Under consideration for publication in Formal Aspects of Computing

Automated Verification and Refinement for Physical-Layer Protocols

Geoffrey M. Brown[1] and Lee Pike[2]

[1]Indiana University, Bloomington,
[2]Galois Inc.

Abstract. This paper demonstrates how to use a *satisfiability modulo theories* (SMT) solver together with a bounded model checker to verify properties of real-time physical layer protocols. The method is first used to verify the Biphase Mark protocol, a protocol that has been verified numerous times previously, allowing for a comparison of results. The techniques are extended to the 8N1 protocol used in universal asynchronous receiver transmitters (UARTs). We then demonstrate the use of temporal refinement to link a finite state specification of 8N1 with its real-time implementation. This refinement relationship relieves a significant disadvantage of SMT approaches—their inability to scale to large problems. Finally, capturing the impact of metastability on timing requirements is a key issue in modeling physical-layer protocols. Rather than model metastability directly, a contribution of our models is treating its effect as a constraint on non-determinism.

Keywords: Satisfiability modulo theories (SMT); Infinite-state model checking; real-time; physical layer; Biphase Mark; UART

1. Introduction

A recently-developed formal verification technique combines a *satisfiability modulo theories* (SMT) solver and a bounded model checker to prove LTL safety properties over infinite-state transition systems. The proof technique implemented is *k-induction*, a generalization of induction over (infinite-state) transition systems; for brevity, we will call the technique *infinite-bmc induction* (for *inf*inite-state *b*ounded *m*odel *c*hecking *induction*) [dMRS03, Rus06]. One implementation of infinite-bmc induction is in SRI International's *Symbolic*

Correspondence and offprint requests to: `geobrown@cs.indiana.edu`, `leepike@gmail.com`
Based on material originally published in [BP06] and on "Temporal Refinement Using SMT and Model Checking with an Application to Physical-Layer Protocols" by Brown and Pike which appears in "The Proceedings of the Fifth ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'07)" ©2007 IEEE. [BP07]

Analysis Laboratory (SAL) [dMOR$^+$04]. In this paper, we apply infinite-bmc induction to easily prove the correctness of a class of real-time protocols.[1]

The first example considered in this paper is the Biphase Mark protocol (BMP) used in CD-player decoders, Ethernet, and Tokenring [BP06]. To motivate why infinite-bmc induction is of interest in real-time verification, consider that the verification of BMP presented herein results in an orders-of-magnitude reduction in effort as compared to the protocol's previous formal verifications using mechanical theorem-proving. Our verification required 3 invariants, whereas a published proof using the mechanical theorem-prover PVS required 37 [VdG04]. Using infinite-bmc induction, proofs of the 3 invariants were completely automated, whereas the PVS proof initially required some 4000 user-supplied proof directives, in total. Another proof using PVS is so large that the tool required 5 hours just to *check* the manually-generated proof whereas the SAL proof is generated automatically in seconds [Hun98]. BMP has also be verified by J. Moore using Nqthm, a precursor to ACL2, requiring a substantial proof effort (Moore cites the work as being one of the "best ideas" of his career) [Moo94].[2]

We extend the techniques used to verify BMP to the 8N1 protocol used in UARTs, and through that proof, we uncover errors in an industry application note used to help engineers select appropriate timing constraints for UARTs [Max03]. The 8N1 protocol also provides an opportunity to address a significant limitation of SMT based bounded model-checking—the technique has difficulty scaling to large systems.

Traditionally in model-checking, a finite-state abstraction of a real-time protocol is used to model the passage of time with an asynchronous interleaving semantics. The benefit of such an abstraction is that if the finite-state model is not too large, it can be verified automatically using standard model checking techniques (e.g., BDDs). Furthermore, a finite-state model of the protocol can be composed with finite-state models of synchronous hardware, and the composition may also be automatically model-checked. However, to ensure that the safety properties proved of the finite-state model hold of the infinite-state one, in which the progress of time is modeled with better fidelity, one must complete a refinement proof from the former to the latter. Describing how to complete a highly-automated refinement proof is a central result of this article.

We describe a simple refinement approach, derived from Abadi and Lamport's refinement method [AL91]. The approach combines the ease of finite-state verification with the fidelity of an infinite-state real-time model. (We emphasize that the approach described is one to refine *temporal* constraints and does not address other refinements such as data refinement.) We demonstrate how a nondeterministic specification can be formally refined into a more deterministic implementation. The refinement ensures that the safety properties that hold of the finite-state model also hold of the infinite-state model. As a case-study, we demonstrate the approach by verifying a refinement relationship between a finite-state model of the 8N1 protocol and an implementation that explicitly captures real-time constraints by linear inequalities over the real numbers (in the associated SAL files, BMP is likewise refined). To prove the correctness of the protocol, we use BDDs. To prove the refinement holds, we use infinite-bmc induction. Besides a method of temporal refinement, this paper introduces a succinct and general constraint-based model of physical-layer protocols that simplifies the proofs and decomposes the model of the environment (i.e., the effects of metastability) and the protocol specification. We believe this approach to modeling metastability is novel, although it is related to the work of Seshia, Bryant, and Stevens [SBS05].

