Specification and Verification of Gate-Level VHDL Models of Synchronous and Asynchronous Circuits

David M. Russinoff

Technical Report 99 May 10, 1994

Computational Logic, Inc.
1717 West Sixth Street, Suite 290
Austin, Texas 78703-4776

TEL: +1 512 322 9951
FAX: +1 512 322 0656

EMAIL: russ@cli.com

This work was sponsored in part at Computational Logic, Inc. by National Aeronautics and Space Administration Langley Research Center (NAS1-18878). The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., NASA Langley Research Center or the U.S. Government.

Abstract

We present a mathematical definition of a hardware description language (HDL) that admits a semantics-preserving translation to a subset of VHDL. Our HDL includes the basic VHDL propagation delay mechanisms and gate-level circuit descriptions. We also develop formal procedures for deriving and verifying concise behavioral specifications of combinational and sequential devices. The HDL and the specification procedures have been formally encoded in the computational logic of Boyer and Moore, which provides a LISP implementation as well as a facility for mechanical proof-checking. As an application, we design, specify, and verify a circuit that achieves asynchronous communication by means of the biphase mark protocol.

1 Introduction

During the course of the design process, a typical hardware device is modeled at various levels of abstraction. The most abstract representation is generally a sequential machine, which is initially derived from a given behavioral specification. This model is then gradually refined to produce a concrete representation, such as a network of gates, which is more amenable to implementation.

A hardware design is validated by demonstrating the equivalence of these representations. This is most commonly effected through simulation. The VHSIC Hardware Description Language (VHDL), which is rapidly becoming an industry standard, supports modeling and simulation of circuits at all stages of the design process. In VHDL, a circuit component is represented as an *entity*, which may be associated with various alternative *architectures*. An architecture may either specify an abstract *behavioral* description of a device, or provide a concrete *structural* definition in terms of simpler components. The equivalence of architectures may be confirmed through comparative simulations. Once a sufficiently low-level VHDL architecture has been derived and validated in this manner, it may be implemented directly, even automatically.

However, since exhaustive testing of complex circuits is impractical, the effectiveness of simulation as a validation method is limited. An alternative approach is provided by formal hardware verification. The object of formal verification is to prove mathematically that a given model of a hardware device satisfies a given behavioral specification. Once this is accomplished, the design problem is reduced to that of implementing the model.

Naturally, the utility of this approach requires that the specification is sufficiently abstract to provide a comprehensive description of functionality, and that the model is sufficiently concrete to allow straightforward implementation. Unfortunately, hardware verification techniques have been deficient in this regard: proofs of of high-level specifications generally depend on similarly high-level implicit assumptions. In particular, verification of sequential machine behavior has been achieved only by basing the hardware model itself on the sequential machine concept.

For the purpose of addressing this problem, we have designed a formal hardware simulation language that admits a semantics-preserving translation to a subset of VHDL. The language is based on the paradigm of *event-driven simulation* and the basic signal propagation and delay mechanisms of VHDL. In particular, it includes the VHDL "delta-delay" mechanism, which complicates its definition considerably. Since this mechanism is irrelevant to our present purpose, we shall present here a simplified version of the language, in which all delays are required to be positive. The full language definition may be found in [6].

The language that we shall describe includes behavioral modules as primitives, which we use to model gates, and structural modules, with which we represent hierarchically constructed circuits. Our goal is to derive and verify abstract specifications of these gate-level models.

First, we consider the relatively simple class of *combinational* circuits, i.e., circuits that are free of cyclic paths. Each output of such a circuit is naturally associated with a certain Boolean function of the inputs. This association is commonly stated as follows: the value of an output at any time may be computed by applying the associated function to the current input values. Obviously, this description is valid only with respect to hardware models that ignore propagation delay. We shall derive a more accurate specification of combinational circuits and verify its validity in the context of our model.

The analysis of sequential circuits is considerably more complicated. While the abstract sequential machine model is well understood, its precise relationship with the actual behavior of the hardware that it is intended to describe is not. The sequential machine characterization is traditionally based on the extravagant assumption that signal values may change only at discrete points occurring at regular time intervals. This allows the behavior of a signal to be represented abstractly as a sequence of values. The value of an output over a given interval is then expressed as a function of the sequence of past input values. Of course, the underlying model again must disregard propagation delay. This approximation seems questionable, since the functionality of the basic state-holding elements generally depends critically on the presence of delays.

We shall treat a class of sequential circuits that may be characterized as *synchronous resettable rising-edge-triggered* devices. The basic memory element employed in their construction is a resettable clocked d-flip-flop, composed of nand gates. After verifying a precise specification of the behavior of this component, we establish a procedure for deriving high-level sequential machine descriptions for this class of circuits, and prove a theorem that gives a precise statement of the relationship between the sequential machine description of a circuit and its behavior as defined by our gate-level semantics.

One advantage of our approach is that we can effectively model asynchronous communication between individually synchronous processors. In fact, we shall present the definition of a circuit in our language that consists of two independently clocked sequential modules, and prove that communication between them is achieved by means of the well known biphase mark protocol [5]. The circuit design and the proof are both based on Moore's work on asynchrony [4], which includes a formal model of asynchronous communication and a rigorous formulation of the protocol.

The syntax and semantics of our language are both based on the *S-expressions* of LISP. This decision was motivated by our desire to support its analysis with the use of the Nqthm system of Boyer and Moore [1]. Nqthm is based on a constructive formal logic for which the intended model is the domain of S-expressions. Thus, there is a correspondence between the formulas of this logic and informal propositions about S-expressions. A user of the system may extend the logic by adding axioms that correspond to definitions of computable functions over this domain.

Mechanical support for the Nqthm logic is provided by a LISP implementation that includes (1) an evaluator that computes values of functions defined in the logic, and (2) a theorem prover that may be used to derive logical consequences of the axioms. Since these theorems may be interpreted as propositions about functions of S-expressions, the prover may be used to verify mechanically the correctness of properties of these functions that have been derived by traditional (informal) mathematical methods.

All of the functions involved in the construction of our language, which we describe informally, meet the computability requirement for encoding as Nqthm definitions [1]. In fact, we have developed an Nqthm theory that formalizes these functions, including the module recognizers that form the syntax of the language, the interpreter that constitutes its semantics, and various procedures for deriving behavioral specifications of its programs. Thus, we have a complete LISP implementation of our language, provided by the Nqthm evaluator.

Moreover, all of our results, which are justified by informal (but mathematically rigorous) proofs, correspond in a natural way to Nqthm formulas. Thus, these proofs could, in principle, be checked mechanically by the Nqthm prover, thereby increasing our confidence in their validity at the the expense of some effort. In general, we have found it practical to employ the prover to check those proofs that involve extensive computation or detailed case analysis, while relying instead on the conventional social review process to detect any errors in our more intelligible proofs. In this instance, most of the results pertaining to specific circuits, including the components of the biphase mark implementation, have been mechanically verified, but at the time of this writing, most of the more general theorems have not.

Another benefit of the Nqthm formalization is that it provides a basis for a LISP implementation of the translator from our language to VHDL [3]. This potentially allows commercial VHDL synthesis tools to be used to implement our programs in silicon. As another application of more immediate interest, we have executed the translations of many of our programs using the Vantage VHDL simulator. For the simulations that we have tested, which include all of those described herein, the Vantage results were identical to those produced by our LISP-based interpreter. Aside from the practical advantage of increased efficiency, this offers evidence that we have achieved our goal of semantically capturing the VHDL subset in which we are interested. Indeed, in the absence of any comprehensive formal semantics for VHDL itself, these empirical observations provide the most convincing evidence possible.

2 Definition of the Language

2.1 S-expressions

Along with the set $\mathbf{N}$ of natural numbers, we posit a set $\mathbf{B} = \{\mathcal{T}, \mathcal{F}\}$ and an infinite set $\mathbf{L}$, the elements of which are called *Boolean* and *literal atoms*, respectively. These three sets are assumed to be pairwise disjoint, and any element of their union is called an *atom*. We further assume that no atom is an ordered pair of atoms, and we recursively define an *S-expression* to be an atom or an ordered pair of S-expressions. $\mathbf{S}$ denotes the set of all S-expressions. Three basic operations on $\mathbf{S}$ are defined: If $z = (x, y) \in \mathbf{S} \times \mathbf{S}$, then $car(z) = x$, $cdr(z) = y$, and $cons(x, y) = z$.

We also assume the existence of various distinct literal atoms, which we shall mention as we proceed. Among these is the atom `NIL`. A *list* is an S-expression that is either `NIL` or an ordered pair $z \in \mathbf{S} \times \mathbf{S}$ such that $cdr(z)$ is a list. The list `NIL` is denoted alternatively as $()$, and a non-`NIL` list z is denoted as $(a_1 \ldots a_n)$, where $a_1 = car(z)$ and $(a_2 \ldots a_n)$ denotes $cdr(z)$. In this case, n is the *length* of z, and $a_1, \ldots, a_n$ are its *members*. For $1 \leq i \leq n$, $nth(i, z)$ is defined to be a_i. A list is a *bit vector* if each of its members is a Boolean atom.

A function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ is an *n-ary Boolean function*. The following Boolean functions are called *elementary*: the 0-ary functions *t0* and *f0*, with values $\mathcal{T}$ and $\mathcal{F}$, respectively; the unary function $not1$; the binary functions *and2*, *or2*, *nand2*, *nor2*, *xor2*; the ternary functions *and3*, *or3*, *nand3*, *nor3*, *xor3*; the quaternary functions *and4*, *or4*, *nand4*, *nor4*, and *xor4*; and the quinary functions *and5*, *or5*, *nand5*, *nor5*, and *xor5*. The definitions of these functions are assumed to be understood.

For the purpose of encoding Boolean function calls, we also assume that each elementary Boolean function f is associated with a unique literal atom $\bar{f}$ that is denoted with the same name as f. Thus, the function *not1* is associated with the literal atom $\overline{not1} =$ `NOT1`. We define a *Boolean term* over a list L of distinct literal atoms to be an S-expression that is either (a) a member of L, or (b) a list $(\bar{f} \tau_1 \ldots \tau_n)$, where f is an n-ary elementary Boolean function and each τ_i is an Boolean term over L.

Let $L = (s_1 \ldots s_k)$ be a list of distinct literal atoms and let $V = (v_1 \ldots v_k)$ be a bit vector. Then $pairlist(L, V)$ is the list $A = ((s_1, v_1) \ldots (s_k, v_k))$, which is called an *association list*. If τ is a Boolean term over L, then we define $eval(\tau, A)$ to be

(a) v_i, if $\tau = s_i$, or (b) $f(eval(\tau_1, A), \ldots, eval(\tau_n, A))$, if $\tau = (\bar{f}\ \tau_1\ \ldots\ \tau_n)$.

2.2 Waveforms

In the simplified version of our language on which the present work is based, our model of time is the set $\mathbf{N}$. Thus, a moment is represented by a natural number, which we take arbitrarily to be the number of picoseconds elapsed since the beginning of a simulation. For $t_1, t_2 \in \mathbf{N}$, the interval $\{t \in \mathbf{N} : t_1 \leq t < t_2\}$ will be denoted as $[t_1, t_2)$.

An *event* is an ordered pair $e = (v, t)$, where $v = value(e) \in \mathbf{B}$ and $t = time(e) \in \mathbf{N}$. Let $w = ((v_n, t_n)\ \ldots\ (v_0, t_0))$ be a list of events. If $t_i > t_{i-1}$ and $v_i \neq v_{i-1}$ for $0 < i \leq n$, and $t_0 = 0$, then w is a *waveform*. We define $\hat{w} : \mathbf{T} \to \mathbf{B}$ by $\hat{w}(t) = v_j$, where j is the greatest value of i satisfying $t_i \leq t$; $\hat{w}(t)$ is called the *value of* w *at* t. Note that $\hat{w}_1 = \hat{w}_2$ iff $w_1 = w_2$. If $t = t_j$, then we shall say that w *has a new value at* t. We also define the *history of* w *relative to* t to be the waveform $hist(w, t) = ((v_j, t_j)\ \ldots\ (v_0, t_0))$.

A *packet* is a list of waveforms, $p = (w_1\ \ldots\ w_n)$, $n \geq 0$. For any $t \in \mathbf{N}$, the *value of* p *at* t is the bit vector $\hat{p}(t) = (\hat{w}_1(t)\ \ldots\ \hat{w}_n(t))$; p *has a new value at* t if any member of p does. The *history* of p relative to t is the packet $hist(p, t) = (hist(w_1, t)\ \ldots\ hist(w_n, t))$.

The behavior of each signal occurring in a circuit will be modeled as a waveform. When a waveform is considered in the context of a current time t_0, each of its members e is viewed as a past, current, or future event, according to the relationship between $time(e)$ and t_0. Past and present events are immutable, but future events are subject to deletion as they are superceded by newly scheduled events.

The fundamental operation of the simulator is the updating of a waveform w at a time t_0 by scheduling a new event $e = (v, t_v)$, where $t_0 < t_v$. This may be performed by either of two procedures, corresponding to the *transport* and *inertial* delay modes of VHDL. The simpler of these is transport mode, in which each event (v', t') with $t' \geq t_v$ is deleted from w, and e is then *consed* to the result, unless that result already has value v at t_v. The updated waveform w' is computed as the value of *transport*(w, v, t_v), which is defined recursively as follows:

(1) Let $car(w) = (v_f, t_f)$. If $t_f \geq t_v$, then $w' = transport(cdr(w), v, t_v)$; otherwise:

(2) If $v_f = v$, then $w' = w$; otherwise:

(3) $w' = cons((v, t_v), w)$.

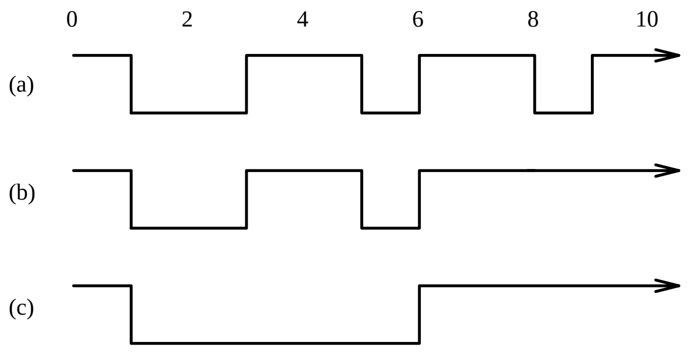


Figure 1: Transport and Inertial Delay

Alternatively, w' may be described in terms of the function $\hat{w}'$:

$$\hat{w}'(t) = \begin{cases} v & \text{if } t \geq t_v \\ \hat{w}(t) & \text{if } t < t_v. \end{cases}$$

Inertial mode is somewhat more complicated: every event (v', t') with $t' > t_0$ is deleted from w, and if $\hat{w}(t_0) \neq v$, then a single event with value v is consed to the result. If $\hat{w}(t_v) = v$, then the time of this event is the time of the last event of w that precedes t_v; otherwise, it is t_v. Note that this procedure takes the current time t_0 as an additional argument, and requires that $t_0 < t_v$. The recursive definition of $w' = \mathit{inertial}(w, v, t_0, t_v)$ is given as follows:

(1) Let $\bar{w} = \mathit{hist}(w, t_0)$. If $\hat{w}(t_0) = v$, then $w' = \bar{w}$; otherwise:

(2) Let $\mathit{car}(w) = (v_f, t_f)$. If $t_f \geq t_v$, then $w' = \mathit{inertial}(\mathit{cdr}(w), v, t_0, t_v)$; otherwise:

(3) If $v_f = v$, then $w' = \mathit{cons}((v, t_f), \bar{w})$; otherwise:

(4) $w' = \mathit{cons}((v, t_v), \bar{w})$.

The difference between the two propagation modes is illustrated in Fig. 1. The diagram labelled (a) represents the waveform

$$w = ((\mathcal{T}, 9)\,(\mathcal{F}, 8)\,(\mathcal{T}, 6)\,(\mathcal{F}, 5)\,(\mathcal{T}, 3)\,(\mathcal{F}, 1)\,(\mathcal{T}, 0)).$$

The results of updating w at time 1 by scheduling an event with time 7 and value $\mathcal{T}$, in both transport and inertial modes, are

$$\mathit{transport}(w, \mathcal{T}, 7) = ((\mathcal{T}, 6)\,(\mathcal{F}, 5)\,(\mathcal{T}, 3)\,(\mathcal{F}, 1)\,(\mathcal{T}, 0))$$

and

$$inertial(w, \mathcal{T}, 1, , 7) = ((\mathcal{T}, 6)\ (\mathcal{F}, 1)\ (\mathcal{T}, 0)),$$

as shown in (b) and (c), respectively.

Transport mode is often used to model wires (along which pulses of arbitrarily small duration are propagated to the delayed signal), while gate outputs are generally modeled by inertial delay. The latter, which is the VHDL default mode, will be used in all of our examples.

The following is a useful summary of both propagation functions. Each result may be proved by a straightforward induction. Note that (b) is consistent with our earlier informal observation that past and present events are immutable:

Lemma 2.1 *Let w be a waveform, let t_0, t_1, and t_v be time objects with $t_0 < t_v$, and let w' be either $transport(w, v, t_v)$ or $inertial(w, v, t_0, t_v)$. Then*

(a) $\hat{w}'(t) = v$ for $t \geq t_v$;
(b) $\hat{w}'(t) = \hat{w}(t)$ for $t \leq t_0$;
(c) if $t_1 \leq t_0 \leq t_2 \leq t_v$ and $\hat{w}(t) = u$ for $t \in [t_1, t_2)$, then $\hat{w}'(t) = u$ for $t \in [t_1, t_2)$.

A similar induction shows that both procedures are "idempotent" in the following sense:

Lemma 2.2 *If w is a waveform and t_0, t_v, t'_0, t'_v are time objects with $t_0 < t_v$, $t'_0 < t'_v$, $t_0 < t'_0$, and $t_v < t'_v$, then*

(a) $transport(transport(w, v, t_v), v, t'_v) = transport(w, v, t_v)$;
(b) $inertial(inertial(w, v, t_0, t_v), v, t'_0, t'_v) = inertial(w, v, t_0, t_v)$.

2.3 Behavioral Modules

The simplest programs of our language are the behavioral modules, which contain explicit information concerning propagation delay and the functional dependence of outputs on inputs.

A *behavioral module* is a list $M =$ (`BEHAV` $I\ O\ T\ P\ D$), where

(1) `BEHAV` is the identifying literal atom for modules of this type;

(2) $I = I(M) = (r_1\ \ldots\ r_m)$ is a list of literal atoms called the *inputs* of M;

(3) $O = O(M) = (s_1\ \ldots\ s_n)$ is a list of literal atoms called the *outputs* of M;

(4) $T = T(M) = (\tau_1\ \ldots\ \tau_n)$ is a list of elementary Boolean terms over $I(M)$, called the *output terms* of M;

(5) $D = D(M) = (d_1 \ldots d_n)$ is a list of positive natural numbers, the *delays* of M;

(6) $P = P(M) = (p_1 \ldots p_n)$ is a list of literal atoms called the *propagation modes* of M, each of which is either `TRANSPORT` or `INERTIAL`.

The members of the list $(r_1 \ldots r_m\ s_1 \ldots s_n)$ are required to be distinct and are called the *signals* of M.

Note that each output is associated with a term, a mode, and a delay. If every term is either an atom or a list of atoms, (i.e., contains no nested function calls), then M is *primitive*.

