Naval Research Laboratory
Washington, DC 20375-5320

NRL/5540/MR—2021/1

Logical Analysis of Multiple Clock Domains

DR. GERARD ALLWEIN

Center for High Assurance Computer Systems Branch
Information Technology Division

August 26, 2021

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
26-08-2021	NRL Memorandum Report	1 Oct 2020 – 30 Sept 2021

4. TITLE AND SUBTITLE

Logical Analysis of Multiple Clock Domains

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
062235N

6. AUTHOR(S)

Dr. Gerard Allwein

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
6B23

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/5540/MR--2021/1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
875 N. Randolph Street
Arlington VA 22217-1995

10. SPONSOR / MONITOR'S ACRONYM(S)
ONR

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This is a report on how to manage multiple clock domains from a logical perspective starting from some basic hardware. The hardware is a chain of flip-flops used to synchronize signals from one clock domain to another. The analyses is done in Channel Theory and Distributed Logic. Channel Theory was the precursor to Distributed Logic and both are viable for the analyses. The logical analyses takes the form of a pertinent example showing how to represent the key facts necessary for the logic. At the end, the synchronizing primitives of the languages Esterel, Lustre, and Signal are briefly explained. These languages are early attempts at defining computation by corralling asynchrony to be synchrony. None of the languages has been applied (to the author's knowledge) to FPGA applications. Yet, the synchrony primitives capture the essential features used in the chain of flip-flops that are used in FPGA applications. Many FPGA applications use message queues to enforce synchronous behavior, the analyses here could have used those. However, the flip-flop use is more basic and presents the key features adequately yet has minimal footprint in hardware.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Dr. Gerard Allwein
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	UU	27	**19b. TELEPHONE NUMBER** *(include area code)*
UU	UU	UU			(202) 404-3748

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

This page intentionally left blank.

CONTENTS

EXECUTIVE SUMMARY E-1

1. INTRODUCTION 1

2. MULTIPLE CLOCK DOMAINS 2

3. LOGIC SYSTEMS 3
   - 3.1 Primer on Channel Theory 3
   - 3.2 Classifications and Infomorphisms 4
   - 3.3 Primer on Distributed Logic 7

4. SIMPLE SYNCHRONIZER 10
   - 4.1 Channel Theory Version 10
   - 4.2 Distributed Logic Version 12
   - 4.3 Testing: Distributed Logic Version 13

5. CLOCK DOMAINS AS MODAL EFFECTS 15
   - 5.1 Clock Domains and Modal Operators 16
   - 5.2 An Aura of Localities 18

6. ESTEREL, LUSTRE, AND SIGNAL 19
   - 6.1 Clocks 19
   - 6.2 Esterel 20
   - 6.3 Lustre 20
   - 6.4 Signal 21

REFERENCES 23

This page intentionally left blank.

EXECUTIVE SUMMARY

This is a report on connecting hardware synchronization of clock domains with logical analyses. The hardware synchronization takes the form of a chain of flip-flops from two domains that require synchronization. To synchronize a signal from one clock domain to another means to generate the signal in the first domain and feed it to a circuit that latches the signal. The latch in turn is connected to series of flip-flops (a circuit) from the second domain. Since the clock domains are different, each circuit is clocked at a different rate. There are two problems this set up addresses: (1) the clock on the first domain runs faster than the clock on the second, and the clock on the second runs faster than the clock on the first. The second problem could be handled by polling the signal from the first domain from the second. The first problem requires the signal gets latched until such time as the second can catch the rising edge of the signal. The chain of flip-flops pushes the probabilistic behavior to an acceptable range due to possible metastable signals in the flip-flop chain. Many FPGA applications use message queues to enforce synchronous behavior, the analyses here could have used those. However, the flip-flop use is more basic and presents the key features adequately yet has minimal footprint in hardware.

The logic analyses is via two logical theories, Channel Theory and Distributed Logic. The former is a precursor to the latter. It is easier to set up the Channel Theory since every piece of the chain of flip-flops is represented. The latter is a bit more abstract but easier to implement in hardware monitors (called *spiders*). The setup takes the form of using a flip-flop for a latch connected to a chain of flip-flops. The relevant information is then extracted and represented in the logics. One thing to notice in the logic presentation is that there is no concept of time. The synchronizer use has the effect of removing time as measure in the sense of the real numbers. Rather, only clock ticks are used and there is no time associated with those ticks. A clock domain has all its components performing in unison at each clock tick. The only notion of time remaining is one tick occurring before another. This has the effect of allowing clock ticks to be pushed into the background in favor of state changes since those state changes can only occur on clock ticks. Similarly, the logic need only abstract over before and after, i.e., previous clock tick, current clock tick, and next clock tick. The fact that more than one clock domain can appear in an FPGA application means that once signals are adequately synchronized between the clock domains, logical formulas associated with each domain can be kept separated and not collide in the logical reasoning.

Carrying the logic analyses further, we associate a modality with each clock domain. In an application language that supports strong typing, this could be reified as an *effect type*. An effect type is more like a type modifier than an actual type. Using effect types, all signals can have the effect type as well as any other typing information derivable from the application code. The languages VHDL and Verilog do not support strong typing and never will. However, the language ReWire could be extended with effect types and thus enforce adherence to clock domain rules at compilation time.

At the end, the synchronizing primitives of the languages Esterel, Lustre, and Signal are briefly explained. These languages are early attempts at defining computation by corralling asynchrony to be synchrony. None of the languages has been applied (to the author's knowledge) to FPGA applications. Yet, the synchrony primitives capture the essential features used in the chain of flip-flops that are used in FPGA applications.

This page intentionally left blank.

LOGICAL ANALYSIS OF MULTIPLE CLOCK DOMAINS

1. INTRODUCTION

We show how to manage multiple clock domains from a logical perspective starting from some basic hardware. The hardware is a chain of flip-flops used to synchronize signals from one clock domain to another. Asynchronous queues are used in many applications. We do not address them here because they would not affect the analyses. The analyses is done in Channel Theory [1–3] and Distributed Logic [4, 5]. Channel Theory was the precursor to Distributed Logic and both are viable for the analyses.

We first define the overall characteristics of a clock domain. The key feature is transfer of signals between clock domains. Since the clocks of two domains are not synchronized, they require some sort of synchronizing hardware component that will accomplish the transfer. Many applications use message queues to transfer information between domains. These are high-level components and are somewhat overkill for many basic FPGA applications. As such, we will describe the simplest of flip-flop synchronizers. There are many forms of flip-flop synchronizers that have better properties than what we describe. However, they are variations of this basic synchronizer. The hardware queues can be seen as fancy flip-flop synchronizers.

Assume we want to transfer signals on Clock Domain 1 (CD1) to Clock Domain 2 (CD2). There are two basic problems to be overcome: (1) signals generated in CD1 faster than they can be recognized in CD2, and (2) signals generated in CD1 slower than they are being capable of being recognized in CD2. In the first instance, the generated signals must be latched in CD2 until such time as CD2 can recognize them. In the second instance, a mere polling will suffice. The polling still requires that signals be guarded against becoming metastable due to the different clock rates..The chain of flip-flops will solve both problems to within a probability of failure. The flip-flop chain can be increased in size to lower the failure probability.

The logic analyses is via two logical theories, Channel Theory and Distributed Logic. The former is a precursor to the latter. It is easier to set up the Channel Theory since every piece of the chain of flip-flops is represented. The latter is a bit more abstract but easier to implement in hardware monitors (called *spiders*). The setup takes the form of using a flip-flop for a latch connected to a chain of flip-flops. The relevant information is then extracted and represented in the logics. One thing to notice in the logic presentation is that there is no concept of time. The synchronizer use has the effect of removing time as measure in the sense of the real numbers. Rather, only clock ticks are used and there is no time associated with those ticks.

A clock domain has all its components performing in unison at each clock tick. The only notion of time remaining is one tick occurring before another. This has the effect of allowing clock ticks to be pushed into the background in favor of state changes since those state changes can only occur on clock ticks. Similarly, the logic need only abstract over before and after, i.e., previous clock tick, current clock tick, and next clock tick. The fact that more than one clock domain can appear in an FPGA application means that once signals are adequately synchronized between the clock domains, logical formulas associated with each domain can be kept separated and not collide in the logical reasoning.

Manuscript approved August 24, 2021.

Carrying the logic analyses further, we associate a modality with each clock domain. In an application language that supports strong typing, this could be reified as an *effect type*. An effect type is more like a type modifier than an actual type. Using effect types, all signals within a domain can have the effect type as well as any other typing information derivable from the application code. The languages VHDL and Verilog do not support strong typing and never will. However, the language ReWire [6, 7] could be extended with effect types and thus enforce adherence to clock domain rules at compilation time.

At the end are some short analyses on the languages Esterel, Lustre, and Signal. These are early generations of synchronous languages relying upon message passing. They were not developed for FPGA applications and have not been applied to FPGA applications except (possibly) sparsely in academic settings. None of these languages has the notion of a clock domain as a feature but rather use a partial order of clock frequencies. As such, they never really define properties of a clock domain. The domains could possibly be constructed from the partial order and expressions in a language but that is fraught with error to do manually and we are not aware of any automatic tools. Seeing as these languages were developed without FPGA applications in mind, this is not so much a defect as it simply was not in their field of view for applications of these languages. Generally, the languages are academic efforts at coming to grips with synchronized message transfer and they have succeeded at that.