The remainder of the paper is organized as follows. In Section 2, we provide the requisite background and describe the SAL model checker. In Section 3, we introduce the modeling techniques used for the BMP and 8N1 protocols which are presented in Sections 4 and 5, respectively. We develop a finite-state specification for 8N1, apply this specification to a system design incorporating transmitter and receiver shift-registers, and verify the refinement between real-time implementation and specification in Section 6. We conclude with a discussion and "lessons-learned" in Section 7.

[1] The specifications and proofs in SAL associated with this paper are available at `http://www.cs.indiana.edu/~lepike/pub_pages/bmp-jrnl.html`. SAL can be obtained at `http://fm.csl.sri.com/`.
[2] `http://www.cs.utexas.edu/users/moore/best-ideas/`.

2. Background and SAL

2.1. Background

SAL is not specifically a real-time model checker; time is modeled using the real numbers, and real-time constraints are captured as formulas (usually linear inequalities) over the reals. Partially-parameterized verifications of BMP are reported using the real-time model checkers Uppaal [HRSV01] and Hytech [HPWT01]. Real-Time model checkers are automated; in infinite-bmc induction provers, while each proof *attempt* is automated, the user iteratively builds up invariants (by refinement from automatically-generated counterexamples on failed proof attempts) to ensure a proof succeeds. (Reducing the complexity of such invariants for mixed finite-state and infinite-state specifications is a central motivation for the refinement approach presented in this paper.) With respect to expressiveness, the theory that real-time model checkers decide is weaker than the theory decided by contemporary SMT solvers. For example, SRI's Yices is a SMT solver for the satisfiability of (possibly quantified) formulas containing uninterpreted functions, real and integer linear arithmetic, arrays, fixed-size bit-vectors, recursive datatypes, tuples, records, and lambda expressions [DdM06].

The first infinite-bmc induction verification of a real-time protocol was Dutertre and Sorea's verification of the startup protocol for the Time-Triggered Architecture (a fault-tolerant bus) [DS04]. Subsequently, the reintegration protocol for NASA Langley's SPIDER (another fault-tolerant bus) was verified [PJ05].

2.2. The Symbolic Analysis Laboratory (SAL)

SAL has a high-level modeling language for specifying transition systems. A transition system is specified by a *module*. A module consists of a set of state variables (declared to be *input*, *output*, or *local*) and guarded transitions. A transition is *enabled* if its guard is true. Within a module, transitions are composed with the `[]` operator, and of the enabled transitions in a state, one is nondeterministically executed. When a transition is exercised, the next-state values are assigned to variables; for example, in the guarded transition, `G --> a' = a - 1; b' = a`, if the guard `G` holds and the transition is exercised, then in the next state, the variable `a` is decremented by 1 and the variable `b` is updated to the previous value of `a`. In the language of SAL, "`;`" denotes statement separation, not sequential composition (thus, variable assignments can be written in any order); furthermore, in a variable assignment, the next-state variable values of other variables can be referenced. If no variables are updated in a transition (i.e., `G -->`), the state idles.

Modules can be composed both synchronously (`||`) and asynchronously (`[]`)—note `[]` is overloaded—and composed modules communicate via shared variables. In a synchronous composition, a transition from each module is simultaneously applied; a synchronous composition is deadlocked if either module has no enabled transition. In an asynchronous composition, an enabled transition from exactly one of the modules is nondeterministically applied.

The language is typed, and predicate sub-types can be declared. (At the time of writing, SAL does not generate type-correctness conditions to ensure the correctness of sub-types.) Types can be both interpreted and uninterpreted, and base types include the reals, naturals, and booleans; array types, inductive datatypes, and tuple types can be defined. Both interpreted and uninterpreted constants and functions can be specified. Parameterized values are represented as uninterpreted constants from some parameterized type.

Bounded model checkers have historically been used to find counterexamples, but they can also be used to prove invariants by induction over the state space [dMOR$^+$04]. SAL supports *k-induction*, a generalization of the induction principle, which can prove some invariants that may not be strictly inductive. By incorporating a SMT solver, SAL can do k-induction proofs over infinite-state transition systems.