Gates are generally modeled as primitive modules with inertial delays. For example, we represent a simple 2-input nand gate as the primitive module `nand2`:

```
(BEHAV (A B) (C) ((NAND2 A B)) (2000) (INERTIAL))
```

We may define a similar behavioral module, with n inputs and 1 output, corresponding to each elementary n-ary Boolean function, arbitrarily taking the delay to be 2000 in each case. In the sequel, we shall refer to these primitive modules without explicitly listing their definitions.

Another example of a behavioral module is the 1-bit adder `adder1`:

```
(BEHAV (A B C) (L H)
  ((XOR3 A B C) (OR2 (AND2 A (OR2 B C)) (AND2 B C)))
  (12000 10000)
  (INERTIAL INERTIAL))
```

The two outputs of this module represent the 2-bit sum of the three input bits. Since the higher-order "carry" output bit is not expressed as an elementary function of the inputs, this is not a primitive module.

Let $s = nth(j, O(M))$ be an output of a behavioral module M and let $\tau = nth(j, T(M))$ be the corresponding term. For any bit vector V of the same length as $I(M)$, we define the *combinational value* of s w.r.t. V as

$$cv(s, V, M) = eval(\tau, pairlist(I(M), V)).$$

We shall say that a list of waveforms is an *input* (resp., *output*) *packet* for a module M if it has the same length as $I(M)$ (resp., $O(M)$). The semantics of behavioral modules are defined by a function *exec* of four arguments: (1) a module M, (2) an input packet p_{in} for M, (3) an output packet $p_{out} = (w_1 \ldots w_n)$ for M, and (4) a time object t_0. The value of $exec(M, p_{in}, p_{out}, t_0)$ is the updated output packet $p'_{out} = (w'_1 \ldots w'_n)$ that results from "executing" M at t_0. It is defined as follows: For $i = 1, \ldots, n$, let v_i be the combinational value of $nth(i, O(M))$ w.r.t. $\hat{p}_{in}(t_0)$, and let $t_i = t_0 + nth(i, D(M))$. Then w'_i is either $transport(w_i, v_i, t_i)$ or $inertial(w_i, v_i, t_0, t_i)$, according to $nth(i, P(M))$.

Our first observation concerning the behavior of *exec* is that its value depends only on the current values of the input:

Lemma 2.3 *Let p_1 and p_2 be input packets and let p_{out} be an output packet for a behavioral module M. For any $t_0 \in \mathbf{T}$, if $\hat{p}_1(t_0) = \hat{p}_2(t_0)$, then $exec(M, p_1, p_{out}, t_0)$ $= exec(M, p_2, p_{out}, t_0)$.*

Two other basic properties may be derived as consequences of Lemmas 2.1(b) and 2.2:

Lemma 2.4 *Let p_{in} and p_{out} be an input packet and an output packet for a behavioral module M. For any $t_0 \in \mathbf{T}$, $hist(exec(M, p_{in}, p_{out}, t_0), t_0) = hist(p_{out}, t_0)$.*

Lemma 2.5 *Let p_{in} and p_{out} be an input packet and an output packet for a behavioral module M and let t_0 and t_1 be time objects. If $t_0 < t_1$ and $\hat{p}_{in}(t_0) = \hat{p}_{in}(t_1)$, then $exec(M, p_{in}, exec(M, p_{in}, p_{out}, t_0), t_1) = exec(M, p_{in}, p_{out}, t_0)$.*

2.4 Structural Modules

Our language also includes modules that represent hierarchically constructed circuits. These structures contain information concerning interconnections among the modules of which they are composed.

A *structural module* is a list $M =$ (`STRUCT` $I\ O\ S\ LI\ LO$), where

(1) `STRUCT` is the identifying literal atom for modules of this type;

(2) $I = I(M) = (r_1 \ldots r_m)$ is a list of literal atoms called the *(global) inputs* of M;

(3) $O = O(M) = (s_1 \ldots s_n)$ is a list of literal atoms called the *(global) outputs* of M;

(4) $S = S(M) = (\mu_1 \ldots \mu_k)$ is a list of (structural or behavioral) modules, called the *submodules* of M;

(5) $LI = LI(M) = (A_1 \ldots A_k)$, where for $j = 1, \ldots, k$, $A_j = (a_{j1} \ldots a_{jm_j})$ is a list of literal atoms called the j^{th} *local inputs* of M, and m_j is the length of $I(\mu_j)$;

(6) $LO = (B_1 \ldots B_k)$, where for $j = 1, \ldots, k$, $B_j = (b_{j1} \ldots b_{jn_j})$ is a list of literal atoms called the j^{th} *local outputs* of M, and n_j is the length of $O(\mu_j)$.

The members of the list $(r_1 \ldots r_m\, b_{11} \ldots b_{1n_1} \ldots b_{k1} \ldots b_{kn_k})$, consisting of the global inputs and all local outputs, are required to be distinct and are called the *signals* of M. There is no such constraint on the global outputs or local inputs, but each local input must be a signal of M, and each global output must be a local output.

Note that the local inputs and outputs of M correspond to its submodules. Thus, intuitively, the submodules of a structure generate signals that are distinct from each other and from the structure's inputs. Each signal may be connected to arbitrarily many submodule inputs. A signal other than a global input may serve as any number of global outputs, but global inputs and outputs are distinct.

Our first example is the structural module `adder2`, composed of nine nand gates and intended as a gate-level "implementation" of the behavioral module `adder1`:

```
(STRUCT (A B C) (L H)
  (nand2 nand2 nand2 nand2 nand2 nand2 nand2 nand2 nand2)
  ((A B) (A T1) (B T1) (T2 T3) (C T4) (T5 T4) (C T5) (T5 T1) (T7 T6))
  ((T1) (T2) (T3) (T4) (T5) (T6) (T7) (H) (L)))
```

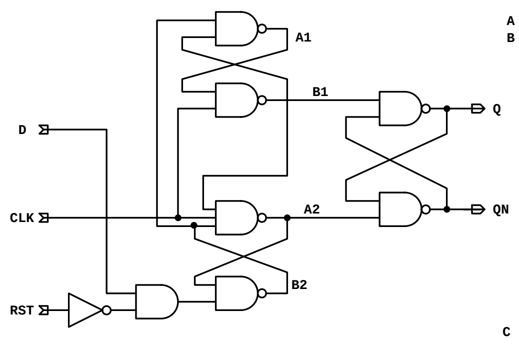
A circuit diagram for `adder2` appears in Fig. 2(b). Later, we shall compare its behavior with that of the behavioral module `adder1`.

An important feature of our language is that it allows cyclic signal paths, thereby providing for the modeling of state-holding devices. Figure 2(a) shows a clocked resettable d-flip-flop, which is modeled by the structural module `dff`:

```
(STRUCT (CLK RST D) (Q QN)
  (not1 and2 nand2 nand2 nand3 nand2 nand2 nand2)
  ((RST) (RN D) (B2 B1) (A1 CLK) (B1 CLK B2) (A2 DD) (B1 QN) (Q A2))
  ((RN) (DD) (A1) (B1) (A2) (B2) (Q) (QN)))
```

In addition to five 2-input nand gates, the submodules of `dff` include an inverter `not1`, an a 2-input and gate `and2`, and a 3-input nand gate `nand3`, the definitions of which are assumed to be understood.

We shall define the semantics of structural modules by means of a function *step*, based on the *exec* function of the preceding subsection. Note that the notions of input and output packets may be naturally applied to any module. For a structural module M, however, instead of a simple output packet, the third argument of *step* must be an object that consists of a waveform corresponding to each signal generated by each component of M. Thus, for any module M, we define a *bundle for M* to be a list B such that (a) if M is behavioral, then B is an output packet for M; (b) if M



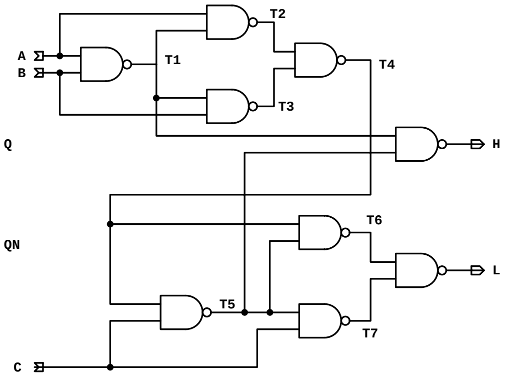


Figure 2: (a) D-Flip-Flop (b) 1-Bit Adder

is a structure with $S(M) = (\mu_1 \ldots \mu_k)$, then $B = (\beta_1 \ldots \beta_k)$, where β_i is a bundle for μ_i, $i = 1, \ldots, k$.

Let B be a bundle for a module M and let s be a signal of M that is not an input of M. The *waveform for s determined by B* is the waveform w that is computed as follows: (a) if M is behavioral and $s = nth(j, O(M))$, then $w = nth(j, B)$; (b) if M is structural and $s = nth(j, nth(i, LO(M)))$, then w is the waveform for $nth(j, O(nth(i, S(M)))$ determined by $nth(i, B)$.

The *output packet for M determined by B*, denoted as $outp(M, B)$, is defined as follows: (a) if M is behavioral, then $outp(M, B) = B$; (b) if M is structural with $O(M) = (s_1 \ldots s_n)$, then $outp(M, B) = (w_1 \ldots w_n)$, where for $1 \leq j \leq n$, w_j is the waveform for s_j determined by B.

Let M be a structural module with $nth(i, LI(M)) = (a_{i1} \ldots a_{in_i})$. Let p be an input packet and let B be a bundle for M. The *i^{th} input packet determined by p and B*, denoted as $inp(i, M, p, B)$, is the input packet $(w_1 \ldots w_m)$ for $nth(i, S(M))$, where for $1 \leq j \leq m$, w_j is computed as follows: (a) if s_j is a global input $nth(k, I(M))$, then $w_j = nth(k, p)$; (b) if s_j is a local output, then w_j is the waveform for s_j determined by B.

We may now define *step*. Let p and B be an input packet and a bundle, respectively, for an arbitrary module M, and let $t \in \mathbf{T}$. Then $step(M, p, B, t)$ is the bundle B', defined as follows: (a) if M is behavioral, then $B' = exec(M, p, B, t)$ if p has a new value at t, and $B' = B$ if not; (b) if M is structural with $S(M) = (\mu_1 \ldots \mu_k)$ and $B = (\beta_1 \ldots \beta_k)$, then $B' = (\beta'_1 \ldots \beta'_k)$, where $\beta'_i = step(\mu_i, inp(i, M, p, B), \beta_i, t)$.

Thus, the execution of a structure at time t amounts to the execution of each behavioral component for which the value of some input signal changes at t.

We have the following generalization of Lemma 2.3:

Lemma 2.6 *Let p_1 and p_2 be input packets and let B be a bundle for a module M. Let $t_0 \in \mathbf{T}$. If $hist(p_1, t_0) = hist(p_2, t_0)$, then $step(M, p_1, B, t_0) = step(M, p_2, B, t_0)$.*

The *history* of a structural bundle $(\beta_1 \ldots \beta_k)$ relative to a time t is recursively defined as $hist(B, t) = (hist(\beta_1, t) \ \ldots \ hist(\beta_k, t))$. Lemma 2.4 may be generalized as follows:

Lemma 2.7 *Let p and B be an input packet and a bundle for a module M. For any $t_0 \in \mathbf{T}$, $hist(step(M, p, B, t_0), t_0) = hist(B, t_0)$.*

2.5 Simulation

Let p and B be an input packet and a bundle for a module M. For any $t \in \mathbf{N}$, we define $t_{next}(t, p, B, M)$ to be the minimum element of the set of all $t' \in \mathbf{N}$ that occur as times of events in the waveforms of p and B and that satisfy $t' > t$, if this set is nonempty; otherwise, $t_{next}(t, p, B, M)$ is undefined.

A simulation of M consists of repeated applications of *step*, which are performed by the function *run*. For $t_0, t_f \in \mathbf{T}$, we define $run(M, p, B, t_0, t_f)$ to be the bundle B' that is computed recursively as follows: Let $t_{next} = t_{next}(t_0, p, B, M)$. If t_{next} is defined and $t_{next} \leq t_f$, then $B' = run(M, p, step(M, p, B, t_{next}), t_{next}, t_f)$; otherwise, $B' = B$.

The definition of our top-level simulation function *sim* depends on *run* as well as a function *init*, which generates an initial bundle from a module and an input packet. First, for a given module M, we define the bundle $B_0(M)$:

(1) If M is behavioral, then $B_0(M)$ is the output packet $(w_0 \ \ldots \ w_0)$ for M, where $w_0 = ((\mathcal{F}, 0))$.

(2) If M is structural and $S(M) = (\mu_1 \ldots \mu_k)$, then $B_0(M) = (B_0(\mu_1) \ldots B_0(\mu_k))$.

Thus, every waveform of $B_0(M)$ is the trivial w_0, which has the constant value $\hat{w}_0(t) = \mathcal{F}$. Prior to simulation, each of these waveforms is updated by executing every behavioral component of M. The result is the bundle $init(M, p)$, defined as follows:

(1) If M is behavioral, then $init(M, p) = exec(M, p, B_0(M), 0)$;

(2) If M is structural with $S(M) = (\mu_1 \ \ldots \ \mu_k)$, then
$init(M, p) = (init(\mu_1, inp(1, M, p, B_0(M))) \ \ldots \ init(\mu_k, inp(k, M, p, B_0(M))))$.

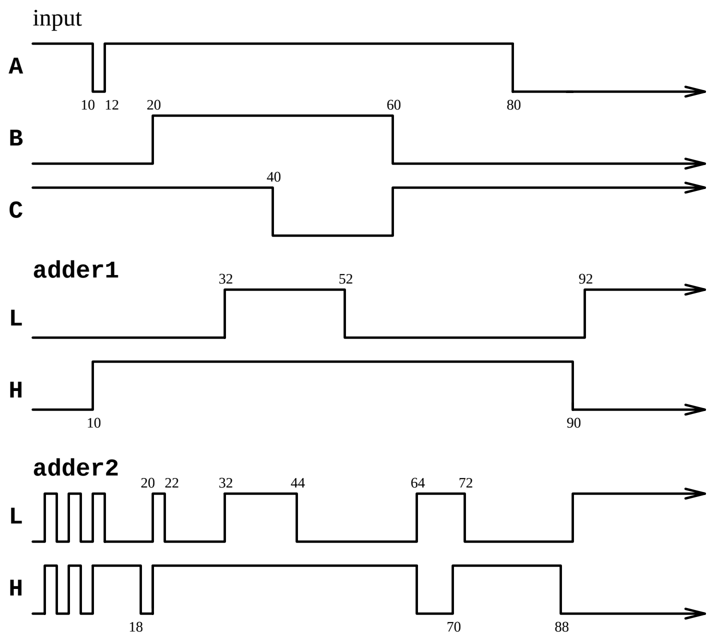


Figure 3: Simulation of `adder1` and `adder2`

Now, given an input packet p for M and a time object t, we define

$$sim(M, p, t) = run(M, p, init(M, p), 0, t).$$

As an example, simulations of the modules `adder1` and `adder2` are illustrated in Fig. 3. The waveforms corresponding to the inputs `A`, `B`, and `C` are

$$w_{\mathrm{A}} = ((\mathcal{F}, 80000)\ (\mathcal{T}, 12000)\ (\mathcal{F}, 10000)\ (\mathcal{T}, 0)),$$

$$w_{\mathrm{B}} = ((\mathcal{F}, 60000)\ (\mathcal{T}, 20000)\ (\mathcal{F}, 0)),$$

and

$$w_{\mathrm{C}} = ((\mathcal{T}, 60000)\ (\mathcal{F}, 40000)\ (\mathcal{T}, 0)),$$

respectively. The resulting output of the behavioral module `adder1` is

$$sim(\texttt{adder1}, (w_{\mathrm{A}}\ w_{\mathrm{B}}\ w_{\mathrm{C}}), 100000) = (w_{\mathrm{L}}\ w_{\mathrm{H}}),$$

where

$$w_{\mathrm{L}} = (((\mathcal{T}, 92000)\ (\mathcal{F}, 52000)\ (\mathcal{T}, 32000)\ (\mathcal{F}, 0)))$$

and

$$w_{\mathrm{H}} = ((\mathcal{F}, 90000)\ (\mathcal{T}, 10000)\ (\mathcal{F}, 0)).$$

This example exhibits a fundamental characteristic of inertial delay, as distinguished from transport delay: an input pulse of duration less than the delay, as occurs in w_{A} over the interval $[10000, 12000)$, is not reflected in the output.

The corresponding simulation of the circuit `adder2` produces a bundle of the form

$$sim(\texttt{adder2}, (w_{\mathrm{A}} w_{\mathrm{B}} w_{\mathrm{C}}), 100000) = ((w'_{T1})(w'_{T2})(w'_{T3})(w'_{T4})(w'_{T5})(w'_{T6})(w'_{T7})(w'_H)(w'_L)),$$

of which we illustrate the output waveforms w'_L and w'_H. Note that the behavior of `adder2` is somewhat more complicated than that of `adder1`, although for stable inputs, the two modules eventually produce the same values.

Some basic properties of the simulator that will be critical in the subsequent development are summarized in the following three lemmas. The first of these provides for the decomposition of a simulation interval:

Lemma 2.8 *If p is an input packet for a module M, and $t_1 \leq t_2$, then $sim(M, p, t_2) = run(M, p, sim(M, p, t_1), t_1, t_2)$.*

An equally important result is the following, which describes the behavior of a structural module in terms of that of its components. Its proof (which appears in [6]) depends on the two properties of *step* that are stated in Lemmas 2.6 and 2.7, namely that module execution is neither predictive (with respect to input) nor retroactive (with respect to output):

Lemma 2.9 *Let p be an input packet for a structural module M with $S(M) = (\mu_1 \ldots \mu_k)$. Let $t \in \mathbf{N}$ and $B = (\beta_1 \ldots \beta_k) = sim(M, p, t)$. Then $\beta_i = sim(\mu_i, b_i, t)$, where $b_i = inp(i, M, p, B)$, $i = 1, \ldots, k$.*

Finally, as a consequence of Lemma 2.5, we observe that at any time during a simulation of a behavioral module, the output packet is the result of executing the module at that time, regardless of whether the execution actually occurs, i.e., whether there is change in input:

Lemma 2.10 *Let p be an input packet for a behavioral module M, let $t \in \mathbf{N}$, and let $B = sim(M, p, t)$. Then $B = exec(M, p, B, t)$.*

3 Specification of Synchronous Circuits

3.1 Combinational Modules

Before undertaking a characterization of synchronous sequential circuits, we shall consider the relatively simple class of *combinational* circuits. Let $\rho = (s_1 \ldots s_p)$ be a list of signals of a structural module M such that for each i, $1 < i \leq p$, there exists j such that s_{i-1} is a member of $nth(j, LI(M))$ and s_i is a member of $nth(j, LO(M))$. Then ρ is a *path* in M from s_1 to s_p. If $s_1 = s_p$, then ρ is a *loop* in M. An arbitrary module M is *combinational* if either (a) M is behavioral or (b) M is structural with no loops and all of its submodules are combinational.

The notion of *combinational value*, which previously applied only to outputs of behavioral modules, may be extended to combinational modules. Let s be any signal of a combinational module M and let V be a bit vector of the same length as $I(M)$.

