may be viewed as open faults and lead to stuck-at-0 faults at the gate output. Stuck-at-0 faults were also observed after unintended recombination had taken place in the synthetic circuit. Missing orthogonality between (repressor) signals and corresponding promoters as well as transcriptional read-through because of insufficient insulation by terminators lead to faults that resembled bridge faults in CMOS circuits. Although the gate's behavior was similar to that under bridge fault, we would like to stress an important difference between the biological bridge faults and those *in silico*: their directionality. While bridge faults in CMOS are modeled as bidirectional, high resistance connections, their microbiological counterparts (missing orthogonality and read-through) *a priori* are often directional.

In addition to permanent faults, noise plays a pronounced role in both silicon and biological computers. Although signal noise is an issue in low-power CMOS circuits with reduced dynamic range, the situation is intensified in biological circuits; e.g., see the stochastic behavior for $I = 10$ and $I = 100$ input molecules in Fig. 3 where unintended short-0 pulses are to be expected. While such pulses may lead to unintended, but time-limited, outputs in stateless circuits, their presence at the input of stateful gates like the toggle switch may lead to a permanent failure of the circuit.

5 Conclusions

In this chapter, we presented design techniques for digital circuits, both for silicon and biological computers. We saw that, while many high-level notions, like the abstract concept of Boolean gates and their representation, are identical in both worlds, distinctions have to be made when descending from the abstraction levels towards functional implementations. This distinction is already visible in the Boolean representation domain, as we saw that different implementations require different production rules in the CHP language.

What we have presented in this chapter is only a small part of the vast space of circuit design, both for electrical and biological circuits. But, we have tried to present a relevant selection of similarities and prototypical differences between the two. Clearly, there is a rich interplay possible between the fields of CMOS circuit design and synthetic biology circuit design, though care must be taken to not blindly transfer notions or complete designs from one to the other.

References

1. Anant Agarwal and Jeffrey H. Lang. *Foundations of Analog and Digital Electronic Circuits*. Morgan Kaufmann, San Francisco, 2005.
2. Jennifer A. N. Brophy and Christopher A. Voigt. Principles of genetic circuit design. *Nature Methods*, 11(5):508–520, 2014.
3. Matthew S. Dodd, Dominic Papineau, Tor Grenne, John F. Slack, Martin Rittner, Franco Pirajno, Jonathan O'Neil, and Crispin T. S. Little. Evidence for early life in Earth's oldest hydrothermal vent precipitates. *Nature*, 543:60–64, 2017.
4. Zohar Erez, Ida Steinberger-Levy, Maya Shamir, Shany Doron, Avigail Stokar-Avihail, Yoav Peleg, Sarah Melamed, Azita Leavitt, Alon Savidor, Shira Albeck, et al. Communication between viruses guides lysis–lysogeny decisions. *Nature*, 541:488–493, 2017.
5. F. Joel Ferguson and John P. Shen. A CMOS fault extractor for inductive fault analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(11):1181–1194, 1988.
6. Véronique Ferlet-Cavrois, Lloyd W. Massengill, and Pascale Gouker. Single event transients in digital CMOS: A review. *IEEE Transactions on Nuclear Science*, 60(3):1767–1790, 2013.
7. Timothy S. Gardner, Charles R. Cantor, and James J. Collins. Construction of a genetic toggle switch in Escherichia coli. *Nature*, 403:339–342, 2000.
8. Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
9. C. A. R. Hoare. Communicating sequential processes. In *The Origin of Concurrent Programming*, pages 413–443. Springer, Heidelberg, 1978.
10. Michael Hucka, Andrew Finney, Herbert M. Sauro, Hamid Bolouri, John C. Doyle, Hiroaki Kitano, Adam P. Arkin, Benjamin J. Bornstein, Dennis Bray, Athel Cornish-Bowden, et al. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.
11. Evgeny Katz. Enzyme-based logic gates and networks with output signals analyzed by various methods. *ChemPhysChem*, 18(13):1688–1713, 2017.
12. Ahmad S. Khalil and James J. Collins. Synthetic biology: Applications come of age. *Nature Reviews Genetics*, 11(5):367–379, 2010.
13. Christina Kiel, Eva Yus, and Luis Serrano. Engineering signal transduction pathways. *Cell*, 140(1):33–47, 2010.