Let $(S, I, \rightarrow)$ be a transition system where S is a set of states, $I \subseteq S$ is a set of initial states, and $\rightarrow$ is a binary transition relation. If k is a natural number, then a *k-trajectory* is a sequence of states $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$ (a 0-trajectory is a single state). Let k be a natural number, and let P be a property. The k-induction principle is then defined as follows:

- *Base Case*: Show that for each k-trajectory $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$ such that $s_0 \in I$, $P(s_j)$ holds, for $0 \leq j < k$.
- *Induction Step*: Show that for all k-trajectories $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$, if $P(s_j)$ holds for $0 \leq j < k$, then $P(s_k)$ holds.

The principle is equivalent to the usual transition-system induction principle when $k = 1$. In SAL, the user

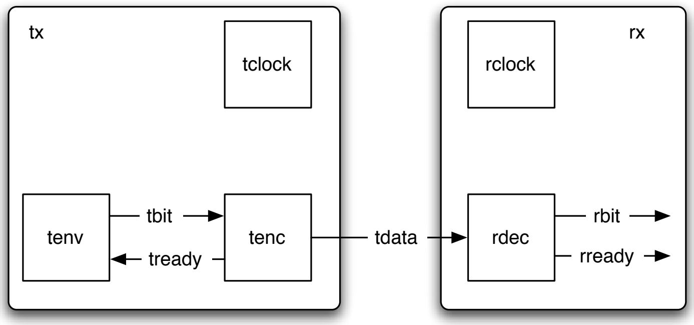

Fig. 1. System Block Diagram

specifies the depth at which to attempt an induction proof, but the attempt itself is automated. The main mode of user-guidance in the proof process is in iteratively building up *disjunctive invariants* [Rus00] by weakening an invariant based on the counterexamples returned by SAL in a failed proof attempt [BP06]. The invariant is weakened by adding an additional *disjunct* representing a configuration of the system not covered by the original invariant. This is in contrast to the traditional approach of strengthening an invariant by adding a *conjunct*. The benefit of disjunctive invariants is that each disjunct need only cover a special case of the system while each conjunct needs to hold in every configuration of the system's state.

Finally, SAL also has finite-state BDD-based and bounded model checkers as well as other tools such as a finite-state deadlock checker and simulator.

3. Protocol Modeling

All of the models discussed in this paper, including the finite-state and real-time specifications of 8N1 and BMP, are defined using the structure illustrated in Figure 1. For each protocol, there is a transmitter (`tx`) and a receiver (`rx`), each of which contains subcomponents. The transmitter contains an encoder (`tenc`) that encodes data according to the protocol, an environment (`tenv`) that supplies data, and a local clock (`tclock`). The receiver contains a decoder (`rdec`) that decodes the data according to the protocol and a local clock (`rclock`). The transmitter produces data on the boolean signal `tdata` which is consumed by the receiver.

Each block is modeled as a module in SAL, and the transmitter's modules are synchronously composed with each other, as are the receiver's modules. The entire system is the asynchronous composition of the transmitter and receiver, composed synchronously with a constraint module used to model metastability (illustrated in Figure 4 and described later in this section). While all the protocols have this structure, specific instances use different names. For example, the transmitter clock in Figure 3 is named `tclock_rt` while we call it `tclock` here. The name differences are intended to help disambiguate our discussion.

```
tx : MODULE = tclock || tenc || tenv;
rx : MODULE = rclock || rdec;
system : MODULE = (tx [] rx) || constraint;
```

The protocols described in this paper use a common `tenv` module (Figure 2)—its role is to generate a random bit (`tbit`) whenever requested by the encoder module (i.e., `tready = TRUE`). The encoder is protocol-specific and defined in the subsequent sections. We initialize the `tbit` to be `TRUE`. This initialization is necessary for the BMP verification. The nature of the protocol, as described in detail in Section 4, assumes an initial synchrony between the transmitter and receiver (the receiver is also initialized so that `rbit = TRUE`; see Figure 7). The 8N1 protocol includes an initial synchronization mechanism, so the initialization assumption is not necessary to prove the correctness of this protocol.

In the real-time protocols, the clocks are modeled as *timeout automata* in which the progress of time is enforced cooperatively by (possibly nondeterministically) updating variables called *timeouts* over the real numbers [DS04]. Thus, we begin by defining a type synonym:

```
TIME : TYPE = REAL;
```

```
tenv : MODULE =
BEGIN
  INPUT tready : BOOLEAN
  OUTPUT tbit  : BOOLEAN
  INITIALIZATION
    tbit = TRUE;
  TRANSITION
  [
      tready --> tbit' IN {TRUE, FALSE}
   [] else   -->
  ]
END;
```

Fig. 2. Transmitter's Encoder

```
time(t1 : TIME, t2: TIME): TIME = IF t1 <= t2 THEN t1 ELSE t2 ENDIF;

TPERIOD : {x : TIME | 0 < x};
TSETTLE : {x : TIME | 0 <= x  AND x < TPERIOD};
TSTABLE : TIME =  TPERIOD - TSETTLE;

tclock_rt : MODULE =
BEGIN
  INPUT  rclk   : TIME
  OUTPUT tclk   : TIME
  INITIALIZATION
    tclk  IN {x : TIME | 0 <= x AND x <= TPERIOD};
  TRANSITION
  [
    tclk = time(tclk, rclk) --> tclk' = tclk + TPERIOD;
  ]
END;
```

Fig. 3. Transmitter Clock (Real-Time)

The timeouts of the the transmitter and receiver are `tclk` and `rclk`, respectively. The "current" time is defined as the minimum of these two values, and the transmitter and receiver are each allowed to execute only when their timeout is the current time. As we shall see, "time" advances in steps of non-deterministic size bounded by a set of linear constraints.