2. MULTIPLE CLOCK DOMAINS

The following diagram shows the basic layout of a flip-flop synchronizer from [8]. The reset line for the flip-flop of Bit 0 must not be activated until sufficient time has passed for CD2 to activate Bit 1 flip-flop. This allows the signal output at Q of Bit 0 to be recognized at D of the Bit 1 flip-flop. From now on we will use the locution Bit 0.D, say, to reference the *D* input of flip-flop Bit 0 and similarly for the other signals.

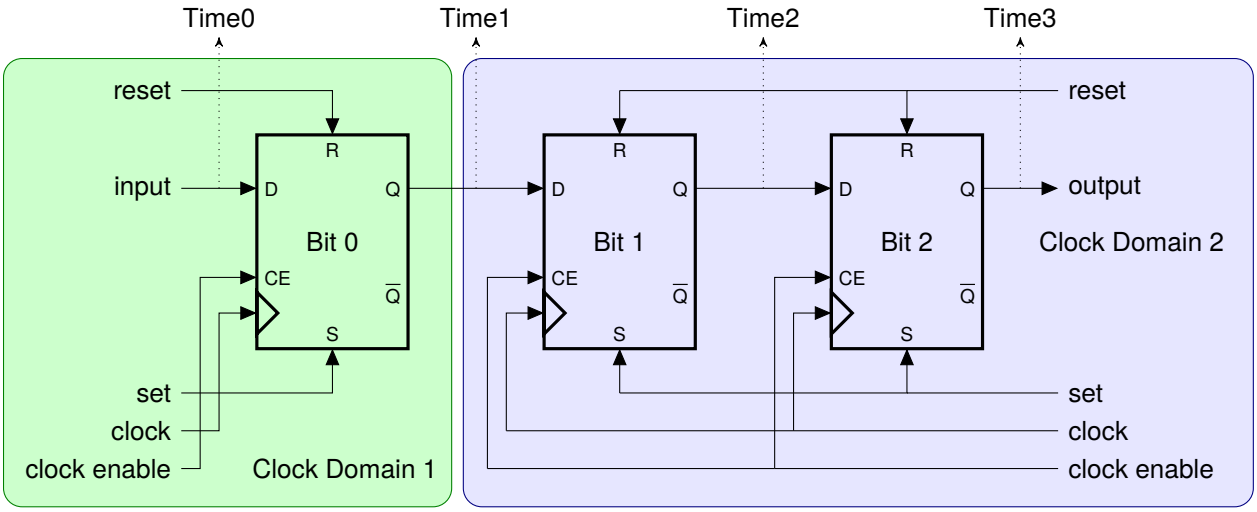


Figure 2.1: Simple Synchronizer

The sampling of sampling Bit 0.D by flip-flop Bit 1 of Clock Domain 2 may cause the output Bit 1.Q to go metastable. The rate of the clock in CD2 should allow the output to settle down in time for Bit 2.D to

accurately record the signal from Bit 1.Q. The probability for Bit 2.Q becoming metastable can be calculated. If the probability is greater than the circuit requirements, flip-flops can be added after Bit 2 to lower the probability to an acceptable range.

One thing to notice is that if we are just concerned with representing signals and not clock rates, then all we need consider is that signals from CD1 get transferred to CD2. While any signal from CD1 can be transferred to CD2, most usually are not. The result is that CD1 has only a small (and very finite) number of synchronizers connecting it to CD2. At this point, we need only be concerned with what signals get transferred and what signals cannot be transferred. Now we have abstracted our clock domains into regions of like-clocked signals and a few transfer portals (synchronizers).

With the previous abstraction in mind, we can look at the situation from a language perspective. This perspective will support either FPGA languages or logical languages. For VHDL and Verilog, since they cannot support effect types, the use of effect types in the analyses is merely to label parts of the design. In this report, we are concentrating on logical languages and not these application languages.

We let the clock domains be represented by application language effect types and match them with logical modalities for logical languages. Let the language types be Time0, Time1, and Time2. The type Time1, can generally be suppressed in CD2 since it is only required in one place, whereas Time0 and Time2 are representative of all the other signals in CD1 and CD2 respectively.

Chaining together two flip-flops requires we have two equations on types,

$$\mathsf{Time1} = \mathsf{Time0} + 1 \text{ and } \mathsf{Time3} = \mathsf{Time2} + 1.$$

and we are allowed to make the type inference

$$\frac{\mathsf{Time1} = \mathsf{Time0} + 1 \text{ and } \mathsf{Time2} = \mathsf{Time1} + 1}{\mathsf{Time2} = \mathsf{Time0} + 2}$$

3. LOGIC SYSTEMS

We use two logic systems, Channel Theory and Distributed Logic. The former is a bit easier to match up with the chain of flip-flops used for synchronization. The latter is more abstract but easier to manage from logical perspective.

3.1 Primer on Channel Theory

In [9] it is pointed out that the notion of communication channel capacity fails to capture salient features of covert and steganographic channels. In image steganography, information is hidden in a cover image. The Shannon analysis of this situation can put measures on the amount of hidden information the communication channel will support. The problem is that the amounts calculated may have little to do with the transfer of actual information because the information has a qualitative nature to it that is not amenable to the baseline Shannon analysis. A more sophisticated framework is required upon which to base the Shannon analysis. Channel Theory is an answer to the qualitative aspect of information. It is logic based.

3.2 Classifications and Infomorphisms

The basic unit of information in Channel Theory is a tuple of a binary relation. The relationship is between a token (a piece of data, say, as in Shannon theory) and a type (what kind of thing is this data of). This is represented as

$$x \models P$$

where x is the piece of data, $\models$ is the relation, and P is the type. The symbol, $\models$, is the usual semantic symbol of logic and is usually interpreted in logic as "x satisfies P". This paper will treat $\models$ as the relation "x is of type P". There is to be no metaphysical or epistomological baggage to be associated with "x is of type P" even though we sometimes use the verb "satisfy" when talking about $\models$. Also, one cannot express any property about a single token unless the property is reified as a type and the expression is via the $\models$ relation. Hence, for a number x as a token, one can only express its value V by an expression of the form $x \models V$. In this sense, Channel Theory enforces a discipline that is sometimes lacking in analysis of information.

To help orient a reader versed in Shannon's theory, we offer here this description. The basic unit of information in Shannon's theory is also a tuple of a binary relation. The relation is restricted to be of the form $x \models V$ where $\models$ is a function and V is value of the token x. The resulting structure is typically called a state space where V is a state and the tokens are forgotten. Channel theory also has state spaces except the tokens are not forgotten and types are values. States are sometimes further collected together to form events. Channel Theory allows this also by first keeping the tokens and then replacing the states as types with events as types. For some event E, $x \models E$ just when $x \models s$ for some state $s \in E$. Hence, Shannon's basic ontology is neatly embedded in Channel Theory's ontology with Channel Theory being somewhat more rigorous about the specification of the entities involved.

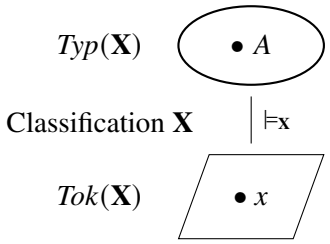
A collection of types and tokens with their relation is known as a *classification*. A more telling term might be *universe of discourse* and one can freely interchange the two terms. A classification is just what you thought it was, it is a collection of things which have the form of "x is a P", or in our parlance, "x is of type P", i.e., $x \models P$. Information can flow between two classifications via an *infomorphism* which is a special pair of contravariant maps between classifications, one for tokens and one for types. When the information flow between two classifications is of such complexity that it cannot be adequately expressed using a single infomorphism, the flow can be re-expressed as a *channel*. A channel is another classification which is connected to the original two classifications via infomorphisms.

3.2.1 Classifications

The basic structures of Channel Theory are deceptively simple. The things that are distributed in a distributive system are contexts called *classifications*. The classifications are connected by *infomorphisms*.

Definition 3.2.1.1 (Barwise–Seligman) A *classification*, $\mathbf{X}$, is a pair of sets and a relation. The sets are called, respectively, the *tokens*, $Tok(\mathbf{X})$, and the *types*, $Typ(\mathbf{X})$. The binary relation, usually symbolized by $\models_{\mathbf{X}}$, is between the two sets, i.e., $\models_{\mathbf{X}} \subseteq Tok(\mathbf{X}) \times Typ(\mathbf{X})$. The term $x \models_{\mathbf{X}} A$ means $\langle x, A\rangle \in \models_{\mathbf{X}}$ with $x \in Tok(\mathbf{X})$ and $A \in Typ(\mathbf{X})$.

A good mental picture to remember the definition of a classification is the following:



The diagram only indicates that $A \in Typ(\mathbf{X})$, and $x \in Tok(\mathbf{X})$, not that $x \models_{\mathbf{X}} A$

It is convenient to talk about all of the tokens satisfying a single type or all of the types satisfying a particular token. The following definition relativizes $Typ(-)$ and $Tok(-)$ within a particular classification.