14. Peter T. Macklem and Andrew Seely. Towards a definition of life. *Perspectives in Biology and Medicine*, 53(3):330–340, 2010.
15. Leonard R. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, C-30(2):107–115, 1981.
16. Alain J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. Technical Report Caltech-CS-TR-89-1, California Institute of Technology, 1989.
17. James Alastair McLaughlin, Matthew Pocock, Göksel Mısırlı, Curtis Madsen, and Anil Wipat. VisBOL: Web-based tools for synthetic biology design visualization. *ACS Synthetic Biology*, 5(8):874–876, 2016.
18. Jacques Monod. *Chance and Necessity: An Essay on the Nature of Philosophy of Modern Biology*. Collins, London, 1972.
19. Jacques Monod and François Jacob. General conclusions: Teleonomic mechanisms in cellular metabolism, growth, and differentiation. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 26, pages 389–401. Cold Spring Harbor Laboratory Press, 1961.
20. Tae Seok Moon, Chunbo Lou, Alvin Tamsir, Brynne C. Stanton, and Christopher A. Voigt. Genetic programs constructed from layered logic gates in single cells. *Nature*, 491:249–253, 2012.
21. Camilo Mora, Derek P. Tittensor, Sina Adl, Alastair G. B. Simpson, and Boris Worm. How many species are there on Earth and in the ocean? *PLoS Biology*, 9(8):e1001127, 2011.
22. Tramy Nguyen, Timothy S. Jones, Pedro Fontanarrosa, Jeanet V. Mante, Zach Zundel, Douglas Densmore, and Chris J. Myers. Design of asynchronous genetic circuits. *Proceedings of the IEEE*, 107(7):1356–1368, 2019.
23. Daniel J. Nicholson. Is the cell really a machine? *Journal of Theoretical Biology*, 477:108–126, 2019.
24. Alec A. K. Nielsen, Bryan S. Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A. Strychalski, David Ross, Douglas Densmore, and Christopher A. Voigt. Genetic circuit design automation. *Science*, 352(6281):aac7341, 2016.
25. Giulia Oliva, Tobias Sahr, and Carmen Buchrieser. The life cycle of L. pneumophila: Cellular differentiation is linked to virulence and metabolism. *Frontiers in Cellular and Infection Microbiology*, 8:3, 2018.
26. Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.
27. Herbert M. Sauro and Kyung Kim. Synthetic biology: It's an analog world. *Nature*, 497:572, 2013.
28. Yolanda Schaerli, Magüi Gili, and Mark Isalan. A split intein T7 RNA polymerase for transcriptional AND-logic. *Nucleic Acids Research*, 42(19):12322–12328, 2014.
29. John Selberg, Marcella Gomez, and Marco Rolandi. The potential for convergence between synthetic biology and bioelectronics. *Cell Systems*, 7(3):213–244, 2018.
30. Brian Cantwell Smith. The foundations of computing. In Matthias Scheutz, editor, *Computationalism: New Directions*, pages 23–58. MIT Press, Cambridge, 2002.
31. Victor Sourjik and Ned S. Wingreen. Responding to chemical gradients: Bacterial chemotaxis. *Current Opinion in Cell Biology*, 24(2):262–268, 2012.
32. Alexander Spirov, Khalid Fahmy, Martina Schneider, Erich Frei, Markus Noll, and Stefan Baumgartner. Formation of the bicoid morphogen gradient: An mRNA gradient dictates the protein gradient. *Development*, 136(4):605–614, 2009.
33. Jonathan J. Y. Teo, Sung Sik Woo, and Rahul Sarpeshkar. Synthetic biology: A unifying view and review using analog circuits. *IEEE Transactions on Biomedical Circuits and Systems*, 9(4):453–474, 2015.