(1) If $s = nth(j, I(M))$, then $cv(s, V, M) = nth(j, V)$;

(2) If M is structural and $s = nth(j, nth(i, LO(M)))$, where $\mu = nth(i, S(M))$ and $(a_1 \ \ldots \ a_m) = nth(i, LI(M))$, then

$$cv(s, V, M) = cv(nth(j, O(\mu)), (cv(a_1, V, M) \ \ldots \ cv(a_m, V, M)), \mu).$$

We shall describe the behavior of combinational modules in terms of the function cv. Our analysis begins with the following characterization of behavioral modules:

Lemma 3.1 *Let $s = nth(j, O(M))$ be the j^{th} output of a behavioral module M, let $d = nth(j, D(M))$ be the corresponding delay, and let $w = nth(j, sim(M, p, t_f))$.*

Assume that for all $t \in [t_1, t_2)$, the combinational value of s w.r.t. $\hat{p}(t)$ is v, where $t_1 + d \leq t_2$ and $t_1 \leq t_f$. Then for all $t \in [t_1 + d, t_2 + d)$, $\hat{w}(t) = v$.

Proof: Let $p_1 = sim(M, p, t_1)$. Then according to Lemma 2.10,

$$p_1 = exec(M, p, p_1, t_1).$$

It follows from Lemma 2.1(a) that the value of $nth(j, p_1)$ is v for all $t \geq t_1 + d$.

We claim that if p' is any output packet for M such that $nth(j, p')$ has value v throughout $[t_1 + d, t_2 + d)$, then so does $nth(j, run(M, p, p', t', t_f))$, for any $t' \geq t_1$. Once this claim is proved, the lemma will follow from Lemma 2.8 upon substituting p_1 and t_1 for p' and t'.

The claim is proved by induction. It suffices to show that if p has a new value at $t'' = t_{next}(t', p, p', M)$, and $p'' = exec(M, p, p', t'')$, then $nth(j, p'')$ has value v throughout $[t_1 + d, t_2 + d)$.

If $t'' \geq t_2$, then the desired result follows from Lemma 2.1(c). Thus, we may assume $t'' < t_2$ and hence, the combinational value of s w.r.t. $\hat{p}(t'')$ is v. In this case, $nth(j, p'')$ has value v on $[t_1 + d, t'' + d)$ by Lemma 2.1(c), and on $[t'' + d, t_2 + d)$ by Lemma 2.1(a). □

Lemma 3.1 is illustrated by the simulation of `adder1` shown in Fig. 3. Note, for example, that the output `L` of `adder1`, with corresponding term `(XOR3 A B C)`, has the combinational value $\mathcal{F}$ throughout the interval from 40000 to 80000, and thus, since its delay is 12000, the actual value of the signal is $\mathcal{F}$ from 52000 to 92000. Note also that this simple behavior is not shared by the combinational module `adder2`.

However, we shall derive a generalization of Lemma 3.1 that provides similar (although somewhat weaker) behavioral specifications of arbitrary combinational modules. First, we associate each signal s of a combinational module M with two parameters, called the *minimum* and *maximum delays* of s, which represent the range of total delays along all paths connecting the inputs of M to s. These are defined as follows:

(1) If s is a member of $I(M)$, then $dmin(s,M) = dmax(s,M) = 0$;

(2) If M is behavioral and $s = nth(j, O(M))$, then $dmin(s,M) = dmax(s,M) = nth(j, D(M))$;

(3) If M is structural and $s = nth(j, nth(i, LO(M)))$, where $\mu = nth(i, S(M))$ and $(a_1 \ \ldots \ a_m) = nth(i, LI(M))$, then

$$dmin(s,M) = dmin(nth(j,O(\mu)),\mu) + min(dmin(a_1,M),\ldots,dmin(a_m,M)),$$

$$dmax(s,M) = dmax(nth(j,O(\mu)),\mu) + max(dmax(a_1,M),\ldots,dmax(a_m,M)).$$

Lemma 3.2 *Let $s = nth(j, O(M))$ be the j^{th} output of a combinational module M, $d = dmin(s,M)$, $d' = dmax(s,M)$, and $w = nth(j, outp(M, sim(M,p,t_f)))$.*

Assume that $\hat{p}$ is constant on the interval $[t_1, t_2)$, where $t_1 + d' \leq t_2$ and $t_1 \leq t_f$. Let $v = cv(s, \hat{p}(t_1), M)$. Then for all $t \in [t_1 + d', t_2 + d)$, $\hat{w}(t) = v$.

Proof: For behavioral M, the conclusion follows from Lemma 3.1. For structural M, we shall show that it holds more generally for any local output s of M and the waveform w for s determined by $B = sim(M,p,t_f)$. The proof is by induction on the length of the longest path in M terminating at s.

Suppose $s = nth(j, nth(i, LO(M)))$. Let $\mu = nth(i, S(M))$, $\beta = nth(i, B)$, $(a_1 \ \ldots \ a_m) = nth(i, LI(M))$, and $b = inp(i, M, p, B) = (w_1 \ \ldots \ w_m)$. Then $w = nth(j, outp(\mu, \beta))$, and by Lemma 2.9, $\beta = sim(\mu, b, t_f)$.

For $1 \leq \ell \leq m$, let $d_\ell = dmin(a_\ell, M)$, $d'_\ell = dmax(a_\ell, M)$, and $v_\ell = cv(a_\ell, \hat{p}(t_1), M)$. If a_ℓ is a local output of M, then by inductive hypothesis, $\hat{w}_\ell(t) = v_\ell$ for all $t \in [t_1 + d'_\ell, t_2 + d_\ell)$; otherwise, a_ℓ is an input, and the same is true trivially. Thus, $\hat{b}(t) = (v_1 \ \ldots \ v_m)$ for all $t \in [t_1 + \Delta, t_2 + \delta)$, where $\Delta = max(d'_1, \ldots, d'_m)$ and $\delta = min(d_1, \ldots, d_m)$.

By the definition of cv,

$$v = cv(nth(j,O(\mu)),(v_1 \ \ldots \ v_m),\mu) = cv(nth(j,O(\mu)),\hat{b}(t_1+\Delta),\mu).$$

Since μ is combinational,$\hat{w}(t) = v$ for all

$$\begin{aligned} t &\in [t_1 + \Delta + dmax(nth(j, O(\mu)), \mu), t_2 + \delta + dmin(nth(j, O(\mu)), \mu)) \\ &= [t_1 + d', t_2 + d). \ \Box \end{aligned}$$

As an example, consider the output signal `L` of the combinational module `adder2`. By tracing all paths from the inputs to `L`, we may compute $cv(\texttt{L}, (a\ b\ c), \texttt{adder2})$ as a nested $nand2$ expression that may be shown to be tautologically equivalent to $xor3(a, b, c)$. By a similar calculation, we have

$$dmin(\texttt{L}, \texttt{adder2}) = 4000 \text{ and } dmax(\texttt{L}, \texttt{adder2}) = 12000.$$

Thus, according to Lemma 3.2, if $t_1 + 12000 \leq t_2$, $t_1 \leq t_f$, and the input packet p for `adder2` has the constant value $\hat{p}(t) = (a\ b\ c)$ for $t \in [t_1, t_2)$, then

$$w = nth(1, outp(\texttt{adder2}, sim(\texttt{adder2}, p, t_2)))$$

has the value $\hat{w}(t) = xor3(a, b, c)$ for $t \in [t_1 + 12000, t_2 + 4000)$. This result is illustrated in Fig. 3: since the input packet has the constant value $(\mathcal{T}\ \mathcal{T}\ \mathcal{T})$ on the interval $[20000, 40000)$, the value of the first output is $xor3(\mathcal{T}, \mathcal{T}, \mathcal{T}) = \mathcal{T}$ on the interval $[32000, 44000)$.

3.2 Sequential Modules

We shall describe a class of sequential circuits that may be characterized as *synchronous resettable rising-edge-triggered* devices. The flip-flop `dff` of Subsection 2.4 will be used as a primitive in the construction of these circuits.

Let M be a structural module with $I(M) = (r_1\ \ldots\ r_m)$, where $m \geq 2$, $S(M) = (\mu_1 \ldots \mu_k)$, and for $i = 1, \ldots, k$, $nth(i, LI(M)) = (a_{i1} \ldots a_{im_i})$ and $nth(i, LO(M)) = (b_{i1}\ \ldots\ b_{in_i})$. Let $q \in \mathbf{N}$. Then M is a *sequential module* with multiplicity $q = mult(M)$ if either (a) $q = 0$ and $M = \texttt{dff}$, or (b) $0 < q \leq k$ and the following conditions hold:

(1) For $1 \leq i \leq q$, μ_i is a sequential module;

(2) For $q < i \leq k$, μ_i is a combinational module;

(3) For $1 \leq i \leq k$ and $1 \leq j \leq m_i$, $a_{ij} = r_1$ iff $i \leq q$ and $j = 1$;

(4) For $1 \leq i \leq k$ and $1 \leq j \leq m_i$, $a_{ij} = r_2$ iff $i \leq q$ and $j = 2$;

(5) If $(s_1 \ldots s_p)$ is a path in M with $s_1 = s_p$, then for some i and j, where $1 \leq i \leq p$ and $1 \leq j \leq q$, s_i is a member of $nth(j, LO(M))$;

(6) If $(s_1 \ldots s_p)$ is a path in M with s_1 a global input and s_p a global output of M, then for some i and j, where $1 \leq i \leq p$ and $1 \leq j \leq q$, s_i is a member of $nth(j, LO(M))$.

Throughout the remainder of this section, we shall assume that M is a sequential module with $I(M)$, $S(M)$, $LI(M)$, and $LO(M)$ as denoted above. Note that M must have at least two inputs, r_1 and r_2, which we call the *clock* and *reset*, respectively; the other inputs are called *data*. According to (3) and (4), if $M \neq$ `dff`, then the clock and reset of M are connected to the clock and reset, respectively, of each sequential submodule of M, and to no other submodule inputs.

We define a path in M to be *combinational* if it contains no signal that is a local output of a sequential submodule. According to (5) of the definition, M contains no combinational loop; according to (6), no combinational path connects an input to an output.

We define a signal s of M to be *native* if there is no combinational path from any global input to s; the signals `Q` and `QN` of `dff` are also defined to be *native.* Thus, all outputs of M are native signals.

A native signal s of M is *registered* if either (a) $M =$ `dff` and s is an output of M, or (b) $M \neq$ `dff` and s is a local output b_{ij} where $i \leq q$ and $nth(j, O(\mu_i))$ is a registered signal of μ_i. This property will have special significance in connection with asynchronous communication.

Two examples of sequential modules are diagrammed in Fig. 4. The enabled d-flip-flop, `edff`, is defined to be the following structure:

```
(STRUCT
  (CLK RST EN D)
  (Q QN)
  (dff not1 nand2 nand2 nand2)
  ((CLK RST S4) (EN) (S1 Q) (D EN) (S2 S3))
  ((Q QN) (S1) (S2) (S3) (S4)))
```

Clearly, this module satisfies the definition, with $mult($`edff`$) = 1$.

The 3-bit counter `count3` is a sequential module of multiplicity 3, defined as follows:

```
(STRUCT
  (CLK RST EN)
  (Q0 Q1 Q2)
(edff edff edff and2 xor2 xor2)
((CLK RST EN QN0) (CLK RST EN S3) (CLK RST EN S2)
 (Q0 Q1) (S1 Q2) (Q0 Q1))
((Q0 QN0) (Q1 QN1) (Q2 QN2) (S1) (S2) (S3)))
```

Note that all outputs of both of these modules are registered.

3.3 Sequential Values

Our description of the behavior of sequential modules will be based on a function that computes a sequence of values for each output corresponding to a given sequence of input values. The definition of this function the notion of *state.* An object Σ is a *state* of M if

(1) $M = \texttt{dff}$ and $\Sigma \in \mathbf{B}$,

(2) $mult(M) = 1$ and Σ is a state of μ_1, or

(3) $mult(M) = q > 1$ and $\Sigma = (\sigma_1 \ \ldots \ \sigma_q)$, where for $i = 1, \ldots, q$, σ_i is a state of μ_i.

Thus, a state associates a Boolean value with each flip-flop. The *reset state* $\Sigma_0(M)$ is the state for which each of these values is $\mathcal{F}$:

(1) $\Sigma_0(\texttt{dff}) = \mathcal{F}$;

(2) If $mult(M) = 1$, then $\Sigma_0(M) = \Sigma_0(\mu_1)$;

(3) If $mult(M) = q > 1$, then $\Sigma_0(M) = (\Sigma_0(\mu_1) \ \ldots \ \Sigma_0(\mu_q))$.

A *data vector* for M is a bit vector of length $m - 2$, the components of which correspond to the data inputs of M. We shall define a function $next(V, \Sigma, M)$ that computes a state of M from a data vector V and a state Σ. This definition requires two auxiliary functions.

First, for a native signal s and a state Σ of M, we define the *native value* of s determined by Σ, denoted as $nv(s, \Sigma, M)$, as follows:

(1) $nv(\texttt{Q}, \Sigma, \texttt{dff}) = \Sigma$ and $nv(\texttt{QN}, \Sigma, \texttt{dff}) = not1(\Sigma)$;

(2) If $mult(M) = 1$ and $s = b_{1j}$, then $nv(s, \Sigma, M) = nv(nth(j, O(\mu_1)), \Sigma, \mu_1)$;

(3) If $mult(M) = q > 1$ and $s = b_{ij}$, where $i \leq q$, then

$$nv(s, \Sigma, M) = nv(nth(j, O(\mu_i)), nth(i, \Sigma), \mu_i);$$

(4) If $mult(M) = q \geq 1$ and $s = b_{ij}$, where $i > q$, then

$$nv(s, \Sigma, M) = cv(nth(j, O(\mu_i)), (nv(a_{i1}, \Sigma, M) \ \ldots \ nv(a_{im_i}, \Sigma, M)), \mu_i).$$

Now, let $V = (v_3 \ \ldots \ v_m)$ and Σ be a data vector and a state of M, respectively. We define the *resultant value* of a signal s determined by V and Σ, denoted as $rv(s, V, \Sigma, M)$, as follows:

(1) If $s = r_i$ is a data input of M, then $rv(s, V, \Sigma, M) = v_i$;

(2) If s is native to M, then $rv(s, V, \Sigma, M) = nv(s, \Sigma, M)$;

(3) If $mult(M) = q > 0$ and $s = b_{ij}$, where $i > q$, then

$$rv(s, V, \Sigma, M) = cv(nth(j, O(\mu_i)), (rv(a_{i1}, V, \Sigma, M) \ \ldots \ rv(a_{im_i}, V, \Sigma, M)), \mu_i).$$

We may now define the function *next*. Let $mult(M) = q$ and for $i = 1, \ldots, q$, let $L_i = (rv(a_{i1}, V, \Sigma, M) \ \ldots \ rv(a_{im_i}, V, \Sigma, M))$. Then $next(V, \Sigma, M) = \Sigma'$, where

(1) If $q = 0$ (i.e., $M =$ `dff`), then $\Sigma' = v_3$;

(2) If $q = 1$, then $\Sigma' = next(L_1, \Sigma, \mu_1)$;

(3) If $q > 1$ and $\Sigma = (\sigma_1 \ \ldots \ \sigma_q)$, then $\Sigma' = (next(L_1, \sigma_1, \mu_1) \ \ldots \ next(L_q, \sigma_q, \mu_q))$.

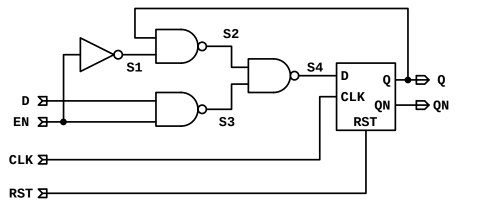
Now, let $\mathcal{V} = (V_3 \ldots V_m)$, where for $i = 3, \ldots, m$, $V_i = (v_{i1} \ldots v_{in})$ is a bit vector of length n. $\mathcal{V}$ may be viewed as a Boolean matrix, the rows of which correspond to the data inputs of M. Each column of this matrix, $\bar{V}_j = (v_{3j} \ \ldots \ v_{mj})$, where $j = 1, \ldots, n$, is a data vector for M. A sequence of $n + 1$ states is determined by $\mathcal{V}$ as follows:

$$state(j, \mathcal{V}, M) = \begin{cases} \Sigma_0(M) & \text{if } j = 0 \\ next(\bar{V}_j, , state(j-1, \mathcal{V}, M), M) & \text{if } 0 < j \leq n. \end{cases}$$

For any native signal s of M, the j^{th} *sequential value* of s determined by $\mathcal{V}$ is defined as

$$sv(j, s, \mathcal{V}, M) = nv(s, state(j, \mathcal{V}, M), M).$$

Thus, the sequential values corresponding to a given matrix of input values are determined by the functions *nv* and *next*. As an illustration, we shall analyze the behavior of these functions for the modules `edff` and `count3`. Clearly, a state of



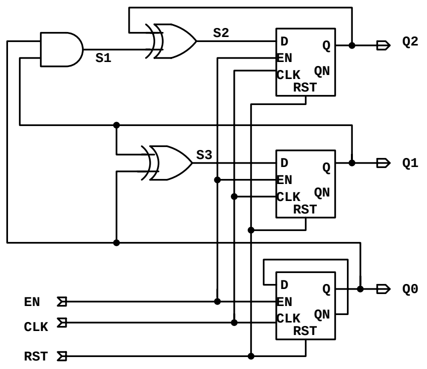


Figure 4: (a) `edff` (b) `count3`

`edff` is a state of `dff`, i.e., a Boolean value. If Σ is such a state and $V = (v_3\ v_4)$ is a data vector, then

$$rv(\mathtt{Q}, V, \Sigma, \mathtt{edff}) = nv(\mathtt{Q}, V, \Sigma, \mathtt{edff}) = nv(\mathtt{Q}, \Sigma, \mathtt{dff}) = \Sigma$$

and

$$rv(\mathtt{QN}, V, \Sigma, \mathtt{edff}) = nv(\mathtt{QN}, V, \Sigma, \mathtt{edff}) = nv(\mathtt{QN}, \Sigma, \mathtt{dff}) = not1(\Sigma).$$

Expanding the definition of rv, we have

$$rv(\mathtt{S4}, V, \Sigma, \mathtt{edff}) = nand2(nand2(not1(v_3), \Sigma), nand2(v_3, v_4)),$$

which is also the value of $next(V, \Sigma, \mathtt{edff})$. A trivial calculation yields the following:

Proposition 3.1 *Let Σ and $V = (v_3\ v_4)$ be a state and a data vector for* `edff`. *Then*

$$nv(\mathtt{Q}, V, \Sigma, \mathtt{edff}) = \Sigma \text{ and } nv(\mathtt{QN}, V, \Sigma, \mathtt{edff}) = not1(\Sigma);$$

$$next(V, \Sigma, \mathtt{edff}) = \begin{cases} v_4 & \text{if } v_3 = \mathcal{T} \\ \Sigma & \text{if } v_3 = \mathcal{F}. \end{cases}$$

A state of `count3` is a vector of 3 Boolean values, corresponding to the

$$mult(\mathtt{count3}) = 3$$

occurrences of `edff`. If $\Sigma = (\sigma_0\ \sigma_1\ \sigma_2)$ and $V = (v_3)$ are a state and a data vector, then

$$\begin{aligned} rv(\texttt{S1}, V, \Sigma, \texttt{count3}) &= and2(\sigma_0, \sigma_1), \\ rv(\texttt{S2}, V, \Sigma, \texttt{count3}) &= xor2(and2(\sigma_0, \sigma_1), \sigma_2), \\ rv(\texttt{S3}, V, \Sigma, \texttt{count3}) &= xor2(\sigma_0, \sigma_1), \end{aligned}$$

and it follows from Proposition 3.1 that

$$next(V, \Sigma, \texttt{count3}) = \begin{cases} (not1(\sigma_0)\ \ xor2(\sigma_0, \sigma_1)\ \ xor2(and2(\sigma_0, \sigma_1), \sigma_2) \text{ if } v_3 = \mathcal{T} \\ \Sigma \text{ if } v_3 = \mathcal{F}. \end{cases}$$

This result is conveniently expressed in terms of the function $inc(W)$, defined as follows for an arbitrary bit vector W:

(1) If $W = \texttt{NIL}$, then $inc(W) = \texttt{NIL}$; otherwise:

(2) If $car(W) = \mathcal{T}$, then $inc(W) = cons(\mathcal{F}, inc(cdr(W)))$; otherwise:

(3) $inc(W) = cons(\mathcal{T}, cdr(W))$.