Definition 3.2.1.2 Let $\mathbf{X} = (Tok(\mathbf{X}), Typ(\mathbf{X}), \models_{\mathbf{X}})$ be a classification, then for any $A \in Typ(\mathbf{X})$, $Tok(A) = \{y \mid y \models_{\mathbf{X}} A\}$ and, for any $x \in Tok(\mathbf{X})$, $Typ(x) = \{B \mid x \models_{\mathbf{X}} B\}$.

3.2.2 Infomorphisms

The "flow" of information flow is rarely qualified in many theories of information flow although it is frequently quantified as data flow. Since the currency of information is the tuple "x is of type P", to translate information (where here we are using "translate" in its sense as a preservation mapping), one first thinks to translate the x to a y and the P to a Q. This turns out not to be in accord with most uses of classifications within mathematics and logic. More to the point, the morphisms of classifications must relate tokens and types of two classifications in a special way, not simply translate token-type tuples to token-type tuples. The reason for this is that the way set mappings work is similar to the way logical morphism work. In particular

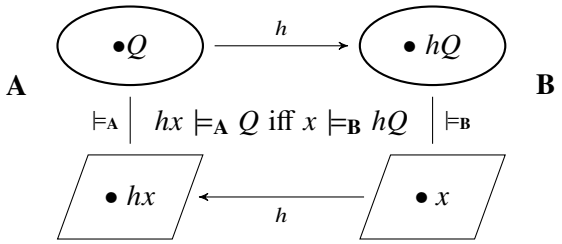
$$x \in f^{-1}C \text{ iff } fx \in C.$$

where we elide the parens wherever possible.

Definition 3.2.2.1 An Infomorphism requires a source and targeta that are classifications. Assume classifications $\mathbf{A} = (Tok(\mathbf{A})\ Typ(\mathbf{A}), \models_{\mathbf{A}})$ and $\mathbf{B} = (Tok(\mathbf{B}), Typ(\mathbf{B}), \models_{\mathbf{B}})$. An *infomorphism* $h : \mathbf{A} \rightarrow \mathbf{B}$ is a pair of contravariant maps, $\overrightarrow{h}$ and $\overleftarrow{h}$ such that $\overrightarrow{h} : Typ(\mathbf{A}) \rightarrow Typ(\mathbf{B})$ and $\overleftarrow{h} : Tok(\mathbf{B}) \rightarrow Tok(\mathbf{A})$, and for all p and Q, the following condition is satisfied:

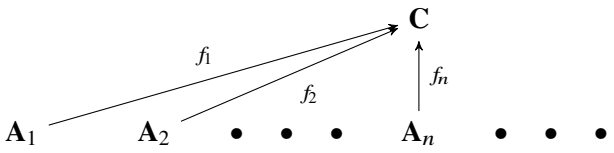
$$hx \models_{\mathbf{A}} Q \text{ iff } x \models_{\mathbf{B}} hQ,$$

where for ease of presentation, $\overleftarrow{h}(x)$ is displayed as hx and $\overrightarrow{h}(Q)$ as hQ where the type of the argument disambiguates the context. We declare by fiat the direction of the infomorphism to be the direction of its map on types.



3.2.3 Channels

Definition 3.2.3.1 An **information channel** is consists of an indexed family $C = \{f_i : \mathbf{A_i} \to \mathbf{C}\}$ of infomorphisms with a common codomain **C** called the **core** of the channel. Diagrammatically,



where each f_i is an infomorphism. Frequently in the sequel, the term *channel* will be (mis)used to refer to the core of the channel. This is for mere expediency and the reader is asked to be forgiving. There is never any question as to which morphisms are involved.

We will only have need for binary channels (binary refers to two legs, not binary circuits), pictured thusly:

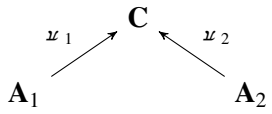


Figure 1: Binrary Channel

where $\imath_i$ refers to an injection on types and and a projection π_i on tokens.

Every channel has a *theory* which is a collection sequents of the form:

$$\Gamma \Vdash_{\mathbf{C}} \Delta$$

where Γ and Δ are sets of logical formulas. Γ is thought of conjunctively and Δ disjunctively. The tokens in the channel **C** constitute a binary relation. We will only need a simplified notion of sequent where the right and left hand sides are single formulas:

$$\imath_1 D_1 \Vdash_{\mathbf{C}} \imath_2 D_2.$$

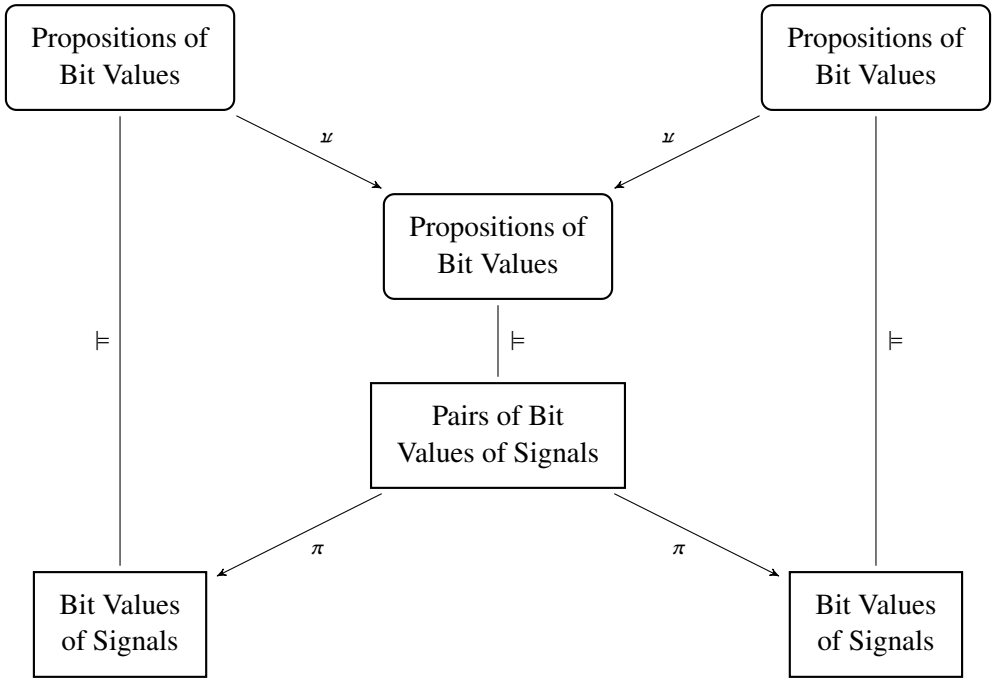
The sense of this is that any tokens that make $\mathscr{u}_1 D_1$ true must make $\mathscr{u}_2 D_2$ true. The general reasoning pattern is

$$\begin{array}{lll} \pi_1 m \models_{\mathbf{A}_1} c_1 \text{ iff} & m \models_{\mathbf{C}} \mathscr{u}_1 c_1 & \text{infomorphism condition} \\ \text{implies } m \models_{\mathbf{C}} \mathscr{u}_2 c_2 & & ? \\ \text{iff} & \pi_2 m \models_{\mathbf{A}_2} c_2 & \text{infomorphism condition} \end{array}$$

The question mark must be mediated by a sequent of the form

$$\mathscr{u}_1 c_1 \Vdash_{\mathbf{C}} \mathscr{u}_2 c_2.$$

Our channels will have the form



3.3 Primer on Distributed Logic

The basic logic is detailed in [5]. We use the term Distributed Logic in a general sense such as modal logic; each has several logics that can fall under the term. Distributed Logic also includes modal logic as simple case. A particular distributed logic is actually a collection of *local* modal logics that are connected in a formal way via distributed operators. The local modal logic has a classical base that admits the usual necessity and possibility operators that abstract over next-state relations.

Distributed Logic lends itself well to FPGA applications. Each local logic is seen as being the local logic of a single component. The components are connected via their behavior; that behavior is expressed using distributed relations. The distributed operators abstract over those relations. The abstraction takes the form of the evaluation conditions on the operator as shown in the sequel. The use of the term *distributed relation*

reflects that a relation is relating two distinct collections of states in different components. This is in contrast to the local *endo-relations* (such as next-state relations); these latter are limited to a single locality

We use the term *locality* to denote a local logic and its underlying component structure as expressed in its states and relations. Thus, the distributed relations connect localities. A common distributed relation is a parallelism relation, called *concurrence*, that represents when states in two different localities can simultaneously occur. Another distributed relation is one that relates all components that share the same clock domain.

We will not go into the logic of Distributed Logic but instead concentrate on interpretations. The logical apparatus can be slightly overwhelming. The frames used in interpreting logic formulas are a bit simpler and is all we will need to evaluate the synchronizers.

3.3.1 The Interpretations

The technical term *frame* in logic can represent many different situations. We use it primarily to represent a component in an FPGA application. Each frame consists of a collection of states, a Boolean algebra of sets (where the sets are sets of states), the $\in$ relation between states and elements of the Boolean algebra of sets, and at least one *local relation*. In other words, it is a particular type of classification where the $\models$ relation is the set theoretic membership relation $\in$. The local relation can represent the next state relation when viewing the component as a finite automaton. However, local relations (not the $\in$ relation, the $\in$ relation is not a local relation) can also represent other notions as the need arises. An example of a non-next-state relation is where some states are considered security critical and related to states that are not security critical and required to not contain information on any of the security critical states.