Proposition 3.2 *Let $\Sigma = (\sigma_0\ \sigma_1\ \sigma_2)$ and $V = (v_3)$ be a state and a data vector for* `count3`. *Then*

$$nv(\texttt{Q0}, V, \Sigma, \texttt{count3}) = \sigma_0,\ nv(\texttt{Q1}, V, \Sigma, \texttt{count3}) = \sigma_1,\ nv(\texttt{Q2}, V, \Sigma, \texttt{count3}) = \sigma_2;$$

$$next(V, \Sigma, \texttt{count3}) = \begin{cases} inc(\Sigma) & \textit{if } v_3 = \mathcal{T} \\ \Sigma & \textit{if } v_3 = \mathcal{F}. \end{cases}$$

3.4 Behavior of `dff`

Naturally, the behavior of sequential modules depends on that of the primitive `dff`. A precise behavioral specification of `dff` is given by the following lemma, the proof of which is an elaboration of the informal argument found in [7]:

Lemma 3.3 *Let $t_1 + 4000 \le t_-$, $t_- + 6000 \le t_2$, and $t_1 \le t_f$. Let $p = (w_{\text{CLK}}\, w_{\text{RST}}\, w_{\text{D}})$ be an input packet for* `dff`*, and suppose that*

$$\hat{w}_{\text{CLK}}(t) = \begin{cases} \mathcal{F} & \textit{for all } t \in [t_1 - 6000, t_1) \cup [t_-, t_2) \\ \mathcal{T} & \textit{for all } t \in [t_1, t_-), \end{cases}$$

$$\hat{w}_{\text{RST}}(t) = r \textit{ for all } t \in [t_1 - 8000, t_1),$$

and

$$\hat{w}_{\mathrm{D}}(t) = d \textit{ for all } t \in [t_1 - 6000, t_1).$$

Let $sim(\texttt{dff}, p, t_f) = ((w_{\mathrm{RN}})\ (w_{\mathrm{DD}})\ (w_{\mathrm{A1}})\ (w_{\mathrm{B1}})\ (w_{\mathrm{A2}})\ (w_{\mathrm{B2}})\ (w_{\mathrm{Q}})\ (w_{\mathrm{QN}}))$ *and let* $v = and2(not1(r), d)$. *Then* $\hat{w}_{\mathrm{Q}}(t) = v$ *and* $\hat{w}_{\mathrm{Q}}(t) = not1(v)$ *for all* $t \in [t_1 + 6000, t_2 + 4000)$. *Moreover, if these same values hold for all* $t \in [t_1, t_1 + 4000)$, *then they also hold for all* $t \in [t_1 + 4000, t_1 + 6000)$.

Proof: By Lemmas 3.1 and 2.9, we have $\hat{w}_{\mathrm{RN}}(t) = not1(r)$ for all $t \in [t_1 - 6000, t_1 + 2000)$. Applying the same two lemmas again, we have $\hat{w}_{\mathrm{DD}}(t) = v$ for all $t \in [t_1 - 4000, t_1 + 2000)$. Similarly, $\hat{w}_{\mathrm{A2}}(t) = \hat{w}_{\mathrm{B1}}(t) = \mathcal{T}$ for $t \in [t_1 - 4000, t_1 + 2000)$, $\hat{w}_{\mathrm{B2}}(t) = not1(v)$ for $t \in [t_1 - 2000, t_1 + 4000)$, and hence $\hat{w}_{\mathrm{A1}}(t) = v$ for $t \in [t_1, t_1 + 4000)$.

We shall consider the case $v = \mathcal{F}$; the case $v = \texttt{T}$ is similar. In this case, $\hat{w}_{\mathrm{B1}}(t) = \mathcal{T}$ for $t \in [t_1 + 2000, t_1 + 6000)$, and hence $\hat{w}_{\mathrm{A2}}(t) = \mathcal{F}$ for $t \in [t_1 + 2000, t_1 + 6000)$.

Let t' be the least time such that $t' > t_1 + 2000$ and some waveform in the set $\{w_{\mathrm{A1}}, w_{\mathrm{B1}}, w_{\mathrm{A2}}, w_{\mathrm{B2}}\}$ assumes a new value at t'. Then $\hat{w}_{\mathrm{A1}}(t) = \hat{w}_{\mathrm{A2}}(t) = \mathcal{F}$ and $\hat{w}_{\mathrm{B1}}(t) = \hat{w}_{\mathrm{B2}}(t) = \mathcal{T}$ for $t \in [t_1 + 2000, t')$. Since $t' \geq t_1 + 4000$, it follows that $\hat{w}_{\mathrm{B1}}(t) = \hat{w}_{\mathrm{B2}}(t) = \mathcal{T}$ and $\hat{w}_{\mathrm{A1}}(t) = \mathcal{F}$ for $t \in [t_1 + 4000, t' + 2000)$. Similarly, $\hat{w}_{\mathrm{A2}}(t) = \mathcal{F}$ for $t \in [t_1 + 4000, min(t' + 4000, t_- + 2000))$. Thus, only w_{A2} can possibly assume a new value at t', and this requires that $t' \geq t_- + 2000$.

Hence, $\hat{w}_{\mathrm{B1}}(t) = \mathcal{T}$ and $\hat{w}_{\mathrm{A2}}(t) = \texttt{F}$ for $t \in [t_1 + 2000, t_- + 2000)$. It follows that $\hat{w}_{\mathrm{QN}}(t) = \mathcal{T}$ for $t \in [t_1 + 4000, t_- + 4000)$, and hence $\hat{w}_{\mathrm{Q}}(t) = \mathcal{F}$ for $t \in [t_1 + 6000, t_- + 4000)$.

Let t'' be the least time such that $t'' > t_1 + 6000$ and either w_{Q} or w_{QN} assumes a new value at t''. By an argument similar to the above, it is easily shown that $t'' \geq t_2 + 4000$. Thus, $\hat{w}_{\mathrm{Q}}(t) = \mathcal{F} = u_r$ for $t \in [t_1 + 6000, t_2 + 4000)$, and $\hat{w}_{\mathrm{QN}}(t) = \mathcal{T} = not1(u_r)$ for $t \in [t_1 + 4000, t_2 + 4000)$.

Now suppose that $\hat{w}_{\mathrm{Q}}(t) = \mathcal{F}$ and $\hat{w}_{\mathrm{QN}}(t) = \mathcal{T}$ for $t \in [t_1, t_1 + 4000)$. Then $\hat{w}_{\mathrm{QN}}(t) = \mathcal{T}$ for $t \in [t'_1, t_2 + 4000)$. It follows that $\hat{w}_{\mathrm{Q}}(t) = \mathcal{F}$ for $t \in [t_1 + 4000, t_2 + 4000)$. □

3.5 Parameters

Our objective is to impose constraints on the input to a sequential module that will allow its outputs to be described in terms of sequential values. In particular, the clock input will be required to exhibit periodic behavior. We shall call each event of its associated waveform a *rising* or *falling edge*, according to whether its value is $\mathcal{T}$ or $\mathcal{F}$. An interval between two successive rising edges is called a *cycle*. Each of the remaining inputs will be required to maintain a stable value over a prescribed interval preceding each rising edge. For the reset input r_2, this value is $\mathcal{T}$ for an initial cycle, and $\mathcal{F}$ for every cycle thereafter.

Under these constraints, we shall show that the behavior of M admits a fairly simple description. A state of M will be associated with each rising edge. This state may computed from the data values prior to the edge and the previous state by the function *next*. The values of the outputs, which may change only during a short interval following a rising edge, are the corresponding sequential values.

We shall describe the behavior of the signals of M in terms of several parameters. First, we associate with each input other than the clock a *setup time*, which represents the duration over which the signal is required to hold constant prior to a rising edge. For the case $M = \texttt{dff}$, as suggested by Lemma 3.3, we define

$$setup(\texttt{RST}, \texttt{dff}) = 8000 \text{ and } setup(\texttt{D}, \texttt{dff}) = 6000.$$

Now suppose $mult(M) = q > 0$ and let s be any signal of M other than r_1. Assume $setup(s', M)$ has been defined for each $s' \neq s$ that lies on a combinational path starting at s. For $i = 1, \ldots, k$, let ζ_i be defined as follows:

(1) If $s \neq a_{ij}$ for all j, $1 \leq j \leq m_i$, then $\zeta_i = 0$; otherwise:

(2) If $i \leq q$, then ζ_i is the maximum $setup(nth(j, I(\mu_i)), \mu_i)$ such that $s = a_{ij}$, $j = 2, \ldots, m_i$; otherwise:

(3) $i > q$, and ζ_i is the maximum sum $dmax(nth(j, O(\mu_i)), \mu_i) + setup(b_{ij}, M)$ such that $setup(b_{ij}, M) > 0$, $j = 1, \ldots, n_i$.

Then $setup(s, M) = max(\zeta_1, \ldots, \zeta_k)$.

Each native signal of M is associated with a *minimum* and a *maximum delay*, which determine an interval during which the signal's value may change following a rising edge. For the case $M = \texttt{dff}$, we define

$$dmin(\texttt{Q}, \texttt{dff}) = dmin(\texttt{QN}, \texttt{dff}) = 4000,$$
$$dmax(\texttt{Q}, \texttt{dff}) = dmax(\texttt{QN}, \texttt{dff}) = 6000.$$

Now suppose $mult(M) = q > 0$ and let $s = b_{ij}$ be any native signal of M.

(1) If $i \leq q$, then

$$dmin(s, M) = dmin(nth(j, O(\mu_i)), \mu_i),$$
$$dmax(s, M) = dmax(nth(j, O(\mu_i)), \mu_i);$$

(2) If $i > q$, then

$$dmin(s, M) = dmin(nth(j, O(\mu_i)), \mu_i) + min(dmin(a_{i1}, M), \ldots, dmin(a_{im_i}, M)),$$
$$dmax(s, M) = dmax(nth(j, O(\mu_i)), \mu_i) + max(dmax(a_{i1}, M), \ldots, dmax(a_{im_i}, M)).$$

We also define three parameters pertaining to the behavior of the clock input of M, called the *clock high*, the *clock low*, and the *minimum period* of M. These represent the minimum durations between a rising edge and the next falling edge, a falling edge and the next rising edge, and successive rising edges, respectively. First, we define $high(\texttt{dff}) = 4000$, $low(\texttt{dff}) = 6000$, and $per(\texttt{dff}) = 10000$. For $mult(M) = q > 0$, we define

$high(M) = max(high(\mu_1), \ldots, high(\mu_q))$;
$low(M) = max(low(\mu_1), \ldots, low(\mu_q))$;
$per(M) = max(P_1, P_2, P_3)$, where
$P_1 = max\{per(\mu_i) : 1 \leq i \leq q\}$;
$P_2 = max\{setup(r_i, M) : 2 \leq i \leq m\}$;
$P_3 = max\{setup(b_{ij}, M) + dmax(nth(j, O(\mu_i)), \mu_i) : 1 \leq i \leq q, 1 \leq j \leq n_i\}$.

Consider, for example, the circuits `edff` and `count3`. First, the setup times for the signals of `edff` may be computed directly from the definitions, by tracing along all combinational paths. For example,

$setup(\texttt{RST}, \texttt{edff}) = 8000$,
$setup(\texttt{EN}, \texttt{edff}) = 12000$,
$setup(\texttt{D}, \texttt{edff}) = 10000$;

The setups for `count3` follow trivially:

$setup(\texttt{RST}, \texttt{count3}) = 8000$,
$setup(\texttt{EN}, \texttt{count3}) = 12000$.

In fact, it follows from our definitions that the reset input of every sequential module is 8000.

All outputs of both of these devices are registered. It follows that the minimum and maximum delay of each output are 4000 and 6000, respectively.

Similarly, the clock high and low of each device (in fact, of any sequential device) are 4000 and 6000, respectively, as determined by `dff`. Calculation of the minimum period, on the other hand, involves a comparison of various setups and delays. In the case of `edff`, the minimum period is found to be

$$setup(\texttt{Q}, \texttt{edff}) + dmax(\texttt{Q}, \texttt{dff}) = 10000 + 6000 = 16000;$$

for `count3`, it is

$$setup(\texttt{Q0}, \texttt{count3}) + dmax(\texttt{Q}, \texttt{edff}) = 14000 + 6000 = 20000.$$

3.6 The Main Theorem

The input constraints for sequential modules will be expressed in terms of the functions *setup*, *high*, *low*, and *per*. First, we define a waveform w to be an *n-cycle pulse based at t_0 with high h, low ℓ, and period $\pi = h + \ell$* if for $k = 0, \ldots, n-1$,

$$\hat{w}(t) = \begin{cases} \mathcal{T} & \text{for all } t \in [t_0 + k\pi, t_0 + k\pi + h) \\ \mathcal{F} & \text{for all } t \in [t_0 + k\pi + h, t_0 + (k+1)\pi). \end{cases}$$

If $h \geq high(M)$, $\ell \geq low(M)$, and $\pi \geq per(M)$, then w is an *admissible pulse for M*.

Let $V = (v_1 \ldots v_n)$ be a bit vector and let $\pi \geq u > 0$. Let w be a waveform such that for $k = 1, \ldots, n$, $\hat{w}(t) = v_k$ for all $t \in [t_0 + k\pi - u, t_0 + k\pi)$. Then w is a *stable n-cycle waveform based at t_0 with setup u, value list V, and period π*. If $u = setup(r_2, M)$, $v_1 = \mathcal{T}$, and $v_2 = \ldots = v_r = \mathcal{F}$, then w is an admissible *reset waveform for M*.

For $i = 1, \ldots, k$, let w_i be a stable n-cycle waveform based at t_0 with value list V_i, setup u_i, and period π. Let $\mathcal{V} = (V_1 \ldots V_k)$, $U = (u_1 \ldots u_k)$, and $W = (w_1 \ldots w_k)$. Then W is a *stable n-cycle packet based at t_0 with value matrix $\mathcal{V}$, setup list U, and period π*. If $k = m - 2$ and $u_i = setup(r_{i+2}, M)$ for $i = 1, \ldots, k$, then W is an *admissible data packet for M*.

Let w_1 be an admissible $(n+2)$-cycle pulse for M based at t_0 with period π. Let w_2 be an admissible $(n+1)$-cycle reset waveform for M based at t_0 with period π. Let $w_3 \ldots w_m$) be an admissible n-cycle data packet for M based at $t_0 + \pi$ with value matrix $\mathcal{V}$ and period π. Then $(w_1 \ldots w_m)$ is an *admissible n-cycle input packet for M based at t_0 with value matrix $\mathcal{V}$ and period π*.

We may now state a behavioral specification for sequential modules:

Theorem 3.1 *Let $s = nth(j, O(M))$ be the j^{th} output of a sequential module M, $d' = dmax(s, M)$, and $w = nth(j, outp(M, sim(M, p, t_f)))$.*

Assume that p is an admissible n-cycle input packet for M based at t_0 with value matrix $\mathcal{V}$ and period π, where $t_f \geq t_0 + (n+1)\pi$. For $i = 0, \ldots, n$, let $v_i = sv(i, s, \mathcal{V}, M)$. Then w is a stable $(n+1)$-cycle waveform based at $t_0 + \pi$ with setup $\pi - d'$, value list $(v_0 \ldots v_n)$, and period π;

Assume further that s is a registered signal of M and $v_{i-1} = v_i$, for some i, $1 \leq i \leq n$. Then $\hat{w}(t) = v_i$ for all $t \in [t_0 + (i+1)\pi, t_0 + (i+2)\pi)$.

Theorem 3.1 is an immediate consequence of the following:

Lemma 3.4 *Let $s = nth(j, O(M))$ be the j^{th} output of a sequential module M, $d = dmin(s, M)$, $d' = dmax(s, M)$, and $w = nth(j, outp(M, sim(M, p, t_f)))$.*

Assume that p is an admissible n-cycle input packet for M based at t_0 with value matrix $\mathcal{V}$ and period π. Let $t_0 + (n+1)\pi = t_1$, $t_1 + \pi = t_2$, and assume $t_1 \leq t_f$. Let $v = sv(n, s, \mathcal{V}, M)$. Then $\hat{w}(t) = v$ for all $t \in [t_1 + d', t_2 + d)$.

Suppose further that s is a registered signal of M. If $n > 0$ and $sv(n-1, s, \mathcal{V}, M) = v$, then $\hat{w}(t) = v$ for all $t \in [t_1 + d, t_2 + d)$.

Proof: For the case $M = \texttt{dff}$, the lemma is simply a restatement of Lemma 3.3. Thus, we may assume that $M \neq \texttt{dff}$ and proceed by induction on the structure of M. Let $\mathcal{V} = (V_3 \, \ldots \, V_m)$, where for $i = 3, \ldots, m$, $V_i = (v_{i1} \, \ldots \, v_{ir})$. For $j = 0, \ldots, n$, let $\Sigma_j = state(j, \mathcal{V}, M)$.

Let $B = sim(M, p, t_f)$, and for each signal s of M, let

$$w_s = \begin{cases} nth(i,p) & \text{if } s \text{ is a global input } r_i \\ \text{the waveform for } s \text{ determined by } B & \text{if } s \text{ is a local output } b_{ij}, \end{cases}$$

If s is not r_1 or r_2, then for $0 \leq \ell < n$, let

$$val(s, \ell) = rv(s, (v_{3(\ell+1)} \, \ldots \, v_{m(\ell+1)}), \Sigma_\ell, M).$$

If s is native, then by definition we have

$$val(s, \ell) = nv(s, \Sigma_\ell, M) = sv(\ell, s, \mathcal{V}, M).$$

Thus, for native s, we extend the definition to $\ell = n$ by

$$val(s, n) = sv(n, s, \mathcal{V}, M).$$

For any $\ell \in \mathbf{N}$, let $t^\ell = t_0 + (\ell+1)\pi$, so that $t_1 = t^n$ and $t_2 = t^{n+1} = t^n + \pi$. We shall prove, by induction on ℓ, that the following three statements hold for each $\ell \leq n$:

(a) For each i, $1 \leq i \leq q$, $inp(i, M, p, B)$ is an admissible ℓ-cycle input packet for μ_i based at t_0 with value matrix

$$((val(a_{i3}, 0) \, \ldots \, val(a_{i3}, \ell - 1)) \, \ldots \, (val(a_{im_i}, 0) \, \ldots \, val(a_{im_i}, \ell - 1)))$$

and period π.