We assume a graph of localities usually denoted with sans serif, i.e., nodes are denoted h, k, etc. At each node is a classification consisting of a local logic, a frame, a satisfaction relations $\models^{\mathsf{h}}$, $\models^{\mathsf{k}}$, and at least one local relation. The local relation can be the identity relation if no feature of a component needs to be represented using a local relation.

Definition 3.3.1.1 A *local frame* is a structure $H = (H, \mathcal{H}, \mathbb{H})$ such that H is a collection of states. $\mathcal{H} : \mathsf{h} \rightharpoonup \mathsf{h}$ is a local relation connecting some states of H. We use the same symbol for the frame and its collection of states, and let use disambiguate meaning. $\mathbb{H}$ is a collection of *neighborhoods* or sets of states that are subsets of H and the entire collection is closed under the Boolean operations and under the operations $[h^{\square}\rangle, [h^{\circ}\rangle : \mathbb{H} \to \mathbb{H}$. Hence $\mathbb{H}$ is a modal set algebra. These operators are used to define special collections of states. They have valuation conditions in the sequel given by distributed operators, just set the distribution to a single component. The $\in$ relation between states and elements of the Boolean algebra is left implicit.

Definition 3.3.1.2 A *distributed frame* consists of a graph of nodes where each node is a classification, distributed relations linking the states (tokens) and distributed modal operators linking the set algebras (types).

Propositions in the logic are modeled by elements of the set algebras. A distributed frame with two localities can be pictured as

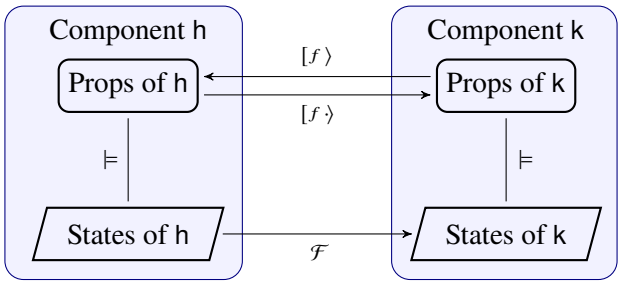


Diagram 3.1: Generic Distribution with Two Components as Localities

where the arrows $[f\rangle, [f\cdot\rangle$ can be $[f^{\square}\rangle, [f^{\square}\cdot\rangle$ and $[f^{\circ}\rangle, [f^{\circ}\cdot\rangle$ interpreted by the distributed relation $\mathcal{F}$, i.e., the lower case f in the modalities is linked with the script $\mathcal{F}$ relation. $[f^{\square}\rangle$ and $[f^{\square}\cdot\rangle$ are necessity operators, and $[f^{\circ}\rangle$ and $[f^{\circ}\cdot\rangle$ are possibility operators.

The relation $\mathcal{F} : \mathsf{h} \rightharpoonup \mathsf{k}$ used in the evaluation of the operators above uses two localities, h and k. These are variables in that any actual FPGA application will fill those localities in by components using as many as are needed.

The distributed relation $\mathcal{F}$ (see diagram) is denoted with an arrow but this is mere convention; $\mathcal{F}$ is a two-place relation that, by fiat, is a morphism from elements of its first position to elements of its second position. The distributed modalities, on the other hand, really are functions although they have special properties required for us to treat them as modalities.

- $\models^{\mathsf{h}} P$ if and only if for all $x \in_{\mathsf{h}} H$, it is the case that $x \models^{\mathsf{h}} P$. Equivalently, $\models^{\mathsf{h}} P$ if and only if for all $x \in_{\mathsf{h}} \top^{\mathbb{H}}$ (where $\top^{\mathbb{H}}$ is the top of the Boolean set algebra $\mathbb{H}$), it is the case that $x \models^{\mathsf{h}} P$.
- $\not\models^{\mathsf{h}} P$ if and only if there is some $x \in_{\mathsf{h}} H$ and $x \not\models^{\mathsf{h}} P$.
- A local logic at h is consistent just when for all $x \in_{\mathsf{h}} H$, it is the case that for all propositions P, not both $x \models^{\mathsf{h}} P$ and $x \not\models^{\mathsf{h}} P$.

Modal formulas $[f^{\square}\rangle$ and $[f^{\circ}\rangle$ for necessity and possibility are evaluated in the usual way except we must respect the distributed nature now of these modalities:

$$x \models^{\mathsf{h}} [f^{\square}\rangle Q \text{ iff } \text{ for all } y,\ \mathcal{F}xy \text{ implies } y \models^{\mathsf{k}} Q \qquad x \models^{\mathsf{h}} [f^{\circ}\rangle Q \text{ iff } \text{ there exists } y,\ \mathcal{F}xy \text{ and } y \models^{\mathsf{k}} Q.$$

The other versions $[f^{\square}\cdot\rangle$ and $[f^{\circ}\cdot\rangle$ run in the other direction:

$$y \models^{\mathsf{k}} [f^{\square}\cdot\rangle P \text{ iff } \text{ for all } x,\ \mathcal{F}xy \text{ implies } x \models^{\mathsf{h}} P \qquad y \models^{\mathsf{k}} [f^{\circ}\cdot\rangle Q \text{ iff } \text{ there exists } x,\ \mathcal{F}xy \text{ and } x \models^{\mathsf{h}} Q.$$

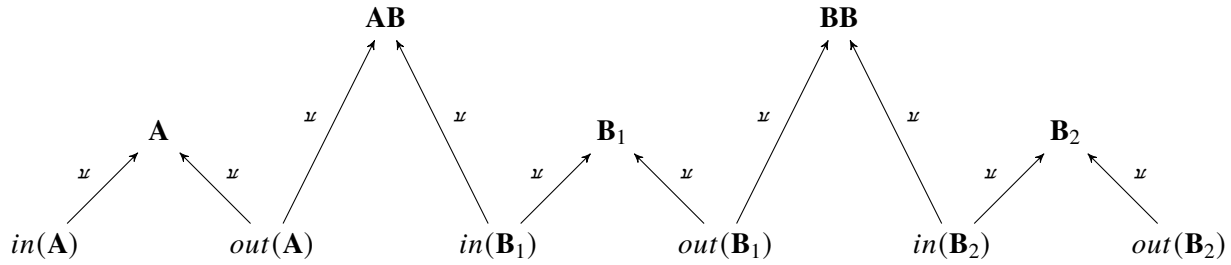
Hence the distributed modalities are evaluated similar to the local modalities except notice the changes between $\models^{\mathsf{h}}$ and $\models^{\mathsf{k}}$ on the two sides of the evaluations.

4. SIMPLE SYNCHRONIZER

We will first analyze the simple synchronizer of Figure 2.1 using Channel Theory and then using Distributed Logic. The Channel Theory version is a bit easier to understand and shows the distributed nature of the synchronizer. This version could be made to support probabilities directly. The Distributed Logic version is a bit more abstract since the intervening channel core is not used. It too could be made to support probabilities but in a more abstract way.

4.1 Channel Theory Version

In [8] there are several techniques for synchronizing signals between two different clock domains. The simple synchronizer in Figure 2.1 is the basis for many of them. There is typically more circuitry devoted to making sure the signals are not corrupted, but that circuity is not going to change the underlying understanding of synchronization. We will use the following Channel Theory diagram



where the $\mathcal{U}$ morphisms represent a pair $(\mathcal{U}, \pi)$ where the first is an injection tagging elements of its domain and π projects elements of pair out with the first element if the $\mathcal{U}$ is on the left and the second element of the $\mathcal{U}$ is on the right.

The top row of channels represents the connections between the flip-flops. The middle row of channels represents the flip-flops themselves, and the bottom row represents input and outputs. The clock strikes only for the middle row of channels. Every clock strike causes the input to move to the output. This is represented by the theories in those channels. The top row of channels is merely representing the connections between flip-flops. The **A** flip-flop is in one clock domain, the $\mathbf{B}_i$ flip-flops are in a second clock domain. As can be seen in the diagram, each flip-flop is represented by a channel. The operation of each flip-flop is simple and a channel might not be needed except for the fact that the output of the $\mathbf{B}_1$ flip-flop can go metastable. It is also convenient to capture the rising edges of the clocks in terms of channels. If the falling edges needed to be represented, two flavors of channels would be required, one for rising edges and one for falling edges.