(b) For each native signal $s = b_{ij}$ of M,

$$\hat{w}_s(t) = val(s, \ell) \text{ for all } t \in [t^\ell + dmax(s, M), t^{\ell+1} + dmin(s, M));$$

if s is a registered signal of M, then the same is true for the interval

$$[t^\ell + dmin(s, M), t^{\ell+1} + dmin(s, M));$$

(c) If $\ell < n$, then for each signal s of M other than r_1 and r_2,

$$\hat{w}_s(t) = val(s, \ell) \text{ for all } t \in [t^{\ell+1} - setup(s, M), t^{\ell+1}).$$

The lemma will then follow from (b), taking $\ell = n$.

Proof of (a): For $\ell = 0$, this follows from (3) and (4) in the definition of *sequential module*. For $\ell > 0$, we must also invoke the inductive hypothesis that (c) holds with ℓ replaced by $\ell - 1$.

Proof of (b): We induct on the length of the longest combinational path terminating at s. Let $s = b_{ij}$. In the base case, where $i \leq q$, the result follows from the inductive assumption that the lemma holds for the sequential submodule μ_i, Lemma 2.9, and (a). In the inductive case, where $i > q$, it follows from Lemmas 2.9 and 3.2.

Proof of (c): This is similarly proved by induction on the length of the longest combinational path terminating at s. In the base case, s is either a global input r_i, $i \geq 3$, or a local output b_{ij}, $i \leq q$. If $s = r_i$, then the claim follows directly from the admissibility of the input packet p. Suppose $s = b_{ij}$, $i \leq q$. It follows from (b) that

$$\hat{w}_s(t) = val(s, \ell) \text{ for all } t \in [t^{\ell} + dmax(s, M), t^{\ell+1}).$$

According to the definition of $per(M)$,

$$\pi \geq setup(b_{ij},, M) + dmax(nth(j, O(\mu_i)), \mu_i).$$

Hence,

$$t^{\ell} + dmax(s, M) = t^{\ell+1} - \pi + dmax(nth(j, O(\mu_i)), \mu_i) \leq t^{\ell+1} - setup(b_{ij}, M).$$

The induction is completed as in the proof of (b). $\Box$

4 Asynchronous Communication

Suppose we have a circuit in which an output of one sequential module, called the *sender*, is connected to a data input of another, called the *receiver*. Under suitable conditions on the sender's input, its output waveform is guaranteed by Theorem 3.1 to be stable with respect to the period of the sender's clock. On the other hand, in order to apply the results of Section 3 to the behavior of the receiver, we must be able to assume that its input is stable with respect to the period of its own clock. In general, this is true only for a synchronous circuit, in which the two modules are driven by the same clock. In this section, we shall examine the asynchronous case, in which the two clock inputs have different periods.

Our treatment of this problem is based on Moore's model of asynchrony [4]. In this model, the behavior of a signal is characterized abstractly by three quantities: a base time, a period, and a bit vector (representing the values assumed on successive cycles). Moore postulates that the receiver's input vector is determined by a function *asynch*, the arguments of which include the sender's output vector, the two periods, and the two base times. In this section, we shall present Moore's function *asynch* and establish the applicability of his model to certain circuits represented in our language. In Section 5, we shall employ a theorem of Moore to show that if the sender's and receiver's periods are known to be approximately equal, then communication may be achieved by means of a well known protocol.

4.1 Smooth and Quasi-Smooth Waveforms

The communication protocol is motivated by the observation that if the time at which the receiver samples its input may be approximated by the sender, then the sender may successfully communicate a value by redundantly writing the value on sufficiently many successive cycles to guarantee that it is the value read by the receiver. For this purpose, the assumption that the sender's output waveform is stable is too weak; the waveform must be known to be constant on each cycle during some critical interval. With this requirement in mind, we define a stable waveform to be *smooth* if its setup time coincides with its period. Thus, w is a smooth n-cycle waveform based at t_0 with value list $V = (v_1 \ldots v_n)$ and period π if for $i = 1, \ldots, n$, $\hat{w}(t) = v_i$ for all $t \in [t_0 + (k-1)\pi, t_0 + k\pi)$.

A somewhat weaker notion of smoothness is needed to describe waveforms that are constant over some but not all cycles. First, we define a list $V = (v_1 \ldots v_n)$ to be a *generalized bit vector* if each v_i is either Boolean or the literal atom `Q`. In this case, we shall call w a *quasi-smooth n-cycle waveform based at t_0 with value list V and period* π if for $i = 1, \ldots, n$, either $v_i =$ `Q` or $\hat{w}(t) = v_i$ for all $t \in [t_0+(k-1)\pi, t_0+k\pi)$. (Thus, the value `Q` corresponds to cycles of unknown behavior.)

Our first objective is to derive a nontrivial representation of an output waveform of a sequential device as a quasi-smooth waveform. For this purpose, we make the following definition: If v is a Boolean atom and V is a bit vector, then $smooth(v, V)$ is the generalized bit vector V', where

(1) If $V =$ `NIL`, then $V' =$ `NIL`; otherwise:

(2) If $car(V) = v$, then $V' = cons(v, smooth(v, cdr(V)))$; otherwise:

(3) $V' = cons(\mathtt{Q}, smooth(car(V), cdr(V)))$.

Thus, if $v = v_0$ and $V = (v_1 \ldots v_n)$, then $V' = (v'_1 \ldots v'_n)$, where for $i = 1, \ldots, n$,

$$v'_i = \begin{cases} v_i \text{ if } v_i = v_{i-1} \\ \mathtt{Q} \text{ if } v_i \neq v_{i-1}. \end{cases}$$

Lemma 4.1 *Let $s = nth(j, O(M))$ be a registered output of a sequential module M. Let $w = nth(j, outp(M, sim(M, p, t_f)))$, where p is an admissible n-cycle input packet for S based at t_0 with value matrix $\mathcal{V}$ and period π, and $t_f \geq t_0 + (n+1)\pi$.*

Let $U = (sv(0, s, \mathcal{V}, M) \ldots sv(n, s, \mathcal{V}, M))$. Then w is an n-cycle quasi-smooth waveform based at $t_0 + 2\pi$ with value list $smooth(car(U), cdr(U))$ and period π.

Proof: For $0 \leq k \leq n$, let $U_k = (sv(n-k, s, \mathcal{V}, M) \ldots sv(n, s, \mathcal{V}, M))$ and $V_k = smooth(car(U_k), cdr(U_k))$. We shall prove, by induction on k, that w is a k-cycle quasi-smooth waveform based at $t_0 + (n-k+2)\pi$ with value list V_k and period π.

The base case $k = 0$ holds vacuously. For $k > 0$, since $cdr(V_k) = V_{k-1}$, we need only consider $car(V_k)$ and the behavior of w on $[t_0 + (n-k+2)\pi, t_0 + (n-k+3)\pi)$. If $car(V_k) = \mathtt{Q}$, there is nothing to prove. In the remaining case, $car(V_k) = car(U_k) = car(U_{k-1})$, i.e., $sv(n-k, s, \mathcal{V}, M) = sv(n-k+1, s, \mathcal{V}, M)$, and the result follows from Theorem 3.1. □

4.2 Describing Output as Input

Next, for a given quasi-smooth waveform with period π_s (representing that of the sender's clock), we would like to derive an alternative representation as a quasi-smooth waveform with a given period π_r (that of the receiver's clock). Let w be an n-cycle quasi-smooth waveform based at t_s (a rising edge of the sender's clock) with value list $V = (v_1 \ldots v_n)$ and period π_s. Assume $t_s \leq t_r < t_s + \pi_s$ (where t_r represents a rising edge of the receiver's clock). We shall construct a list of values $V' = warp(V, t_s, t_r, \pi_s, \pi_r)$ such that w is a quasi-smooth waveform based at t_r with value list V' and period π_r. The definition of $warp$ requires several auxiliary functions.

Let t satisfy $t_s < t \leq t_s + n\pi_s$. Choose k so that $t_s + (k-1)\pi_s < t \leq t_s + k\pi_s$. Then $1 \leq k \leq n$. (k represents the number of cycles of the sender that intersect the interval $[t_r, t)$.) We define

$$sig(V, t_s, t, \pi_s) = \begin{cases} v_1 \text{ if } v_1 = v_2 = \ldots = v_k \\ \mathtt{Q} \text{ if not.} \end{cases}$$

Under the same constraints on t, choose ℓ so that $t_s + \ell\pi_s \leq t < t_s + (\ell+1)\pi_s$. Then $0 \leq \ell \leq n$. ($t_s + \ell\pi_s$ represents the maximum sender's rising edge that is not exceeded by t.) We define

$$t_s^+(V, t_s, t, \pi_s) = t_s + \ell\pi_s$$

and

$$lst^+(V, t_s, t, \pi_s) = (v_{\ell+1} \ \ldots \ v_n).$$

Now we may define $V' = warp(V, t_s, t_r, \pi_s, \pi_r)$: If $t_r + \pi_r > t_s + n\pi_s$, then $V' = \texttt{NIL}$; otherwise,

$$V' = cons(sig, warp(lst^+, t_s^+, t_r + \pi_r, \pi_s, \pi_r)),$$

where $sig = sig(V, t_s, t_r + \pi_r, \pi_s)$, $lst^+ = lst^+(V, t_s, t_r + \pi_r, \pi_s)$, and $t_s^+ = t_s^+(V, t_s, t_r + \pi_r, \pi_s)$.

Lemma 4.2 *Let w be a quasi-smooth n-cycle waveform based at t_s with value list V and period π_s. Let $\pi_r > 0$ and $t_s \leq t_r < t_s + \pi_s$. Let $V' = warp(V, t_s, t_r, \pi_s, \pi_r)$ and let n' be the length of V'. Then w is a quasi-smooth n'-cycle waveform based at t_r with value list V' and period π_r.*

Proof: We may assume $t_r + \pi_r \leq t_s + n\pi_s$, for otherwise, $n' = 0$. Let $V = (v_1 \ \ldots \ v_n)$ and let sig, lst^+, and t_s^+ be defined as in the definition of $warp$. By induction, we may further assume that w is a quasi-smooth $(n'-1)$-cycle waveform based at $t_r + \pi_r$ with value list $cdr(V') = warp(lst^+, t_s^+, t_r + \pi_r, \pi_s, \pi_r)$ and period π_r. We need only show that either $car(V') = sig = \texttt{Q}$, or $\hat{w}$ has the constant value sig on the cycle $[t_r, t_r + \pi_r)$.

Suppose $sig \neq \texttt{Q}$. Choose k so that $t_s + (k-1)\pi_s < t_r + \pi_r \leq t_s + k\pi_s$. According to the definition of sig, $sig = v_1 = v_2 = \ldots = v_k$, and hence, $\hat{w}(t) = sig$ for all $t \in [t_s, t_s + k\pi_s) \supseteq [t_r, t_r + \pi_r)$. $\Box$

4.3 Eliminating Metastability

Lemmas 4.1 and 4.2 together provide a representation of a registered output waveform from the sender as a quasi-smooth waveform with respect to the receiver's clock. In order to achieve communication, we shall design a clocked state-holding device, called a *d-latch*, that converts a quasi-smooth input to a stable output. In our asynchronous circuit, this device will share the receiver's clock, and its output will be connected to the receiver's input.

The d-latch will consist of an inverter and three nand gates. Its functionality will depend on the relative delays of these components. Thus, along with our standard gates `not1` and `nand2`, both of which have delay 2000, we shall require the following faster nand gate, `fnand2`:

```
(BEHAV (A B) ((NAND2 A B)) (1000) (INERTIAL))
```

We define `dlatch` to be the following module, which is diagrammed in Fig. 5:

```
(STRUCT (CLK D) (S2)
  (not1 nand2 nand2 fnand2)
  ((CLK) (CLK D) (S1 S3) (S0 S2))
  ((S0) (S1) (S2) (S3)))
```

Unlike all other circuits that we have encountered, the specified behavior of `dlatch` will also depend on the unique character of inertial delay. In particular, we shall need the following result:

Lemma 4.3 *Let M be a behavioral module with $nth(j, O(M)) = s$, $nth(j, D(M)) = d$, and $nth(j, P(M)) =$* `INERTIAL`*. Let p be an input packet for M, let v be the combinational value of s w.r.t. $\hat{p}(t_0)$, and let $w = nth(j, sim(M, p, t_0)$.*

(a) If $\hat{w}(t_0) = v$, then $w = hist(w, t_0)$;

(a) If $\hat{w}(t_0) \neq v$, then $w = cons((v, t_1), hist(w, t_0))$, where $t_0 < t_1 \leq t_0 + d$.

Proof: By Lemma 2.10 and the definition of *exec*, $w = inertial(w, v, t_0, t_0 + d)$. The lemma follows from the definition of *inertial*. □

The behavioral specification of `dlatch` is an instance of the following, with $d_0 = d_1 = d_2 = 2000$ and $d_3 = 1000$.

Lemma 4.4 *Let G_0 be the inverter*

`(BEHAV (A) (NOT1 A) (`d_0`) (INERTIAL))`

and for $i = 1, 2, 3$, let G_i be the nand gate

`(BEHAV (A B) (NAND2 A B) (`d_i`) (INERTIAL))`,

where $d_1 \leq d_0$ and $d_0 + d_3 < d_1 + d_2$. Let $D = d_0 + d_1 + d_2 + d_3$. Let L be the module

```
(STRUCT (CLK) (D)
  (G0 G1 G2 G3)
  ((CLK) (CLK D) (S1 S3) (S2 S0))
  ((S0) (S1) (S2) (S3))).
```

Let $p = (w_{\text{CLK}}\, w_{\text{D}})$ be an input packet for L, and assume that

$$\hat{w}_{\text{CLK}}(t) = \begin{cases} \mathcal{T} & \textit{for all}\, t \in [t_+, t_-) \\ \mathcal{F} & \textit{for all}\, t \in [t_-, t_f), \end{cases}$$

where $t_- > t_+ + D$ and $t_f > t_- + D$. Let $((w_0)\,(w_1)\,(w_2)\,(w_3)) = sim(L, p, t_f)$. Then $\hat{w}_2$ has a constant value v on $[t_- + D, t_f)$. If $\hat{w}_{\text{D}}$ has a constant value u on $[t_+, t_f)$, then $u = v$.

Proof: For each $t \in \mathbf{N}$, let $B_t = ((w_{0,t})\,(w_{1,t})\,(w_{2,t})\,(w_{3,t})) = sim(L, p, t)$. Then for $i = 0, \ldots, 3$, $w_i = w_{i,t_f}$. Let $t_0 = t_- + d_0$. For each $t \geq t_0$, the following results may be derived from Lemmas 3.1 and 2.9:

(a) $\hat{w}_{0,t}$ has the constant value $\mathcal{F}$ on $[t_+ + d_0, t_0)$;
(b) $\hat{w}_{3,t}$ has the constant value $\mathcal{T}$ on $[t_+ + d_0 + d_3, t_0 + d_3)$;
(c) $\hat{w}_{0,t}$ has the constant value $\mathcal{T}$ on $[t_0, t_f + d_0)$;
(d) $\hat{w}_{1,t}$ has the constant value $\mathcal{T}$ on $[t_- + d_1, t_f + d_1)$.

In particular, for each $t \geq t_0$, $\hat{w}_{0,t}$ and $\hat{w}_{1,t}$ are both constant on $[t_0, t_f)$.

By Lemma 2.9, $(w_{2,t}) = sim(G_2, (w_{1,t}\, w_{3,t}), t)$ and $(w_{3,t}) = sim(G_3, (w_{0,t}\, w_{2,t}), t)$. We shall apply Lemma 4.3 to both G_2 and G_3.

We shall show that for some $t_1 \in [t_0, t_- + D)$ and some $v \in \mathbf{B}$, $\hat{w}_{2,t_1}(t_1) = v$ and $\hat{w}_{3,t_1}(t_1) = not1(v)$. Let $w_{2,t_0}(t_0) = v_2$ and $w_{3,t_0}(t_0) = v_3$. We consider the following cases:

Case 1: $v_3 = not1(v_2)$. In this case, we take $t_1 = t_0$ and $v = v_2$.

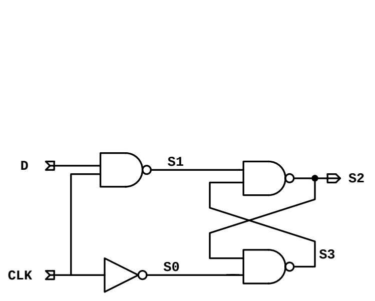
Case 2: $v_3 = v_2$. By Lemma 4.3(b), $w_{2,t_0} = cons((not1(v_2), t_2), hist(w_{2,t_0}, t_0))$, where $t_0 < t_2 \leq t_0 + d_2$, and $w_{3,t_0} = cons((not1(v_2), t_t), hist(w_{3,t_0}, t_0))$, where $t_0 < t_3 \leq t_0 + d_3$.

Subcase 2a: $t_3 < t_2$. Here, $t_{next}(t_0, p, B_{t_0}, L) = t_3$. By Lemma 2.7,$\hat{w}_{2,t_3}(t_3) = \hat{w}_{2,t_0}(t_3) = v_2$ and $\hat{w}_{3,t_3}(t_3) = \hat{w}_{3,t_0}(t_3) = not1(v_2)$. Thus, we have $t_1 = t_3$ and $v = v_2$.

Subcase 2b: $t_2 < t_3$. In this case, $t_{next}(t_0, p, B_{t_0}, L) = t_3$, and we have $\hat{w}_{2,t_2}(t_2) = \hat{w}_{2,t_0}(t_2) = not1(v_2)$ and $\hat{w}_{3,t_2}(t_2) = \hat{w}_{3,t_0}(t_2) = v_2$. In this case, $t_1 = t_2$ and $v = not1(v_2)$.

Subcase 2c: $t_2 = t_3$. We have $\hat{w}_{2,t_2}(t_2) = \hat{w}_{3,t_2}(t_2) = not1(v_2)$. By Lemma 4.3(b), $w_{2,t_2} = cons((v_2, t_2 + d_2), w_{2,t_0})$, and $w_{3,t_2} = cons((v_2, t_2 + d_3), w_{3,t_0})$. It follows from our hypotheses that $d_3 < d_2$. Hence, $\hat{w}_{2,t_2+d_3}(t_2 + d_3) = not1(v_2)$ and $\hat{w}_{3,t_2+d_3}(t_2 + d_3) = v_2$. Thus, $t_1 = t_2 + d_3$ and $v = not1(v_2)$.

Now, by Lemma 4.3(a), $\hat{w}_{2,t_1} = hist(\hat{w}_{2,t_1}, t_1)$ and $\hat{w}_{3,t_1} = hist(\hat{w}_{3,t_1}, t_1)$. Hence, $t_{next}(t_1, p, B_{t_1}, L) \geq t_f$. It follows that for any $t' \in [t_1, t_f)$, $B_{t'} = B_{t_1}$, and in particular, $w_{2,t_f}(t') = w_{2,t'}(t') = w_{2,t_1}(t') = v$. Thus, w_{2,t_f} has the constant value v on $[t_1, t_f) \supseteq [t_- + D, t_f)$.