In the table below, the propositions $d = x$ refer to the input of a flip-flop with the label D in the previous flip-flop diagrams. Similarly, the propositions $q = x$ refer to the output of a flip-flop with the label Q. We have the following sets where $1/2$ stands for the metastable "value" and $\mathcal{U}_i$ applies a tag to all the elements of the set to which it is applied. An index of 1 in $\mathcal{U}_1$ represents a $\mathcal{U}$ on the left of the channel and an index of 2 represents a $\mathcal{U}$ on the right of the channel:

Classification	Tokens	Types
$in(\mathbf{A})$	$0, 1$	$d = 0, d = 1$
$out(\mathbf{A})$	$0, 1$	$q = 0, q = 1$
$in(\mathbf{B}_1)$	$0, 1$	$d = 0, d = 1/2, d = 1$
$out(\mathbf{B}_1)$	$0, 1/2, 1$	$q = 0, q = 1/2, q = 1$
$in(\mathbf{B}_2)$	$0, 1$	$d = 0, d = 1$
$out(\mathbf{B}_2)$	$0, 1$	$q = 0, q = 1$
$\mathbf{A}$	$\langle 0, 0\rangle, \langle 1, 1\rangle$	$\mathcal{U}_1(Typ(in(\mathbf{A}))) \cup \mathcal{U}_2(Typ(out(\mathbf{A})))$
$\mathbf{B}_1$	$\langle 0, 0\rangle, \langle 1/2, 1/2\rangle, \langle 1, 1\rangle$	$\mathcal{U}_1(Typ(in(\mathbf{B}_1))) \cup \mathcal{U}_2(Typ(out(\mathbf{B}_1)))$
$\mathbf{B}_2$	$\langle 0, 0\rangle, \langle 1, 1\rangle$	$\mathcal{U}_1(Typ(in(\mathbf{B}_2))) \cup \mathcal{U}_2(Typ(out(\mathbf{B}_2)))$
$\mathbf{AB}$	$\langle 0, 0\rangle, \langle 1, 1/2\rangle, \langle 1, 1\rangle$	$\mathcal{U}_1(Typ(out(\mathbf{A}))) \cup \mathcal{U}_2(Typ(in(\mathbf{B}_1)))$
$\mathbf{BB}$	$\langle 0, 0\rangle, \langle 1/2, 0\rangle, \langle 1/2, 1\rangle, \langle 1, 1\rangle$	$\mathcal{U}_1(Typ(out(\mathbf{B}_1))) \cup \mathcal{U}_2(Typ(in(\mathbf{B}_2)))$

We note that the outputs of $\mathbf{A}$ are the inputs to $\mathbf{B}_1$. Similarly, the outputs of $\mathbf{B}_1$ are (except for one instance) the inputs of $\mathbf{B}_2$. There is a convention that the output of a flip-flop is only read by the input of the next flip-flop when the clock strikes. The assumption is that the metastable state of the output of $\mathbf{B}_1$ settles down to a logical 0 or 1 before the input to $\mathbf{B}_2$ is read.

It is possible to view **AB** and **BB** either as a continuum of channels that span the time between timing cycles in the two domains or as single channels that have a timer property associated with them. This is reflected in the tables by $out(\mathbf{B}_1)$ containing the token $1/2$ yet $in(\mathbf{B}_2)$ does not contain the $1/2$ token as it never appears in the second position of any of the tokens in the **BB** channel.

This is still not a complete representation of the situation as there is only a probability that the metastable state will settle down to a 1 within a clock slice of the second domain. One can add more flip-flops in the second domain to lower the probability that the metastable state will persist past the synchronizing circuit. This could be represented by putting probabilities on the theory in **BB**. However, doing this means carrying those probabilities throughout the rest of the synchronizing circuit. This would give us a Shannon-style communications channel built from the flip-flops of the second domain. We do not do this as our analyses is qualitative. Further down below, we restate the situation using Distributed Logic. That seems like a perfect driver for a paper on Probabilistic Distributed Logic.

The theories in the channels are

Classification	Theory
A	$\mathscr{u}_1(0) \Vdash_{\mathbf{A}} \mathscr{u}_2(0)$
	$\mathscr{u}_1(1) \Vdash_{\mathbf{A}} \mathscr{u}_2(1)$
$\mathbf{B}_1$	$\mathscr{u}_1(0) \Vdash_{\mathbf{B}_1} \mathscr{u}_2(0)$
	$\mathscr{u}_1(1/2) \Vdash_{\mathbf{B}_1} \mathscr{u}_2(0), \mathscr{u}_2(1/2), \mathscr{u}_2(1)$
	$\mathscr{u}_1(1) \Vdash_{\mathbf{B}_1} \mathscr{u}_2(1)$
$\mathbf{B}_2$	$\mathscr{u}_1(0) \Vdash_{\mathbf{B}_2} \mathscr{u}_2(0)$
	$\mathscr{u}_1(1) \Vdash_{\mathbf{B}_2} \mathscr{u}_2(1)$
AB	$\mathscr{u}_1(0) \Vdash_{\mathbf{AB}} \mathscr{u}_2(0)$
	$\mathscr{u}_1(1) \Vdash_{\mathbf{AB}} \mathscr{u}_2(1/2)$
	$\mathscr{u}_1(1) \Vdash_{\mathbf{AB}} \mathscr{u}_2(1)$
BB	$\mathscr{u}_1(0) \Vdash_{\mathbf{BB}} \mathscr{u}_2(0)$
	$\mathscr{u}_1(1/2) \Vdash_{\mathbf{BB}} \mathscr{u}_2(0), \mathscr{u}_2(1)$
	$\mathscr{u}_1(1) \Vdash_{\mathbf{BB}} \mathscr{u}_2(1)$

4.2 Distributed Logic Version

This amounts to flattening the Channel Theory diagram. The reason it is possible is that the channels have binary relations as their token sets. We will assume the following localities:

$in(\mathbf{A})$	a_1
$out(\mathbf{A})$	a_2
$in(\mathbf{B}_1)$	$\mathsf{b1}_1$
$out(\mathbf{B}_1)$	$\mathsf{b1}_2$
$in(\mathbf{B}_2)$	$\mathsf{b2}_1$
$out(\mathbf{B}_2)$	$\mathsf{b2}_2$

The locality graph is then

$$\mathsf{a}_1 \xrightarrow{\mathscr{A}} \mathsf{a}_2 \xrightarrow{\mathscr{AB}} \mathsf{b1}_1 \xrightarrow{\mathscr{B}1} \mathsf{b1}_2 \xrightarrow{\mathscr{BB}} \mathsf{b2}_1 \xrightarrow{\mathscr{B}2} \mathsf{b2}_2$$

There are the following theories in the localities, truth functionally valid formulas not involving modalities have been suppressed:

locality	theory	locality	theory
a_1	$(d=0) \supset [a^{\square}\rangle\,(q=0)$	$\mathsf{b2}_2$	$(q=0) \supset [b2^{\square}\cdot\rangle\,(d=0)$
	$(d=1) \supset [a^{\square}\rangle\,(q=1)$		$(q=1) \supset [b2^{\square}\cdot\rangle\,(d=1)$
a_2	$(q=0) \supset [a^{\square}\cdot\rangle\,(d=0)$	$\mathsf{b2}_1$	$(d=0) \supset [b2^{\square}\rangle\,(q=0)$
	$(q=1) \supset [a^{\square}\cdot\rangle\,(d=1)$		$(d=1) \supset [b2^{\square}\rangle\,(q=1)$
	$(q=0) \supset [ab^{\square}\rangle\,(d=0)$		$(d=0) \supset [bb^{\square}\cdot\rangle\,(q=0)$
	$(q=1) \supset [ab^{\square}\rangle\,(d=1)$		$(d=1) \supset [bb^{\square}\cdot\rangle\,(q=1)$
			$(d=1/2) \supset [bb^{\square}\cdot\rangle\,((q=0) \vee (q=1))$
$\mathsf{b1}_1$	$(d=0) \supset [b1^{\square}\rangle\,(q=0)$	$\mathsf{b1}_2$	$(q=0) \supset [b1^{\square}\cdot\rangle\,(d=0)$
	$(d=1/2) \supset [b1^{\square}\rangle\,(q=1/2)$		$(q=1/2) \supset [b1^{\square}\cdot\rangle\,(d=1/2)$
	$(d=1) \supset [b1^{\square}\rangle\,(q=1)$		$(q=1) \supset [b1^{\square}\cdot\rangle\,(d=1)$
	$(d=0) \supset [ab^{\square}\cdot\rangle\,(q=0)$		$(q=0) \supset [bb^{\square}\rangle\,(d=0)$
	$(d=1/2) \supset [ab^{\square}\cdot\rangle\,(q=1)$		$(q=1) \supset [bb^{\square}\rangle\,(d=1)$
	$(d=1) \supset [ab^{\square}\cdot\rangle\,(q=1)$		$(d=1/2) \supset [bb^{\square}\rangle\,((q=0) \vee (q=1/2) \vee (q=1))$
			$(d=1) \supset [bb^{\square}\rangle\,(q=1)$

4.3 Testing: Distributed Logic Version

This amounts to flattening the Channel Theory diagram. The reason it is possible is that the channels have binary relations as their token sets. We will assume the following localities:

Flip-Flop	locality appellation	states of the form $\langle \mathsf{D}, \mathsf{Q} \rangle$
Bit 0	a	$\{\langle 0,0\rangle, \langle 0,1\rangle\langle 1,0\rangle, \langle 1,1\rangle\}$
Bit 1	b1	$\{\langle 0,0\rangle, \langle 0,1\rangle\langle 1,0\rangle, \langle 1,1\rangle\}\cup$
		$\{\langle 0,1/2\rangle, \langle 1,1/2\rangle, \langle 1/2,0\rangle, \langle 1/2,1\rangle\}$
Bit 2	b2	$\{\langle 0,0\rangle, \langle 0,1\rangle\langle 1,0\rangle, \langle 1,1\rangle\}$

The locality graph is then

$$\mathsf{a} \xrightarrow{\mathcal{F}} \mathsf{b1} \xrightarrow{\mathcal{G}} \mathsf{b2}$$

The following table is the local relation specifications:

relation type	relation tuple
$\mathcal{A} : \mathsf{a} \rightharpoonup \mathsf{a}$	$\langle\langle 0,0\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 0,1\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 0,0\rangle, \langle 1,0\rangle\rangle$
	$\langle\langle 0,1\rangle, \langle 1,0\rangle\rangle$
	$\langle\langle 1,0\rangle, \langle 0,1\rangle\rangle$
	$\langle\langle 1,1\rangle, \langle 0,1\rangle\rangle$
	$\langle\langle 1,0\rangle, \langle 1,1\rangle\rangle$
	$\langle\langle 1,1\rangle, \langle 1,1\rangle\rangle$
$\mathcal{B}1 : \mathsf{b1} \rightharpoonup \mathsf{b1}$	$\langle\langle 0,0\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 0,1\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 0,0\rangle, \langle 1,0\rangle\rangle$
	$\langle\langle 0,1\rangle, \langle 1,0\rangle\rangle$
	$\langle\langle 1,0\rangle, \langle 0,1\rangle\rangle$
	$\langle\langle 1,1\rangle, \langle 0,1\rangle\rangle$
	$\langle\langle 1,0\rangle, \langle 1,1\rangle\rangle$
	$\langle\langle 1,1\rangle, \langle 1,1\rangle\rangle$
	$\langle\langle 0,0\rangle, \langle 1/2,0\rangle\rangle$
	$\langle\langle 0,1\rangle, \langle 1/2,0\rangle\rangle$
	$\langle\langle 1,0\rangle, \langle 1/2,1\rangle\rangle$
	$\langle\langle 1,1\rangle, \langle 1/2,1\rangle\rangle$
	$\langle\langle 1/2,0\rangle, \langle 0,1/2\rangle\rangle$
	$\langle\langle 1/2,1\rangle, \langle 1,1/2\rangle\rangle$
	$\langle\langle 1/2,0\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 1/2,1\rangle, \langle 1,1\rangle\rangle$
	$\langle\langle 0,1/2\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 0,1/2\rangle, \langle 1,0\rangle\rangle$
	$\langle\langle 1,1/2\rangle, \langle 0,1\rangle\rangle$
	$\langle\langle 1,1/2\rangle, \langle 1,1\rangle\rangle$
$\mathcal{B}2 : \mathsf{b2} \rightharpoonup \mathsf{b2}$	$\langle\langle 0,0\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 0,1\rangle, \langle 0,0\rangle\rangle$
	$\langle\langle 0,0\rangle, \langle 1,0\rangle\rangle$
	$\langle\langle 0,1\rangle, \langle 1,0\rangle\rangle$
	$\langle\langle 1,0\rangle, \langle 0,1\rangle\rangle$
	$\langle\langle 1,1\rangle, \langle 0,1\rangle\rangle$
	$\langle\langle 1,0\rangle, \langle 1,1\rangle\rangle$
	$\langle\langle 1,1\rangle, \langle 1,1\rangle\rangle$

For $\mathcal{X} \in \{\mathcal{A}, \mathcal{B}2\}$,

$$\mathcal{X} = \{\langle x, y\rangle \mid x_1 = y_2 \text{ and } x, y \in \mathit{Tok}(\mathsf{x})\}.$$

That is, in one time step, the input at D of state x at x is x_1 and x_1 is transferred to the output Q and hence x_1 becomes the second component y_2 of the state y.

The rest of the relations are

$$\mathcal{B}\mathsf{1} = \{\langle x, y\rangle \mid (x_1 = y_2) \vee (x_1 = 1/2) \text{ and } x, y \in \mathit{Tok}(\mathsf{x})\}.$$
$$\mathcal{AB} = \{\langle x, y\rangle \mid (x_1 = y_2 \text{ or } (x_1 = 1 \text{ and } y_2 = 1/2)) \text{ and } x \in \mathit{Tok}(\mathsf{a}) \text{ and } y \in \mathit{Tok}(\mathsf{b1})\}$$
$$\mathcal{BB} = \{\langle x, y\rangle \mid x_1 = y_2 \text{ or } (x_1 = 1/2 \text{ and } (y_2 = 0 \text{ or } y_2 = 1)) \text{ and } x \in \mathit{Tok}(\mathsf{b1}) \text{ and } y \in \mathit{Tok}(\mathsf{b2})\}$$

There are the following theories in the localities, truth functionally valid formulas not involving modalities have been suppressed:

locality	theory	locality	theory
a	$(d = 0) \supset [a^{\square}\rangle (q = 0)$	b2	$(q = 0) \supset [b2^{\square}\cdot\rangle (d = 0)$
	$(d = 1) \supset [a^{\square}\rangle (q = 1)$		$(q = 1) \supset [b2^{\square}\cdot\rangle (d = 1)$
a	$(q = 0) \supset [a^{\square}\cdot\rangle (d = 0)$	b2	$(d = 0) \supset [b2^{\square}\rangle (q = 0)$
	$(q = 1) \supset [a^{\square}\cdot\rangle (d = 1)$		$(d = 1) \supset [b2^{\square}\rangle (q = 1)$
	$(q = 0) \supset [ab^{\square}\rangle (d = 0)$		$(d = 0) \supset [bb^{\square}\cdot\rangle (q = 0)$
	$(q = 1) \supset [ab^{\square}\rangle ((d = 0) \vee$		$(d = 1) \supset [bb^{\square}\cdot\rangle (q = 1)$
	$(d = 1/2) \vee (d = 1))$		
b1	$(d = 0) \supset [b1^{\square}\rangle (q = 0)$	b1	$(q = 0) \supset [b1^{\square}\cdot\rangle (d = 0)$
	$(d = 1/2) \supset [b1^{\square}\rangle (q = 0) \vee$		$(q = 1/2) \supset [b1^{\square}\cdot\rangle (d = 1/2)$
	$(q = 1/2) \vee (q = 1))$		
	$(d = 1) \supset [b1^{\square}\rangle (q = 1)$		$(q = 1) \supset [b1^{\square}\cdot\rangle (d = 1)$
	$(d = 0) \supset [ab^{\square}\cdot\rangle (q = 0)$		$(q = 0) \supset [bb^{\square}\rangle (d = 0)$
	$(d = 1/2) \supset [ab^{\square}\cdot\rangle (q = 1)$		$(q = 1) \supset [bb^{\square}\rangle (d = 1)$
			$(q = 1/2) \supset [bb^{\square}\rangle ((d = 0) \vee (d = 1))$

The theories reflect the fact that while $1/2$ might be output by the flip-flop at b1, that $1/2$ is not received by the flip-flop at b2. The reasoning is that while the output of the flip-flop at b1 is metastable, the underlying assumption was that the flip-flop at b2 would not see it owing to the probability of that recognition is below the threshold for the modeling. Of course one could change the tables to reflect this non-deterministic behavior all the way through the flip-flop chain and then arrange any receiving circuits to handle this behavior.

This points up an underlying problem with digital circuity. There is always a probability of failure, however much of our analysis is devoted to correctly functioning circuits so we can be sure their correct behavior is as designed. There are strategies for fail-safe however these are not foolproof and will require a probability or a possibilistic based analysis.

5. CLOCK DOMAINS AS MODAL EFFECTS

An *effect type* for a clock domain is somewhat like an *aura* or color for all the elements (signals and components) in that clock domain.

5.1 Clock Domains and Modal Operators

The clock domain or aura modal operators are interpreted by relations which are reflexive, symmetric, and transitive. Assume a clock domain r interpreted by the relation $\mathcal{R}$. It must have the following properties

R1: reflexivity: $\mathcal{R} \subseteq \mathcal{I}$ ($\mathcal{I}$ is the identity relation),

R2: symmetry: $\mathcal{R} = \breve{\mathcal{R}}$, ($\breve{\mathcal{R}}$ is the relational converse of $\mathcal{R}$),

R3: transitivity: $\mathcal{R} \cdot \mathcal{R} \subseteq \mathcal{R}$ ($- \cdot -$ is relational composition).

These properties validate the following axioms in modal logic (the appellations are from Chellas [10]):

T: reflexivity: $[r^{\square}\rangle\, P \supset P$,

B: symmetry: $P \supset [r^{\square}\rangle\, [r^{\diamond}\rangle\, P$,

4: transitivity: $[r^{\square}\rangle\, P \supset [r^{\square}\rangle\, [r^{\square}\rangle\, P$.

The reason for the reflexivity axiom $[r^{\square}\rangle\, P \supset P$ stems from the modeling condition:

$$x \overset{\mathsf{h}}{\models} [r^{\square}\rangle\, P \overset{\mathsf{h}}{\supset} P \text{ iff } (\forall y(\mathcal{R}xy \text{ implies } y \overset{\mathsf{h}}{\models} P)) \text{ implies } x \overset{\mathsf{h}}{\models} P.$$

The rest of the conditions are similar and determine that $\mathcal{R}$ is an equivalence relation. If we think of tuple of the equivalence relation as an arc in a graph, then the equivalence classes determine what are known in graph theory as *cliques*. Hence, the clique that x finds itself within according to $\mathcal{R}$ is the proposition P simply because x must be related to every y in its clique. So there can only be a single P for which these axioms are true given any x. Different x may find themselves in different cliques.