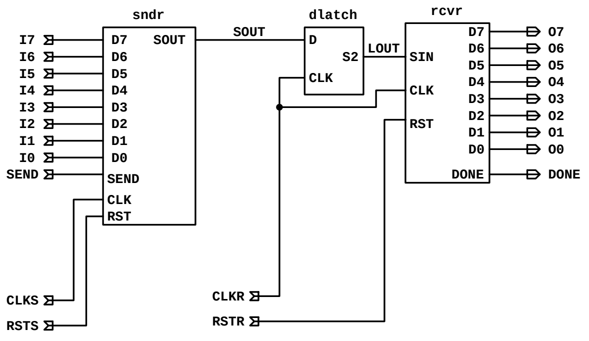


Figure 5: (a) `dlatch` (b) `bpm`

Finally, suppose that $\hat{w}_{\text{D}}$ has a constant value u on $[t_+, t_f)$. Then $\hat{w}_1(t) = not1(u)$ for $t \in [t_+ + d_1, t_- + d_1)$. Since $\hat{w}_3(t) = \mathcal{T}$ on $[t_+ + d_0 + d_3, t_0 + d_3)$, the combinational value corresponding to S2 is u on the intersection of these intervals, $[max(t_+ + d_1, t_+ + d_0 + d_3), min(t_- + d_1, t_0 + d_3))$. Thus, by Lemma 3.1, $\hat{w}_2(t) = u$ for $t \in [max(t_+ + d_1 + d_2, t_+ + d_0 + d_3 + d_2), min(t_- + d_1 + d_2, t_0 + d_3 + d_2))$. In particular, $\hat{w}_2(t) = u$ for $t \in [t_0, t_0 + d_3 + d_2)$. Thus, $v_2 = u$. Moreover, Subcases 2b and 2c, in which $\hat{w}$ assumes the value $not1(v_2)$ at some point in this interval, are eliminated. In the remaining cases, $v = v_2 = u$. □

In order to avail ourselves of the results of [4], we must restate Lemma 4.4 in terms of Moore's function *det*. If V is a generalized bit vector and *oracle* is a bit vector, then $det(V, oracle)$ is the bit vector V', defined as follows:

(1) If $V =$ `NIL`, then $V' =$ `NIL`; otherwise:

(2) If $car(V) \in \mathbf{B}$, then $V' = cons(car(V), det(cdr(V), oracle))$; otherwise:

(3) If $oracle =$ `NIL`, then $V' = cons(\mathcal{T}, det(cdr(V), oracle))$; otherwise:

(4) $V' = cons(car(oracle), det(cdr(V), cdr(oracle)))$.

Lemma 4.5 *Let $p = (w_{\text{CLK}}\ w_{\text{D}})$ be an input packet for* `dlatch`*, where w_{CLK} is an n-cycle pulse based at t_0 with high $h > 7000$, low $\ell > 7000$, and period $\pi = h + \ell$, and w_{D} is a quasi-smooth n-cycle waveform based at t_0 with value list V and period π. Let $((w_0)\ (w_1)\ (w_2)\ (w_3)) = sim($*`dlatch`*$, p, t_f)$, where $t_f \geq t_0 + n\pi$. Then for some bit vector oracle, w_2 is a stable n-cycle waveform based at t_0 with setup $\ell - 7000$, value list $det(V, oracle)$, and period π.*

Proof: We induct on n. For $n = 0$, the statement is vacuous. For $n > 0$, we may assume that w_2 is a stable $(n-1)$-cycle waveform based at $t_0 + \pi$ with setup $\ell - 7000$, value list $det(cdr(V), oracle')$, and period π. By Lemma 4.4, $\hat{w}_2$ has a constant value v on $[t_0 + h + 7000, t_0 + \pi) = [t_0 + \pi - (\ell - 7000), t_0 + \pi)$, and if $car(V) \neq \mathtt{Q}$, then $car(V) = v$. If $car(V) = \mathtt{Q}$, then let $oracle = cons(v, oracle')$; otherwise, let $oracle = oracle'$. In either case, w_2 is a stable n-cycle waveform based at t_0 with setup $\ell - 7000$, value list $det(V, oracle)$, and period π. □

4.4 The Main Theorem

In Section 5, we shall apply the results of this section to a circuit `bpm`, consisting of two sequential submodules, `sndr` and `rcvr`, and a `dlatch`: According to the definitions that we shall present later, `sndr` has 9 data inputs and one registered output, `SOUT`, while `rcvr` has one data input, `SIN`, and 9 outputs. The circuit `bpm`, which is diagrammed in Fig. 5, is defined as follows:

```
(STRUCT
  (CLKS RSTS CLKR RSTR SEND I0 I1 I2 I3 I4 I5 I6 I7)
  (DONE O0 O1 O2 O3 O4 O5 O6 O7)
  (sndr dlatch rcvr)
  ((CLKS RSTS SEND I0 I1 I2 I3 I4 I5 I6 I7) (CLKR SOUT) (CLKR RSTS LOUT))
  ((SOUT) (LOUT) (DONE O0 O1 O2 O3 O4 O5 O6 O7)))
```

The following theorem summarizes our results on asynchrony, as they pertain to the module `bpm`. The theorem refers to Moore's function *asynch*, which is defined as follows: Let V and *oracle* be bit vectors and let $t_s, t_r, \pi_s, \pi_r \in \mathbf{N}$ such that $\pi_s > 0$, $\pi_r > 0$, and $t_s \leq t_r < t_s + \pi_s$. Then

$$asynch(V, t_s, t_r, \pi_s, \pi_r, oracle) = det(warp(smooth(\mathcal{T}, V), t_s, t_r, \pi_s, \pi_r), oracle).$$

Theorem 4.1 *Let $p = (w_{\text{CLKS}}\, w_{\text{RSTS}}\, w_{\text{CLKR}}\, w_{\text{RSTR}}\, w_{\text{SEND}}\, w_0 \,\ldots\, w_7)$ be an input packet for* `bpm`*, where*

(a) $(w_{\text{CLKS}}\, w_{\text{RSTS}}\, w_{\text{SEND}}\, w_0 \,\ldots\, w_7)$ is an admissible n_s-cycle input packet for `sndr` *based at b_s with value matrix $\mathcal{V}$ and period π_s;*

(b) w_{CLKR} is an admissible $(n_r + 2)$-cycle pulse for `rcvr` *based at b_r with high $h > 7000$, low $\ell > 7000 + setup(\mathtt{SIN}, \mathtt{rcvr})$, and period $\pi_r = h + \ell$;*

(c) w_{RSTR} is an admissible $(n_r + 1)$-cycle reset waveform for `rcvr` *based at b_r with period π_r.*

Let $t_r = b_r + \pi_r$. Assume that $b_s + 2\pi_s \leq t_r \leq b_s + (n_s + 2)\pi_s \leq t_r + n_r\pi_r$. Choose j so that $b_s + j\pi_s \leq t_r < b_s + (j+1)\pi_s$ and let $t_s = b_s + j\pi_s$. Assume $sv(j-2, \mathtt{SOUT}, \mathcal{V}, \mathtt{sndr}) = \mathcal{T}$.

Let $U = (sv(j-1, \texttt{SOUT}, \mathcal{V}, \texttt{sndr}) \ldots sv(n_s, \texttt{SOUT}, \mathcal{V}, \texttt{sndr}))$*. Let* w_{LOUT} *be the waveform for* `LOUT` *determined by* $sim(\texttt{bpm}, p, t_f)$*, where* $t_f \geq t_r + n_r\pi_r$*. Then for some bit vector oracle,* $(w_{\text{CLKR}}\ w_{\text{RSTR}}\ w_{\text{LOUT}})$ *is an admissible input packet for* `rcvr` *based at* b_r *with value matrix* $(asynch(U, t_s, t_r, \pi_s, \pi_r, oracle))$ *and period* π_r*.*

Proof: Let w_{SOUT} be the waveform for `SOUT` determined by $sim(\texttt{bpm}, p, t_f)$. According to Lemma 4.1, w_{SOUT} is a quasi-smooth waveform based at t_s with value list $smooth(\mathcal{T}, U)$ and period π_s. It follows from Lemma 4.2 that w_{SOUT} is also a quasi-smooth waveform based at t_r with value list $warp(smooth(\mathcal{T}, U), t_s, t_r, \pi_s, \pi_r)$ and period π_r. Finally, by Lemma 4.5, w_{LOUT} is a stable waveform based at t_r with setup $\ell - 7000 > setup(\texttt{SIN}, \texttt{rcvr})$, value list

$$det(warp(smooth(\mathcal{T}, U), t_s, t_r, \pi_s, \pi_r), oracle) = asynch(U, t_s, t_r, \pi_s, \pi_r, oracle)),$$

for some $oracle$, and period π_r. $\Box$

5 Biphase Mark

Moore's formulation [4] of the biphase mark protocol is based on two functions, *send* and *recv*, which represent the computations performed by the sender and the receiver, respectively. After presenting the definitions of these functions, we shall implement them in the design of the sequential modules `sndr` and `rcvr`. Then, using a theorem of Moore in combination with results of Section 4, we shall show that the circuit `bpm` achieves communication between these modules.

5.1 Sending

The function *send* returns a bit vector that represents an encoding of a given input bit vector *msg*. Each bit of *msg* is encoded as a bit vector called a *cell*, computed as the value of $cell(x, n, k, b)$, where b is the bit of *msg* to be encoded, x is the final bit of the preceding cell, and n and k are parameters of the protocol. A cell consists of two *subcells*, each of which is a uniform bit vector: a *mark subcell* of length n, followed by a *code subcell* of length k. The mark subcell is intended as a signal to the receiver that a new cell has been entered: each of its bits is $not1(x)$. The code subcell is the region in which the receiver is expected to look for information from which it will derive the value b of the encoded bit: if $b = \mathcal{T}$, then each bit of this subcell is x; if $b = \mathcal{F}$, each bit is $not1(x)$.

The definition of *cell* requires three auxiliary functions. First, the subcells are constructed by the function *listn*: for any $n \in \mathbf{N}$ and any x, $listn(n, x)$ is the uniform vector $(x \ldots x)$ of length n. Next, the two subcells are combined by the function *app*: for any two lists $L = (a_1 \ldots a_n)$ and $M = (b_1 \ldots b_m)$, $app(L, M) = (a_1 \ldots a_n b_1 \ldots b_m)$. Finally, the bit occurring in the code subcell is determined by the Boolean function *equal*, where $equal(x, y) = \mathcal{T}$ iff $x = y$, i.e., $equal(x, y) = not1(xor2(x, y))$.

Now, we may define

$$cell(x, n, k, b) = app(listn(n, not1(x)), listn(k, equal(x, b))),$$

and $cells(x, n, k, msg)$ is defined as

(1) `NIL`, if $msg =$ `NIL`;

(2) $app(cell(x, n, k, car(msg)), cells(equal(x, car(msg)), n, k, cdr(msg)))$, if $msg \neq$ `NIL`.

The protocol includes the convention that the value $\mathcal{T}$ is transmitted until the encoded message is sent. Thus, the encoded bit vector constructed by *send* includes "pads" consisting arbitrarily many copies of $\mathcal{T}$ on both sides of the cells. The arguments of *send* include the lengths p_1 and p_2 of these pads:

$$send(msg, p_1, n, k, p_2) = app(listn(p_1, \mathcal{T}), app(cells(\mathcal{T}, n, k, msg), listn(p_2, \mathcal{T}))).$$

5.2 Receiving

Next, we define $recv(i, x, j, L)$[1], which may be shown, under suitable assumptions, to be the inverse of *send*. This function recovers a bit of the encoded message from each cell by first detecting the beginning of the mark subcell, and then reading and decoding a bit at a predetermined location within the cell, which has been calculated to lie within the code subcell. Its arguments are interpreted as follows: i is the number of bits of the original message yet to be recovered, x is the last bit to have been read (from the preceding cell), j is the location within the cell of the bit to be read, and L is the remaining input stream.

The beginning of a new cell is detected by the function $scan(x, L)$, which successively removes bits from the beginning of the list L until a value different from x is found. The recursive definition follows:

(1) If $L =$ `NIL`, then $scan(x, L) =$ `NIL`; otherwise:

(2) If $car(L) = x$, then $scan(x, L) = scan(x, cdr(L))$; otherwise:

[1] For technical reasons, we shall slightly modify Moore's original definition of this function. Our modification does not affect the validity of any of his results.

(3) $scan(x, L) = L$.

We shall require one other auxiliary function: If $n \in \mathbf{N}$ and L is a list, then $cdrn(n, L)$ is defined to be

(1) L, if $n = 0$;

(2) $cdrn(n-1, cdr(L))$, if $n > 0$.

Finally, we define $recv(i, x, j, L)$ to be the bit vector msg, where

(1) If $i = 0$, then $msg = \texttt{NIL}$; otherwise:

(2) Let $S = scan(x, L)$. If $length(S) \leq k$, then $msg = \texttt{NIL}$; otherwise:

(3) Let $b = nth(k+1, S)$ and $L' = cdrn(k+1, S)$. If $b = x$, then $msg = cons(\mathcal{T}, recv(i-1, b, j, L')$; otherwise:

(4) $msg = cons(\mathcal{F}, recv(i-1, b, j, L')$.

5.3 Moore's Theorem

Moore has proved a statement of correctness of the protocol for certain values of the parameters. The lengths of the mark and code subcells generated by *send* are taken to be $n = 5$ and $k = 13$, respectively. The index of the bit read by *recv* following the detection of an edge is $j = 10$, i.e., the eleventh bit after the edge is sampled. The theorem also depends on an assumption concerning the proximity of the two clock periods:

Theorem 5.1 (Moore) *Let $\pi_s > 0$, $\pi_r > 0$, and $17\pi_r \leq 18\pi_s \leq 19\pi_r$. Let $t_s \leq t_r < t_s + \pi_s$. Let msg be a bit vector of length k. Then for any bit vector oracle and any numbers p_1 and p_2,*

$$recv(k, \mathcal{T}, 10, asynch(send(msg, p_1, 5, 13, p_2), t_s, t_r, \pi_s, \pi_r, oracle)) = msg.$$

We shall apply Moore's theorem to the specification of the circuit `bpm`. The sequential submodules `sndr` and `rcvr` of `bpm` remain to be defined. As we present the definitions of the these modules and their components, which are diagrammed in Figs. 6–10, we shall derive characterizations of their behavior that are analogous to Propositions 3.1 and 3.2. The proofs of these results are based on straightforward calculations and have all been mechanically checked. Therefore, the details of these proofs are omitted here.

5.4 Basic Components

The message that is transmitted from `sndr` to `rcvr` will consist of eight bits. It is stored (by both `sndr` and `rcvr`) in a shift register, `shift8`, which is constructed from eight copies of the following 3-port cell, `port3`:

```
(STRUCT
  (CLK RST SHIFT SIN LOAD DIN)
  (Q)
  (edff nand2 nand2 or2 nand2)
  ((CLK RST S3 S4) (DIN LOAD) (SIN SHIFT) (LOAD SHIFT) (S1 S2))
  ((Q QN) (S1) (S2) (S3) (S4)))
```

The behavior of `port3` may be derived easily from that of `edff` (Proposition 3.1):

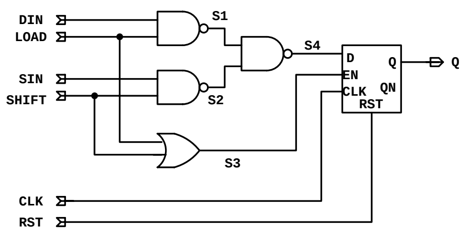
Proposition 5.1 *Let Σ and $V = (shift\ sin\ load\ din)$ be a state and a data vector for* `port3`. *Assume that shift and load are not both $\mathcal{T}$. Then*

$$nv(\mathtt{Q}, V, \Sigma, \mathtt{port3}) = \Sigma;$$

$$next(V, \Sigma, \mathtt{port3}) = \begin{cases} sin & \text{if } shift = \mathcal{T} \text{ and } load = \mathcal{F} \\ din & \text{if } shift = \mathcal{F} \text{ and } load = \mathcal{T} \\ \Sigma & \text{if } shift = \mathcal{F} \text{ and } load = \mathcal{F}. \end{cases}$$

The register `shift8` is defined as follows:

```
(STRUCT
  (CLK RST LOAD SHIFT SIN D0 D1 D2 D3 D4 D5 D6 D7)
  (Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7)
  (port3 port3 port3 port3 port3 port3 port3 port3)
  ((CLK RST SHIFT SIN LOAD D0)
   (CLK RST SHIFT Q0 LOAD D1)
   (CLK RST SHIFT Q1 LOAD D2)
   (CLK RST SHIFT Q2 LOAD D3)
   (CLK RST SHIFT Q3 LOAD D4)
   (CLK RST SHIFT Q4 LOAD D5)
   (CLK RST SHIFT Q5 LOAD D6)
   (CLK RST SHIFT Q6 LOAD D7))
  ((Q0) (Q1) (Q2) (Q3) (Q4) (Q5) (Q6) (Q7)))
```



(a) `port3`

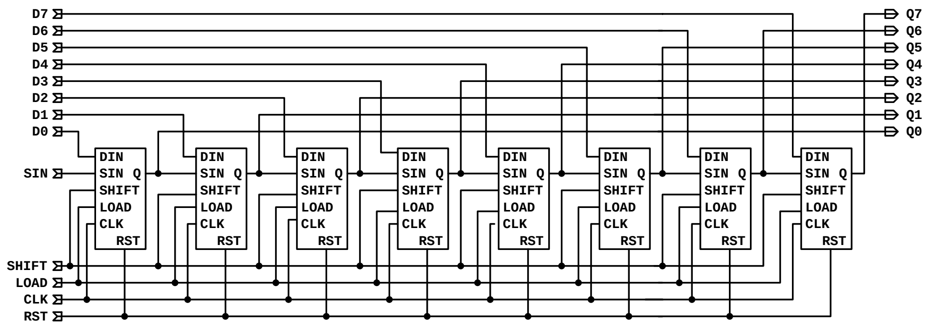


Figure 6: (b) `shift8`

Proposition 5.2 *Let $\Sigma = (\sigma_0 \ \ldots \ \sigma_7)$ and $V = (load\, shift\, sin\, d_0 \ \ldots \ d_7)$ be a state and a data vector for* `shift8`*. Assume that shift and load are not both $\mathcal{T}$. Then*

$$nv(\mathtt{Q}i, V, \Sigma, \mathtt{shift8}) = \sigma_i, \ i = 0, \ldots, 7;$$

$$next(V, \Sigma, \mathtt{shift8}) = \begin{cases} (sin\ \sigma_0 \ \ldots \ \sigma_6) & \textit{if } shift = \mathcal{T} \textit{ and } load = \mathcal{F} \\ (d_0 \ \ldots \ d_7) & \textit{if } shift = \mathcal{F} \textit{ and } load = \mathcal{T} \\ \Sigma & \textit{if } shift = \mathcal{F} \textit{ and } load = \mathcal{F}. \end{cases}$$

In order to describe the shifting operation that is performed by `shift8`, we define, for any $b \in \mathbf{B}$ and any bit vector V,

$$shift(b, V) = \begin{cases} \mathtt{NIL} & \text{if } V = \mathtt{NIL} \\ cons(b, shift(car(V), cdr(V))) & \text{if } V \neq \mathtt{NIL}. \end{cases}$$

Thus, $shift(sin, (\sigma_0 \ \ldots \ \sigma_7)) = (sin\ \sigma_0 \ \ldots \ \sigma_6)$.