A common mathematical systems concept is that of *residuation*. If $f : X \to Y$ and $g : Y \to X$ where X and Y are partial orders, then f and g are residuated if

$$fa \leq b \text{ iff } a \leq gb.$$

In Chellas [10], a system with symmetry is at least a *KB*. The system *KB* has the following pleasing property (see pp. 136 for more ways of axiomatizing *KB*) where the classical logic $\supset$ functions as the $\leq$:

$$\frac{[r^{\diamond}\rangle\, P \supset Q}{P \supset [r^{\square}\rangle\, Q}$$

This makes sense as the symmetry condition removes any distinction between forward and backward possibility and necessity. Hence this is stating a residuation property.

Let P be a formal logic condition we wish to have hold after a piece of computer code is executed. A *weakest precondition* for that piece of computer code is the least powerful statement if the statement holds before the computer code is executed, then P will hold after. The computer code can be associated with a relation between its inputs and its outputs. This situation reified in distributed logic (or modal logic) local necessity modal operator.

Let $[h^{\square}\rangle\, P$ be the weakest precondition with respect to the bit of computation to which $[h^{\square}\rangle$ refers, i.e, the computation associated with h is the relation $\mathcal{H}$, with the computation representing $\mathcal{H}$ in intensional form (i.e., a formula as opposed to a table). The following axioms should hold for any clock domains r, s, and localities h, k with $f : \mathsf{h} \rightharpoonup \mathsf{k}$ a clock synchronizer, and P is a proposition at h and Q a proposition at k:

Q1: $[r^{\square}\rangle\, P \overset{\mathsf{h}}{\supset} [r^{\square}\rangle\, [h^{\square}\rangle\, P$,

Q2: $[r^{\square}\rangle\, P \overset{\mathsf{h}}{\supset} [h^{\square}\rangle\, [r^{\square}\rangle\, P$,

Q3: $[f^{\circ}\rangle\, [k^{\square}\rangle\, Q \overset{\mathsf{h}}{\supset} [h^{\square}\rangle\, [f^{\circ}\rangle\, Q$,

Q4: $[f^{\circ}\cdot\rangle\, [h^{\square}\rangle\, P \overset{\mathsf{k}}{\supset} [k^{\square}\rangle\, [f^{\circ}\cdot\rangle\, P$,

Q5: $[r^{\circ}\rangle\, [f^{\circ}\rangle\, Q \overset{\mathsf{h}}{\supset} [f^{\circ}\rangle\, [s^{\circ}\rangle\, Q$,

Q6: $[s^{\circ}\cdot\rangle\, [f^{\circ}\cdot\rangle\, P \overset{\mathsf{k}}{\supset} [f^{\circ}\cdot\rangle\, [r^{\circ}\cdot\rangle\, P$.

These were stated with necessity because that corresponds to the weakest precondition. However, it seems easier to read them with possibility:

$$[r^{\circ}\rangle\, [h^{\circ}\rangle\, P \overset{\mathsf{h}}{\supset} [r^{\circ}\rangle\, P \qquad [h^{\circ}\rangle\, [r^{\circ}\rangle\, P \overset{\mathsf{h}}{\supset} [r^{\circ}\rangle\, P.$$

Axiom Q1 says that if some x is in some domain formalized as $\mathcal{R}$, then going through a local component transition leaves us within the same domain. Axiom Q2 says that if x transists locally to a state in a domain, then x must have already started in that domain. Both of these assume that $\mathcal{H}$, used to interpret $[h^{\square}\rangle$ and $[h^{\circ}\rangle$, is not the transition relation implementing a synchronizer but is merely the next state relation at h.

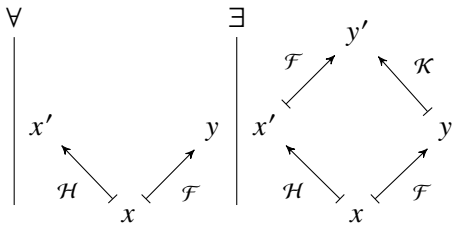
The Axioms Q3 and Q4 say that the synchronizer forces a forward and backwards simulation condition between the localities on either side of the synchronizer. The forward simulation says that transitions on h drive those on k, the backward simulation says that transitions on k were driven by those on h. Simulations relations and axioms are listed below.

Axioms Q5 and Q6 says the clock domains are related by a proverse and converse simulations respectively. It must be noted that the synchronizer requires state changes or it could not function as a synchronizer; it would be as though the clocks were turned off if this were not the case.

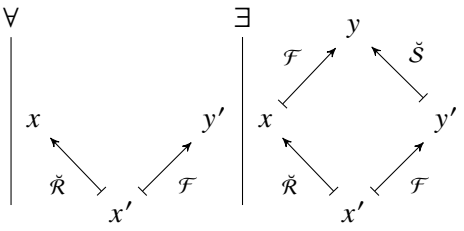
The simulation relations are summarized in the follow:

Name	Frame Condition	Axiom
Forward Simulation	$\mathcal{F}xy$ and $\mathcal{H}xx'$ implies $\exists y'(\mathcal{K}yy'$ and $\mathcal{F}x'y')$	$[f^\circ\rangle\,[k^\square\rangle\,Q \overset{\mathsf{h}}{\supset} [h^\square\rangle\,[f^\circ\rangle\,Q$
Backward Simulation	$\mathcal{F}x'y'$ and $\mathcal{K}yy'$ implies $\exists x(\mathcal{H}xx'$ and $\mathcal{F}xy)$	$[f^\circ\cdot\rangle\,[h^\square\cdot\rangle P \overset{\mathsf{k}}{\supset} [k^\square\cdot\rangle\,[f^\circ\cdot\rangle P$
Proverse Simulation	$\mathcal{F}x'y'$ and $\mathcal{R}xx'$ implies $\exists y(\mathcal{S}yy'$ and $\mathcal{F}xy)$	$[r^\circ\rangle\,[f^\circ\rangle\,Q \overset{\mathsf{h}}{\supset} [f^\circ\rangle\,[s^\circ\rangle\,Q$
Converse Simulation	$\mathcal{F}xy$ and $\mathcal{S}yy'$ implies $\exists x'(\mathcal{R}xx'$ and $\mathcal{F}x'y')$	$[s^\circ\cdot\rangle\,[f^\circ\cdot\rangle P \overset{\mathsf{k}}{\supset} [f^\circ\cdot\rangle\,[r^\circ\cdot\rangle P$

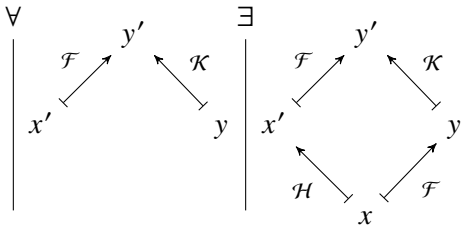
Diagrammatically (but not in the diagrams of category theory, the arrows are merely tuples in the relation indicated by the arrow's label),



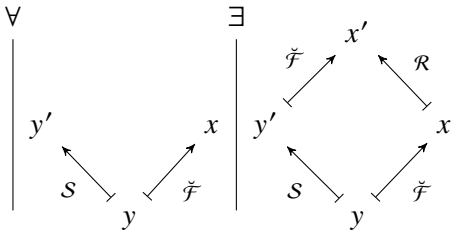
Forward



Proverse



Backward



Converse

The intermediate localities of the example can be made to recede into the background by taking the relational composition of the relations representing the flip-flops and using the result as $\mathcal{R}$ where the input the first flip-flop comes from a locality h in the r domain and the output of the last modality is in the locality k in the s domain.

5.2 An Aura of Localities

To step back a bit, the structure appears to be an iteration of Distributed Logic where there are localities of localities. The second level are the aurae and correspond, in this particular instance, to clock domains. The synchronizers are represented by simulation relations between the auras.

It might make sense to think of clock domains as consisting of a collection of propositions of localities. Think of the synchronizers as pairing localities as the elements of a relation, $\mathcal{F}$. So $\langle \mathsf{h}, \mathsf{k}\rangle \in \mathcal{F}$ iff there is a

synchronizer from h to k. This relation $\mathcal{F}$ is not an equivalence relation because there is no guaranteed that $\langle \mathsf{k}, \mathsf{h} \rangle \in \mathcal{F}$. However, I do not know if anything interesting follows from this.

An alternate view without jumping set theoretical type levels is to consider an equivalence relation C'_i on all localities i within a single clock domain I. The clock domain itself is another relation

$$C_I = \bigcup_{i \in I} C'_i.$$

In this case, C_I is an equivalence relation and each clique corresponds to a locality.

6. ESTEREL, LUSTRE, AND SIGNAL

Most previous work, e.g., Esterel [11], Lustre [12], Signal [13], etc. use synchronous signals. Asynchronous hardware circuits are not widely used, so these languages might lend themselves to FPGA applications. We are not aware of this happening at the moment.