In addition to `dff` and `edff`, we shall require two other versions of the flip-flop. The first of these, `cdff`, has an input `CLR`, which may be used to override the other data input `D` and reinitialize the state:

```
(STRUCT
  (CLK RST CLR D)
  (Q QN)
  (dff not1 nand2)
  ((CLK RST DCN) (CLR) (D CN))
  ((Q QN) (CN) (DCN)))
```

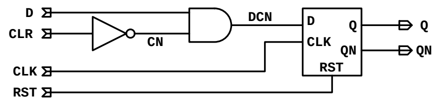
Proposition 5.3 *Let Σ and $V = (clr\ d)$ be a state and a data vector for* `cdff`*. Then*

$$nv(\mathtt{Q}, V, \Sigma, \mathtt{cdff}) = \Sigma \textit{ and } nv(\mathtt{QN}, V, \Sigma, \mathtt{cdff}) = not1(\Sigma);$$

$$next(V, \Sigma, \mathtt{cdff}) = \begin{cases} \mathcal{F} & \textit{if } clr = \mathcal{T} \\ d & \textit{if } clr = \mathcal{F}. \end{cases}$$

The second, `cedff`, is a combination of `edff` and `cdff`:

```
(STRUCT
  (CLK RST CLR EN D)
  (Q QN)
  (dff not1 not1 nand3 nand3 nand2)
  ((CLK RST S5) (EN) (CLR) (Q S1 S2) (D S2 EN) (S3 S4))
  ((Q QN) (S1) (S2) (S3) (S4) (S5)))
```



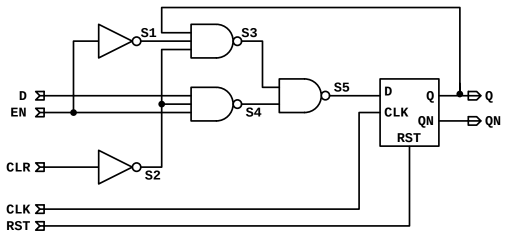


Figure 7: (a) `cdff` (b) `cedff`

Proposition 5.4 *Let* Σ *and* $V = (clr\ en\ d)$ *be a state and a data vector for* `cedff`*. Then*

$$nv(\texttt{Q}, V, \Sigma, \texttt{cedff}) = \Sigma \text{ and } nv(\texttt{QN}, V, \Sigma, \texttt{cedff}) = not1(\Sigma);$$

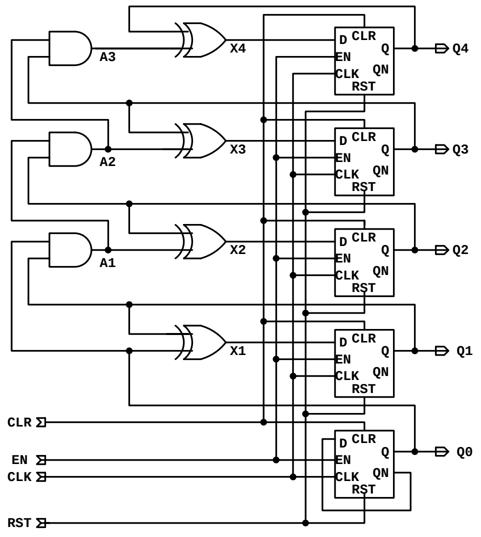
$$next(V, \Sigma, \texttt{cedff}) = \begin{cases} \mathcal{F} & \textit{if } clr = \mathcal{T} \\ d & \textit{if } clr = \mathcal{F} \textit{ and } en = \mathcal{T} \\ \Sigma & \textit{if } clr = \mathcal{F} \textit{ and } en = \mathcal{F}. \end{cases}$$

Using `cedff`, we construct the following 5-bit counter, `count5`:

```
(STRUCT
  (CLK RST CLR EN)
  (Q0 Q1 Q2 Q3 Q4)
  (cedff cedff cedff cedff cedff
   and2 and2 and2 xor2 xor2 xor2 xor2)
  ((CLK RST CLR EN QN0)
   (CLK RST CLR EN X1)
   (CLK RST CLR EN X2)
   (CLK RST CLR EN X3)
   (CLK RST CLR EN X4)
   (Q0 Q1) (A1 Q2) (A2 Q3) (Q0 Q1) (Q2 A1) (Q3 A2) (Q4 A3))
  ((Q0 QN0) (Q1 QN1) (Q2 QN2) (Q3 QN3) (Q4 QN4)
   (A1) (A2) (A3) (X1) (X2) (X3) (X4)))
```

Proposition 5.5 *Let* $\Sigma = (\sigma_0 \ldots \sigma_4)$ *and* $V = (clr\,en)$ *be a state and a data vector for* `count5`*. Then*

$$nv(\texttt{Q}i, V, \Sigma, \texttt{count5}) = \sigma_i,\ i = 0, \ldots, 4;$$



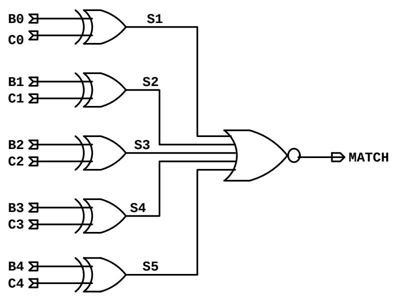


Figure 8: (a) `count5` (b) `comp5`

$$next(V, \Sigma, \texttt{count5}) = \begin{cases} listn(5, \mathcal{F}) & \textit{if } clr = \mathcal{T} \\ inc(cnt) & \textit{if } clr = \mathcal{F} \textit{ and } en = \mathcal{T} \\ \Sigma & \textit{if } clr = \mathcal{F} \textit{ and } en = \mathcal{F}. \end{cases}$$

For convenience in representing states of both *count3* and *count5*, we define, for $k \in \mathbf{N}$ and $n \in \mathbf{N}$,

$$bv_k(n) = \begin{cases} listn(k, \mathcal{F}) & \text{if } n = 0 \\ inc(bv_k(n-1)) & \text{if } n > 0. \end{cases}$$

Thus, $bv_k(n)$ is the k-bit vector that represents the number n.

We shall also require a combinational module, the following 5-bit comparator `comp5`:

```
(STRUCT
  (C0 B0 C1 B1 C2 B2 C3 B3 C4 B4)
  (MATCH)
  (xor2 xor2 xor2 xor2 xor2 nor5)
  ((C0 B0) (C1 B1) (C2 B2) (C3 B3) (C4 B4) (S1 S2 S3 S4 S5))
  ((S1) (S2) (S3) (S4) (S5) (MATCH)))
```

This module simply determines whether two given 5-bit vectors are equal, i.e.,

$$cv(\texttt{MATCH}, (c_0\, b_0\, c_1\, b_1\, \ldots\, c_4\, b_4), \texttt{comp5}) = \begin{cases} \mathcal{T} & \text{if } (c_0\, \ldots\, c_4) = (b_0\, \ldots\, b_4) \\ \mathcal{F} & \text{if not.} \end{cases}$$

5.5 The Sender

The action of `sndr` is controlled by the submodule `scount`, which is defined as follows:

```
(STRUCT
  (CLK RST STOP BIT)
  (MARK CODE)
  (cdff count5 or2 or2 t0 f0 comp5 comp5)
  ((CLK RST STOP S1) (CLK RST S2 Q) (BIT Q) (STOP BIT) () ()
   (F Q0 F Q1 T Q2 F Q3 F Q4) (T Q0 F Q1 F Q2 F Q3 T Q4))
  ((Q QN) (Q0 Q1 Q2 Q3 Q4) (S1) (S2) (T) (F) (MARK) (CODE)))
```

A state of `scount` is a list $(on\, cnt)$ of two components, corresponding to the two sequential submodules, `cdff` and `count5`. As long as both data inputs are $\mathcal{F}$, the value of on remains constant. While $on = \mathcal{T}$, cnt is incremented repreatedly; while $on = \mathcal{F}$, cnt remains unchanged. If either input is $\mathcal{T}$, then on is set accordingly and cnt is reset to $bv_5(0)$. The output values are both determined by cnt:

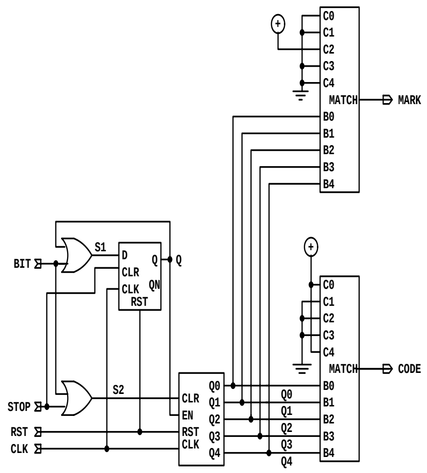
Proposition 5.6 *Let $\Sigma = (on\, cnt)$ and $V = (stop\, bit)$ be a state and a data vector for* `scount`*. Then*

$$nv(\texttt{MARK}, V, \Sigma, \texttt{scount}) = \begin{cases} \mathcal{T} & \text{if } cnt = bv_5(4) \\ \mathcal{F} & \text{if } cnt \neq bv_5(4); \end{cases}$$

$$nv(\texttt{CODE}, V, \Sigma, \texttt{scount}) = \begin{cases} \mathcal{T} & \text{if } cnt = bv_5(17) \\ \mathcal{F} & \text{if } cnt \neq bv_5(17); \end{cases}$$

$$next(V, \Sigma, \texttt{scount}) = \begin{cases} (\mathcal{F}\, bv_5(0)) & \text{if } stop = \mathcal{T} \\ (\mathcal{T}\, bv_5(0)) & \text{if } stop = \mathcal{F} \text{ and } bit = \mathcal{T} \\ (\mathcal{T}\, inc(cnt)) & \text{if } stop = bit = \mathcal{F} \text{ and } on = \mathcal{T} \\ (\mathcal{F}\, cnt) & \text{if } stop = bit = \mathcal{F} \text{ and } on = \mathcal{F}. \end{cases}$$

The definition of `sndr` is as follows:



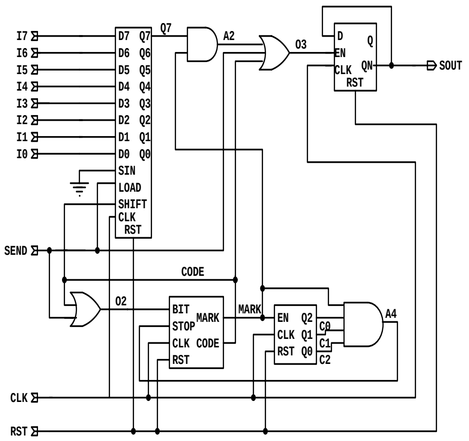


Figure 9: (a) `scount` (b) `sndr`

```
(STRUCT
  (CLK RST SEND I0 I1 I2 I3 I4 I5 I6 I7)
  (SOUT)
  (scount shift8 count3 edff or2 and2 and4 or3 f0)
  ((CLK RST A4 O2) (CLK RST SEND CODE F I0 I1 I2 I3 I4 I5 I6 I7)
   (CLK RST MARK) (CLK RST O3 SOUT) (CODE SEND) (Q7 MARK)
   (MARK C0 C1 C2) (A2 SEND CODE) ())
  ((MARK CODE) (Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7) (C0 C1 C2)
   (Q SOUT) (O2) (A2) (A4) (O3) (F)))
```

This module has two modes of operation. In one mode, it waits dormantly for the `SEND` input to become $\mathcal{T}$. When this occurs, the current values of the other eight data inputs are loaded into the shift register, the state of the flip-flop `edff` (which determines the output value) changes, and the controller `scount` begins counting. This mode is described by the following:

Proposition 5.7 *Let* $V = (s\ d_0\ \ldots\ d_7)$ *be a data vector for* `sndr`*, and let* $\Sigma = (\sigma_1\ \sigma_2\ \sigma_3\ \sigma_4)$ *be a state of* `sndr`*, where* $\sigma_1 = (on\ cnt)$*. Assume that* $on = \mathcal{F}$ *and* $cnt = bv_5(0)$*. Let* $\Sigma' = next(V, \Sigma, \texttt{sndr})$*.*

(a) If $s = \mathcal{T}$*, then* $\Sigma' = ((\mathcal{T}\ bv_5(0))\ (d_0\ \ldots\ d_7)\ \sigma_3\ not1(\sigma_4))$*;*
(b) If $s = \mathcal{F}$*, then* $\Sigma' = \Sigma$*.*

In the other mode of operation, the register contents are encoded and transmitted. Each register bit is encoded as a cell consisting of a 5-bit mark subcell and a

13-bit code subcell, as measured by `scount`. The number of cells that have been transmitted is recorded as the contents of `count3`. At the end of each mark subcell, this number is incremented. At the end of each code subcell, the `scount` counter is reset and the register contents are shifted:

Proposition 5.8 *Let $V = (s\ d_0\ \ldots\ d_7)$ be a data vector for* `sndr`*, and let $\Sigma = (\sigma_1\ \sigma_2\ \sigma_3\ \sigma_4)$ be a state of* `sndr`*, where $\sigma_1 = (on\ cnt)$ and $\sigma_2 = (q_0\ \ldots\ q_7)$. Assume that $s = \mathcal{F}$ and $on = \mathcal{T}$. Let $\Sigma' = next(V, \Sigma, \texttt{sndr})$.*

(a) If $cnt = bv_5(4)$ and $\sigma_3 = bv_3(7)$, then $\Sigma' = ((\mathcal{F} bv_5(0))\, \sigma_2\, inc(\sigma_3)\, xor2(q_7, \sigma_4))$;
(b) If $cnt = bv_5(4)$ and $\sigma_3 \neq bv_3(7)$, then $\Sigma' = ((\mathcal{T} bv_5(5))\, \sigma_2\, inc(\sigma_3)\, xor2(q_7, \sigma_4))$;
(c) If $cnt = bv_5(17)$, then $\Sigma' = ((\mathcal{T}\ bv_5(0))\ shift(\mathcal{F}, \sigma_2)\ \sigma_3\ not1(\sigma_4))$;
(d) If $cnt \neq bv_5(4)$ and $cnt \neq bv_5(17)$, then $\Sigma' = ((\mathcal{T}\ inc(cnt))\ \sigma_2\ \sigma_3\ \sigma_4)$.

Our main theorem on `sndr` is the following specification:

Proposition 5.9 *Let $\mathcal{V} = (V_{\text{SEND}}\, V_{\text{I0}}\ \ldots\ V_{\text{I7}})$ be a list of bit vectors, each of length $n \geq 144$. Let $m = n - 144$. Assume that for $j = 1, \ldots, n$,*

$$nth(j, V_{\text{SEND}}) = \begin{cases} \mathcal{T} & \text{if } j = m \\ \mathcal{F} & \text{if } j \neq m. \end{cases}$$

For $i = 0, \ldots, 7$, let $d_i = nth(m, V_{\text{I}i})$. For $j = 1, \ldots, n$, let $sv_j = sv(j, \texttt{SOUT}, \mathcal{V}, \texttt{sndr})$. Then $(sv_1\ \ldots\ sv_n) = send((d_7\ \ldots\ d_0), m, 5, 13, 0)$.

Proof: Let $\Sigma_j = state(j, \mathcal{V}, \texttt{sndr})$, $j = 0, \ldots, n$. By Proposition 5.7(b), for $j = 0, \ldots, m$,

$$\Sigma_j = \Sigma_0(\texttt{sndr}) = ((\mathcal{F}\ bv_5(0))\ \ listn(8, \mathcal{F})\ \ bv_3(0)\ \ \mathcal{F})$$

and hence $(sv_1\ \ldots\ sv_m) = listn(m, \mathcal{T})$. It remains to show that $(sv_{m+1}\ \ldots\ sv_n) = cells(\mathcal{T}, 5, 13, (d_7\ \ldots\ d_0))$.

By Proposition 5.7(a),

$$\Sigma_{m+1} = ((\mathcal{T}\ bv_5(0))\ \ (d_0\ \ \ldots\ \ d_7)\ \ bv_3(0)\ \ \mathcal{T}).$$

We shall show that for all k, $0 \leq k \leq 7$, if

$$\Sigma_{m+1+18k} = ((\mathcal{T}\ bv_5(0))\ \ app(listn(k, \mathcal{F}), (d_0\ \ldots\ d_{7-k}))\ \ bv_3(k)\ \ x),$$

then

$$(sv_{m+1+18k}\ \ldots\ sv_n) = cells(x, 5, 13, (d_{7-k}\ \ldots\ d_0)).$$

The proposition will follow from this result upon setting $k = 0$.

The proof is by induction on $7 - k$. In the base case, $k = 7$, our assumption is that

$$\Sigma_{m+1+18k} = \Sigma_{m+127} = ((\mathcal{T}\ bv_5(0))\ \ app(listn(7, \mathcal{F}), (d_0))\ \ bv_3(7)\ \ x).$$

By Proposition 5.8(d), for $\ell = 0, \ldots, 4$,

$$\Sigma_{m+127+\ell} = ((\mathcal{T}\ bv_5(\ell))\ \ app(listn(7, \mathcal{F}), (d_0))\ \ bv_3(7)\ \ x),$$

and by Proposition 5.8(a),

$$\Sigma_{m+127+5} = \Sigma_{m+132} = ((\mathcal{F}\ bv_5(0))\ \ app(listn(7, \mathcal{F}), (d_0))\ \ bv_3(0)\ \ xor2(d_0, x)).$$

By Proposition 5.7(b), $\Sigma_{m+132+\ell} = \Sigma_{m+132}$ for $\ell = 0, \ldots, 12$. It follows that

$$\begin{aligned}(sv_{m+127}\ \ldots\ sv_n) &= app(listn(5, not1(x)), listn(13, equal(d_0, x))) \\ &= cell(x, 5, 13, d_0) \\ &= cells(x, 5, 13, (d_0)).\end{aligned}$$

In the inductive case, $k < 7$, we again have, for $\ell = 0, \ldots, 4$,

$$\Sigma_{m+1+18k+\ell} = ((\mathcal{T}\ bv_5(\ell))\ \ app(listn(k, \mathcal{F}), (d_0\ \ldots\ d_{7-k}))\ \ bv_3(k)\ \ x)$$

by Proposition 5.8(d). By Proposition 5.8(b) and (d), for $\ell = 5, \ldots, 17$,

$$\Sigma_{m+1+18k+\ell} = ((\mathcal{T}\ bv_5(\ell))\ \ app(listn(k, \mathcal{F}), (d_0\ \ldots\ d_{7-k}))\ \ bv_3(k+1)\ \ xor2(d_{7-k}, x)).$$

Thus, $(sv_{m+1+18k}\ \ldots\ sv_{m+1+18k+17})$ is

$$app(listn(5, not1(x)), listn(13, equal(d_{7-k}, x))) = cell(x, 5, 13, d_{7-k}).$$

By Proposition 5.8(c), $\Sigma_{m+1+18(k+1)}$ is

$$((\mathcal{T}\ bv_5(0))\ \ app(listn(k+1, \mathcal{F}), (d_0\ \ldots\ d_{7-(k+1)}))\ \ bv_3(k+1)\ \ equal(d_{7-k}, x)).$$

It follows from our inductive hypothesis that

$$(sv_{m+1+18(k+1)}\ \ldots\ sv_n) = cells(equal(d_{7-k}, x), 5, 13, (d_{7-(k+1)}\ \ldots\ d_0)),$$

and hence $(sv_{m+1+18k}\ \ldots\ sv_n)$ is

$$\begin{aligned}&app(cell(x, 5, 13, d_{7-k}), cells(equal(d_{7-k}, x), 5, 13, (d_{7-(k+1)}\ \ldots\ d_0)) \\ &\quad = cells(x, 5, 13, (d_{7-k}\ \ldots\ d_0)). \ \Box\end{aligned}$$

5.6 The Receiver

Its action of the receiver is controlled by a submodule, `rcount`, which is defined as follows:

```
(STRUCT
  (CLK RST STOP START)
  (BIT)
  (cdff count5 or2 t0 f0 comp5)
  ((CLK RST STOP S1) (CLK RST STOP Q) (START Q)
   () () (T Q0 F Q1 F Q2 T Q3 F Q4))
  ((Q QN) (Q0 Q1 Q2 Q3 Q4) (S1) (T) (F) (BIT)))
```

The functionality of `rcount` is similar to that of `scount`. A state is again a list $(on\ cnt)$ of two components, corresponding to the two sequential submodules, `cdff` and `count5`. As long as both data inputs are $\mathcal{F}$, the value of on remains constant. While $on = \mathcal{T}$, cnt is incremented repreatedly; while $on = \mathcal{F}$, cnt remains unchanged. If `STOP` is $\mathcal{T}$, then on and cnt are reset to $\mathcal{F}$ and $bv_5(0)$; otherwise, if `START` is $\mathcal{T}$, then on is set to $\mathcal{T}$. The output value is determined by comparing cnt with $bv_5(9)$:

Proposition 5.10 *Let* $\Sigma = (on\ cnt)$ *and* $V = (stop\ start)$ *be a state and a data vector for* `rcount`. *Then*

$$nv(\texttt{BIT}, V, \Sigma, \texttt{rcount}) = \begin{cases} \mathcal{T} & \textit{if } cnt = bv_5(9) \\ \mathcal{F} & \textit{if } cnt \neq bv_5(9); \end{cases}$$

$$next(V, \Sigma, \texttt{rcount}) = \begin{cases} (\mathcal{F}\ bv_5(0)) & \textit{if } stop = \mathcal{T} \\ (\mathcal{T}\ inc(cnt)) & \textit{if } stop = \mathcal{F} \textit{ and } start = on = \mathcal{T} \\ (\mathcal{T}\ cnt) & \textit{if } stop = \mathcal{F} \textit{ and } start = \mathcal{T} \textit{ and } on = \mathcal{F} \\ (\mathcal{T}\ inc(cnt)) & \textit{if } stop = \mathcal{F} \textit{ and } start = \mathcal{F} \textit{ and } on = \mathcal{T} \\ (\mathcal{T}\ cnt) & \textit{if } stop = start = on = \mathcal{F}. \end{cases}$$

The definition of `rcvr` is as follows:

```
(STRUCT
  (CLK RST SIN)
  (O0 O1 O2 O3 O4 O5 O6 O7 DONE)
  (rcount edff count3 shift8 dff not1 not1 xor2 and4 f0)
  ((CLK RST BIT N2) (CLK RST BIT N1)
   (CLK RST BIT) (CLK RST F BIT X F F F F F F F F)
   (CLK RST A) (SIN) (X) (SIN Q) (Q0 Q1 Q2 BIT) ())
  ((BIT) (Q QN) (Q0 Q1 Q2) (O0 O1 O2 O3 O4 O5 O6 O7)
   (DONE DONEN) (N1) (N2) (X) (A) (F)))
```

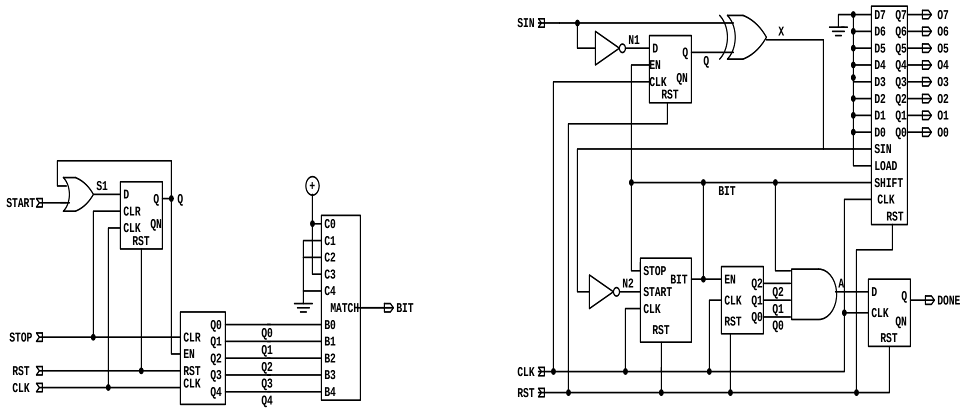


Figure 10: (a) `rcount` (b) `rcvr`

Like `sndr`, `rcvr` has two modes of operation. In the first mode, it waits for an edge, i.e., a change in input. This is detected by comparing the input with the state of the flip-flop `edff`, which is the negation of the most recently read value. In this mode, the controller `rcount` is turned off. When an edge is detected, `rcount` is turned on and its counter is reset:

Proposition 5.11 *Let* $V = (sin)$ *be a data vector for* `rcvr`*, and let* $\Sigma = (\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5)$ *be a state of* `rcvr`*, where* $\sigma_1 = (on\ cnt)$*. Assume that* $on = \mathcal{F}$*,* $cnt = bv_5(0)$*, and* $\sigma_5 = \mathcal{F}$*. Let* $\Sigma' = next(V, \Sigma, \texttt{rcvr})$*.*

(a) If $sin = \sigma_2$*, then* $\Sigma' = ((\mathcal{T}\ bv_5(0))\ \sigma_2\ \sigma_3\ \sigma_4\ \mathcal{F})$*;*

(b) If $sin \neq \sigma_2$*, then* $\Sigma' = \Sigma$*.*

In its second mode, the receiver counts until it reaches the input bit to be sampled. At this point, the appropriate value is shifted into the register `shift8`, the bit counter `count3` is incremented, the current input value is stored in `edff`, and `rcount` is turned off. When the eighth bit has been computed, the state of `dff` is altered to indicate termination:

Proposition 5.12 *Let* $V = (sin)$ *be a data vector for* `rcvr`*, and let* $\Sigma = (\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5)$ *be a state of* `rcvr`*, where* $\sigma_1 = (on\ cnt)$*. Assume that* $on = \mathcal{T}$ *and* $\sigma_5 = \mathcal{F}$*. Let* $\Sigma' = next(V, \Sigma, \texttt{rcvr})$*.*

(a) If $cnt = bv_5(9)$ *and* $\sigma_3 = bv_3(7)$*, then*

$$\Sigma' = ((\mathcal{F}\ bv_5(0))\ not1(sin)\ bv_3(0))\ shift(xor2(\sigma_2, sin), \sigma_4)\ \mathcal{T});$$

(b) If $cnt = bv_5(9)$ and $\sigma_3 \neq bv_3(7)$, then

$$\Sigma' = ((\mathcal{F}\ bv_5(0))\ \ not1(sin)\ \ inc(\sigma_3)\ \ shift(xor2(\sigma_2, sin), \sigma_4)\ \ \mathcal{F});$$

(c) If $cnt \neq bv_5(9)$, then $\Sigma' = ((\mathcal{T}\ inc(cnt))\ \ \sigma_2\ \ \sigma_3\ \ \sigma_4\ \ \mathcal{F})$.

The specification of `rcvr` is given by the following lemma. For its proof, we require the following definition: If L and M are two bit vectors, then

$$push(L, M) = \begin{cases} M & \text{if } L = \texttt{NIL} \\ push(cdr(L), shift(car(L), M)) & \text{if } L \neq \texttt{NIL}. \end{cases}$$

Thus, if $L = (x_1\ \ldots\ x_\ell)$ and $M = (y_1\ \ldots\ y_m)$, where $\ell \leq m$, then

$$push(L, M) = (x_\ell\ \ldots\ x_1\ y_1\ \ldots\ y_{m-\ell}).$$

Proposition 5.13 *Let $\mathcal{V} = (V)$, where V is a bit vector of length n. Assume that $length(recv(8, \mathcal{T}, 10, V)) = 8$. Then for some m, $1 \leq m \leq n$,*

$$sv(j, \texttt{DONE}, \mathcal{V}, \texttt{rcvr}) = \begin{cases} \mathcal{T} & \textit{if } j = m \\ \mathcal{F} & \textit{if } j < m. \end{cases}$$

For $i = 1, \ldots, 7$, let $d_i = sv(m, \texttt{O}i, \mathcal{V}, \texttt{rcvr})$. Then $(d_7\ \ldots\ d_0) = recv(8, \mathcal{T}, 10, V)$.

Proof: Let $V = (v_1\ \ldots\ v_n)$. For $j = 0, \ldots, n$, let $V_j = (v_{j+1}\ \ldots\ v_n)$ and

$$\Sigma_j = state(j, \mathcal{V}, \texttt{rcvr}) = ((on_j\ cnt_j)\ \ flg_j\ \ bits_j\ \ reg_j\ \ done_j).$$

We shall prove the following generalization of the desired result:
Suppose that for some j, $on_j = \mathcal{F}$, $cnt_j = bv_5(0)$, $done_i = \mathcal{F}$ for all $i \leq j$, and

$$length(recv(8 - b, not1(flg_j), 10, V_j)) = 8 - b,$$

where $bits_j = bv_3(b)$. Then for some $m > j$, $done_i = \mathcal{F}$ for all $i < m$, $done_m = \mathcal{T}$, and

$$reg_m = push(recv(8 - b, not1(flg_j), 10, V_j), reg_j).$$

The proposition will then follow from the case $j = 0$.

First note that according to our assumption, $recv(8 - b, not1(flg_j), 10, V_j) \neq \texttt{NIL}$, and hence, $scan(not1(flg_j), V_j) = V_k$ for some k, $j \leq k < n - 10$. Thus, $v_i = not1(flg_j)$ for $i = j + 1, \ldots, k$, and $v_{k+1} = flg_j$. From the definition of $recv$, we have

$$recv(8-b, not1(flg_j), 10, V_j) = cons(xor2(flg_j, v_{k+11}), recv(7-k, v_{k+11}, 10, V_{k+11})),$$

and hence,

$$length(recv(7-b, v_{k+11}, 10, V_{k+11})) = 7-b.$$

By Proposition 5.11, $\Sigma_i = \Sigma_j$ for $i = j, \ldots, k$, and

$$\Sigma_{k+1} = ((\mathcal{T}\ bv_5(0))\ \ flg_j\ \ bits_j\ \ reg_j\ \ \mathcal{F}).$$

By Proposition 5.12(c), for $i = 0, \ldots, 9$,

$$\Sigma_{k+1+i} = ((\mathcal{T}\ bv_5(i))\ \ flg_j\ \ bits_j\ \ reg_j\ \ \mathcal{F}).$$

The proof is by induction on $7-b$. Consider first the base case, $b = 7$. By Proposition 5.12(a),

$$\Sigma_{k+11} = ((\mathcal{F}\ bv_5(0))\ \ not1(v_{k+11})\ \ bv_3(0)\ \ shift(xor2(flg_j, v_{k+11}), reg_j)\ \ \mathcal{T}).$$

Here, the result holds for $m = k+11$, since

$$\begin{aligned} push(recv(8-b, not1(flg_j), 10, V_j), reg_j) &= push((xor2(flg_j, v_{k+11})), reg_j) \\ &= shift(xor2(flg_j, v_{k+11}), reg_j). \end{aligned}$$

Now suppose that $b < 7$, and assume that the claim holds with b replaced with $b+1$. By Proposition 5.12(a),

$$\Sigma_{k+11} = ((\mathcal{F}\ bv_5(0))\ \ not1(v_{k+11})\ \ bv_3(b+1)\ \ shift(xor2(flg_j, v_{k+11}), reg_j)\ \ \mathcal{F}).$$

We may conclude that for some $m > k+11$, $done_i = \mathcal{F}$ for all $i < m$, $done_m = \mathcal{T}$, and

$$\begin{aligned} reg_m &= push(recv(7-b, v_{k+11}, 10, V_{k+11}), shift(xor2(flg_j, v_{k+11}), reg_j)) \\ &= push(cons(xor2(flg_j, v_{k+11}), recv(7-b, v_{k+11}, 10, V_{k+11})), reg_j) \\ &= push(recv(8-b, not1(flg_j), 10, V_j), reg_j). \ \Box \end{aligned}$$

5.7 The Main Theorem

Finally, we present our main result concerning the circuit `bpm`. We assume that the two clock input waveforms are admissible pulses for `sndr` and `rcvr`, respectively, with periods that conform to the constraints imposed by Moore's theorem, and that the other inputs are well-behaved with respect to the clocks, as required by Theorem 3.1. We also assume that the `SEND` input has the value $\mathcal{T}$ on exactly one cycle, during which an 8-bit message is read from the other data inputs. This message is then encoded and transmitted by `sndr`, and received, decoded, and output by `rcvr`. As stated in the theorem, the completion of this process is signalled by the output `DONE`: when its value first becomes $\mathcal{T}$, the other outputs display the decoded message.

Theorem 5.2 *Let* $p_{in} = (w_{\text{CLKS}}\, w_{\text{RSTS}}\, w_{\text{CLKR}}\, w_{\text{RSTR}}\, w_{\text{SEND}}\, w_0 \dots w_7)$ *be an input packet for* `bpm`, *where*

(a) $(\text{c}LKS\, w_{\text{RSTS}}\, w_{\text{SEND}}\, w_0 \dots w_7)$ *is an admissible* n_s*-cycle input packet for* `sndr` *based at* b_s *with value matrix* $\mathcal{V}_s = (V_{\text{SEND}}\, V_{\text{I0}} \dots V_{\text{I7}})$ *and period* π_s*;*

(b) w_{CLKR} *is an admissible* $(n_r + 2)$*-cycle pulse for* `rcvr` *based at* b_r *with high* $h > 7000$*, low* $\ell > 7000 + setup(\texttt{SIN}, \texttt{rcvr})$*, and period* $\pi_r = h + \ell$*;*

(c) w_{RSTR} *is an admissible* $(n_r + 1)$*-cycle reset waveform for* `rcvr` *based at* b_r *with period* π_r*.*

Assume $17\pi_r \le 18\pi_s \le 19\pi_r$*. Suppose that for some* m_s*,* $1 \le m_s \le n_s - 144$*,*

$$nth(j, V_{\text{SEND}}) = \begin{cases} \mathcal{T} & \text{if } j = m_s \\ \mathcal{F} & \text{if } j \ne m_s,\ 1 \le j \le n_s; \end{cases}$$

For $i = 0, \dots, 7$*, let* $d_i = nth(m_s, V_{\text{I}i})$*. Let* $t_r = b_r + \pi_r$*. Assume that* $b_s + 2\pi_s \le t_r \le b_s + (m_s + 2)\pi_s$ *and* $b_s + (n_s + 2)\pi_s \le t_r + n_r\pi_r$*.*

Let $p_{out} = outp(\texttt{bpm}, sim(\texttt{bpm}, p_{in}, t_f))$*, where* $t_f \ge t_r + n_r\pi_r$*. Then* p_{out} *is a stable* n_r*-cycle packet based at* $t_r + \pi_r$ *with value matrix* $\mathcal{V}_r$ *and period* π_r*, for some* $\mathcal{V}_r = (V_{\text{DONE}}\, V_{\text{O0}} \dots V_{\text{O7}})$*. For some* m_r*,* $1 \le m_r \le n_r$*,*

$$nth(j, V_{\text{DONE}}) = \begin{cases} \mathcal{T} & \text{if } j = m_r \\ \mathcal{F} & \text{if } j \ne m_r,\ 1 \le j \le n_r, \end{cases}$$

and for $i = 0, \dots, 7$*,* $nth(m_r, V_{\text{O}i}) = d_i$*.*

Proof: We may assume, without loss of generality, that $n_s = m_s + 144$. For $j = 0, \dots, n_s$, let $sv_j = sv(j, \texttt{SOUT}, \mathcal{V}_s, \texttt{sndr})$. By Proposition 5.9,

$$(sv_1 \dots sv_{n_s}) = send((d_7 \dots d_0), m_s, 5, 13, 0).$$

Since $sv_0 = \mathcal{T}$, we have $sv_j = \mathcal{T}$ for all $j \le m_s$.

Fix j so that $b_s + j\pi_s \le t_r < b_s + (j+1)\pi_s$ and let $t_s = b_s + j\pi_s$. Then $2 \le j \le m_s + 2$, and hence $sv_{j-2} = \mathcal{T}$. Let

$$S = (sv_{j-1} \dots sv_{n_s}) = send((d_7 \dots d_0), m_s - j + 2, 5, 13, 0)$$

and let w_{LOUT} be the waveform for LOUT determined by $sim(\texttt{bpm}, p, t_f)$. By Theorem 4.1, $(w_{\text{CLKR}}\, w_{\text{RSTR}}\, w_{\text{LOUT}})$ is an admissible input packet for `rcvr` based at b_r with value matrix (A) and period π_r, where $A = asynch(U, t_s, t_r, \pi_s, \pi_r, oracle)$ for some bit vector $oracle$.

Let $\mathcal{V}_r = (V_{\text{DONE}}\, V_{\text{O0}} \ldots V_{\text{O7}})$, where

$$V_{\text{DONE}} = (sv(1, \texttt{DONE}, (A), \texttt{rcvr}) \ldots sv(n_r, \texttt{DONE}, (A), \texttt{rcvr}))$$

and for $i = 0, \ldots, 7$,

$$V_{\text{O}i} = (sv(1, \texttt{O}i, (A), \texttt{rcvr}) \ldots sv(n_r, \texttt{O}i, (A), \texttt{rcvr})).$$

By Theorem 3.1, p_{out} is a stable n_r-cycle packet based at $b_r + \pi_r + \pi_r = t_r + \pi_r$ with value matrix $\mathcal{V}_r$ and period π_r.

According to Moore's Theorem, $recv(8, \mathcal{T}, 10, A) = (d_7 \ldots d_0)$. But then, by Proposition 5.13, there exists m_r such that $1 \leq m_r \leq n_r$,

$$nth(j, V_{\text{DONE}}) = \begin{cases} \mathcal{T} & \text{if } j = m_r \\ \mathcal{F} & \text{if } j \neq m_r,\ 1 \leq j \leq n_r, \end{cases}$$

and

$$(nth(m_r, V_{\text{O7}}) \ldots nth(m_r, V_{\text{O0}})) = (d_7 \ldots d_0).$$

Thus, for $i = 0, \ldots, 7$, $nth(m_r, V_{\text{O}i}) = d_i$. $\Box$

References

[1] Boyer, R. S. and Moore, J S., *A Computational Logic Handbook*, Academic Press, Boston, 1988.

[2] Institute of Electrical and Electronic Engineers, *Draft Standard VHDL Language Reference Manual*, 1993.

[3] Kaufmann, M., *A Translator from an HDL of David Russinoff to VHDL*, Internal Note 278, Computational Logic, Inc., July 1993.

[4] Moore, J S., "A Formal model of asynchronous communication and its use in mechanically verifying a biphase mark protocol", *Formal Aspects of Computing* 6, no. 1 (1994):60-91.

[5] Roden, M. S., *Digital Communication Systems Design*, Prentice-Hall, 1988.

[6] Russinoff, D. M., *A Formalization of a Subset of VHDL*, Technical Report 98, Computational Logic, Inc., April, 1994.

[7] Taub, H. and Schilling, D., *Digital Integrated Electronics*, McGraw-Hill, New York, 1977.