In Esterel, programs are interpreted as state machines. Programs in Lustre [12] are interpreted as dataflows. Programs in Signal [13] are interpreted as logical specifications. Lustre is closest to ReWire in the sense that dataflow languages are similar to functional languages.

6.1 Clocks

Each of the languages uses an implicit notion of clocking and provide primitives to capture two senses of "signals in time": simultaneity and rising edges. Implicitly, a design is composed of several clock zones, each zone has its own notion of clock. To route signals between zones, one must use some form of synchronization. Simultaneity is useful for polling a signal to catch its rising edge. Rising edges are used with latches to catch the first time a signal goes true. The latter can be derived from the former but only when the polling is happening at faster rate than the clock of the region from which the signal was sent.

In none of these languages are the actual rates of the clocks used. They would be unknown at compile time. Rather, the facilities of the languages are used to place a partial order on when signals are "present". So one must get used to the separation of real clock times and an ordering on events that is imposed by the program structure. It is this latter that the languages express. In that sense, one might never need an actual variable representing a clock signal in the languages.

There are at least two difficulties that must be overcome:

- Clock C runs slower than clock C', and
- Clock C runs faster than clock C'.

Signals in each of the clock domains C and C' are considered to remain set throughout an entire clock slice and can only change on the rising edge of each clock waveform.

Consider a computation φ running on clock C and a computation φ' running on clock C' and they wish to communicate. Assume is one signal s with which they wish to communicate. φ sends signal s to φ' which receives it as s'. In order to force synchronization between φ and φ', the signal must be coerced into a synchronous signal in the sense that φ can send it and φ' can receive it where the transaction is considered atomic. To implement the atomic action, there are two primitives notions: sampling and latching.

If clock C runs slower than C', then φ' can sample the signal s' at each clock tick. If C runs faster than clock C', then the signal s must be latched so that its value is retained until φ' can get around to reading it.

6.2 Esterel

Esterel allows one to sample a signal every clock cycle with the signal being generated on a different clock than the sampler. A sampler is a busy wait. The sampler triggers for every tick of its clock, runs to completion, and then awaits its next clock tick. This captures the rising edge of a signal. The current state of the signal is saved for every clock tick. When the next clock tick arrives, the current value of the signal can be compared to the previous value and if the signal goes from low to high, then the rising edge has been recognized.

To catch rising edges below the level of coding the above in Esterel, there is the "await" operation. This sets up what is essentially a latch. When run in parallel with code polling the effect of the wait, the rising edge of a signal is recognized.

6.3 Lustre

From [12],

> In LUSTRE any variable and expression denotes a flow, i.e., a pair made of a possibly infinite sequence of values of a given type; a clock, representing a sequence of times. A flow takes the nth value of its sequence of values at the nth time of its clock. Any program, or piece of program has a cyclic behavior, and that cycle defines a sequence of times which is called the basic clock of the program: a flow whose clock is the basic clock takes its nth value at the nth execution cycle of the program. Other, slower, clocks can be defined, thanks to boolean-valued flows: the clock defined by a boolean flow is the sequence of times at which the flow takes the value true.

Lustre has primitive called *pre* that allows one to capture the previous value of a signal. Also, it has an operator −> that says what is on the left is the initial value for when the code is run and what is right will be succeeding values. As an example,

```
node COUNTER(va1_init, Val_incr: int; reset: bool) returns (n: int);
let
  n = Val_init -> if reset then
    Val_init else pre(n) + Val_incr;
tel.
```

which implements a counter. The "tel." is the end of the module indicator. These modules are *nodes* in a flow graph.

Similarly to Esterel, has an operator *current* which picks up the current value of a signal. It has an another operator *when* for picking up the rising edge of signal.

6.4 Signal

Signal works with three-values for signals, t, f, and $\perp$. A signal Y can thus be t or f when "present" and is undefined, $\perp$, when not present. Signal has a *when* operator which samples an input:

$$Y := X \text{ when } B$$

where X, Y, and B are Boolean. This statement sets Y to X if B is present and undefined otherwise. The statement

$$Y := U \text{ default } V$$

"merges" U and Y; when U is present, then $Y = U$. Else, if V is present, then $Y = V$. If neither U or V is present, then Y is $\perp$.

All signals have clocks associated with them. The clocks for variables on the left of := are assigned clocks depending upon the clocks of variables on the right of :=. This has the effect of the clocks being locally synchronized or determined in a retail fashion by the compiler.

The operator \$ is the control theory equivalent of the z^{-1} transform in that it allows one to peek at the k-th value of a signal before the present time:

$$Z := Y \$ 4$$

peeks at the value of Y four time clicks back from the current tick at which the code is being executed. Related to this is a *window* operator:

$$VZ := Y \text{ window } 3$$

assigns the vector VZ the current and two previous values of the signal Y.

There is a prioritized merge:

$$Y := U \text{ default } V$$

which sets the value of Y to U if U is present. Otherwise it sets the value of Y to V if V is present. If neither are present, Y gets $\perp$.

The locution

$$T := \text{event } X$$

defines the event type signal T and its occurrences are simultaneous with those of X. In short, it represents the clock of X.

The locution

$$T \;:=\; \text{when } B$$

defines the event type signal T which is present whenever B is present and true, and is undefined otherwise.

Clocks are normally left implicit. However, it is possible to reify a clock to be an actual signal. The locution

$$T \;:=\; \text{event } B$$

defines the event type tignal T whenever B is present and B is true, and delivers nothing otherwise.

In addition, there are some constraints on clocks that can be specified:

$$X \hat{=} Y$$

and

$$X \hat{<} Y$$

The first specifies that Y has the same clock as X and the second specifies that X is no more frequent than Y. Signal contains a delay operator that references the value of a signal at the previous tick as opposed to the current tick.

These operators can be put together to form a cell:

$$Y \;:=\; X \text{ cell } B$$

for B a Boolean signal. This delivers at the output of Y either the value X if present or the previous value of X when B is currently presented and true. The cell can be implemented with

$$\begin{aligned} Y &:= X \text{ default } Y \$ 1 \\ Y &\hat{=} (\text{event } X) \text{ default } (\text{when } B) \end{aligned}$$

Y is set to X if present, or the previous value of Y if X is not present. In addition (second line), Y is specified to have a clock rate not greater than that of the event of X if present or defaults to that of B whenever B is present and true.

There is a caveat to the above, the actual clock rate of any signal is not known at compile time. Rather, the instructions allow one to place a partial order on the implicit clocks associated with signals. Thus the code can implicitly work with clocks yet need not ever know their rate. Their rate is supplied by the underlying hardware when the code is executed.

REFERENCES

1. G. Allwein, "A Qualitative Framework for Shannon Information Theories," Proceedings of the Proceedings of the New Security Paradigms Workshop, 2004 (ACM Press), 2005, pp. 23 – 31.

2. G. Allwein, Y. Yang, and W. L. Harrison, "Decision Theory via Channel Theory," Proceedings of the Proceedings of the Logic in Cognitive Science Conference, 2010, Logic and Logical Philosophy Journal (The Nicolaus Copernicus University Press), 2011, pp. 81–110.

3. J. Barwise and J. Seligman, *Information Flow: The Logic of Distributed Systems* (CUP, 1997). Cambridge Tracts in Theoretical Computer Science 44.

4. G. Allwein, W. L. Harrison, and D. Andrews, "Simulation logic," *Logic and Logical Philosophy* **23**(3), 277–299 (2014).

5. G. Allwein and W. L. Harrison, "Distributed Modal Logic," in K. Bimbó, ed., *J. Michael Dunn on Information Based Logic: Outstanding Contributions to Logic*, pp. 331–362 (Springer-Verlag, 2016).

6. A. Procter, W. L. Harrison, I. Graves, M. Becchi, and G. Allwein, "A Principled Approach to Secure Multi-Core Processor Design with ReWire," Proceedings of the Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (ACM Digital Library), 2016.

7. W. L. Harrison, A. Procter, I. Graves, M. Becchi, and G. Allwein, "A Programming Model for Reconfigurable Computing Based in Functional Concurrency," Proceedings of the Proceedings of the 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC 2016) (IEEE), 2016, pp. 1–8.

8. T. Dave, A. Jain, and D. Jain, "Synchronizer techniques for multi-clock domain SoCs & FPGAs," https://www.edn.com/electronics-blogs/day-in-the-life-of-a-chip-designer/4435339/Synchronizer-techniques-for-multi-clock-domain-SoCs, 2014.

9. I. S. Moskowitz, L. Chang, and R. E. Newman, "Capacity is the Wrong Paradigm," Proceedings of the Proc. New Security Paradigms Workshop, Sept. 23-26 (ACM Press), 2002, pp. 114–126.

10. B. F. Chellas, *Modal Logic: an introduction* (Cambridge University Press, 1980).

11. G. Berry and E. Sentovich, "Multiclock Esterel," Proceedings of the Correct Hardware Design and Verification Methods, CHARME 2001, volume Lecture Notes in Computer Science, vol 2144 (Springer), 2001.

12. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE* **79**(9), 1305–1320 (1991).

13. P. LeGuernic, T. Gautier, M. L. Borgne, and C. L. Maire, "Programming real-time applications with SIGNAL," *Proceedings of the IEEE* **79**(9), 1321–1336 (1991).