This use of timeouts seems counter-intuitive. A typical implementation would use a finite counter that increments up to a desired timeout value at a specified rate and is then reset to zero. In our use of timeouts one can view the difference between the current time and the timeout variable as the "value" of such a counter. This abstraction allows us to ignore details such as the accuracy and frequency of the clock that increments such a counter. An open exercise in proof refinement is to (separately) validate a counter implementation and substitute it for the abstract timeout mechanism that we use.

We model the transmitter clocks as running at a fixed but arbitrary rate, while all of the frequency errors are captured in the receiver clock relative to the transmitter clock. This modeling simplification does not lead to a loss of generality or fidelity. The real-time transmitter clock periods consist of a settling phase (`TSETTLE`) and a stable phase (`TSTABLE`) as illustrated in Figure 3. The settling phase captures both the (allowance for) propagation delay as well as setup requirements at the receiver (discussed below). `TSETTLE` and `TSTABLE` are uninterpreted parameterized constants, which allows us to verify the model for any combination of settling

```
constraint : MODULE =
BEGIN
  INPUT  tclk     : TIME
  INPUT  rclk     : TIME
  INPUT  rdata    : BOOLEAN
  INPUT  tdata    : BOOLEAN
  LOCAL  stable   : BOOLEAN
  LOCAL  changing : BOOLEAN
  DEFINITION
    stable = NOT changing OR (tclk - rclk < TSTABLE);
  INITIALIZATION
    changing = FALSE
  TRANSITION
  [
      rclk' /= rclk AND (stable => rdata' = tdata) -->
   [] tclk' /= tclk --> changing' = (tdata' /= tdata)
  ]
END;
```

Fig. 4. Constraint Module (Real-Time)

time and receiver clock error. The transmitter settling time can be used to capture the effects of jitter and dispersion in data transmission as well as jitter in the transmitter's clock. In the case of the settling period, the model can be viewed as less deterministic than an actual implementation which might reach stable transmission values sooner. The receiver clocks, which may run at multiple rates, are protocol dependent and are discussed along with the corresponding protocol.

We have initialized `tclk` in the range 0 to `TPERIOD` with similar constraints on the initial value of `rclk` which means our proof of BMP is valid if the transmitter and receiver start in relative synchrony. In practice, BMP and similar protocols rely upon the use "training sequences" to bring the transmitter and receiver into initial synchrony. A proof of these training procedures is beyond the scope of this paper.

The key to correct behavior of the the real-time protocols is the synchronization imposed by the clocks, which respectively constrain when the transmitter changes the value of `tdata` and when the receiver attempts to read the value of `tdata`. If the receiver reads too frequently, then transmitted data will be duplicated; if it reads too infrequently, transmitted data will be lost; and if it reads at the wrong moment, the received data may have random errors due to metastability.

Metastability in the receiver is a major implementation consideration. Briefly, metastable behavior may occur at a latch when the input changes too soon before (a setup time violation) or after (a hold time violation) the latching event. In the case of the protocols described in this paper, the receiver implementation must latch `tdata` using its local clock `rclk` while the input data may change relative to `tclk`.

We model the effect of metastability rather than the physical process. We assume, as do most implementors, that metastable behavior can be contained to the latch where it occurs and that the net effect of metastability is a random bit at the output of the latch. As a simplification, we limit our attention to setup time violations; it is feasible to design a latch with zero hold time (by inserting a delay), but not zero setup time.

Rather than build a complex behavioral model of a latch subject to metastable behavior, we separate the problem into two parts—a non-deterministic latch which sets its output to a random value, and a constraint module which forces the latch to copy its input to its output in those cases where the timing constraints have been satisfied. In each of our decoders, the transmitted signal `tdata` is "latched" by `rdata`; however, in our model, the value assigned to `rdata` (Figures 7 and 13) is random but constrained to be `tdata` by the corresponding constraint module whenever the timing constraints are met.

This constraint module, illustrated in Figure 4, has two transitions. The first holds when the receiver transitions and restricts the latch behavior as described above (`=>` is implication). The second holds when the transmitter transitions and records whether the transmitter changed the data. Thus, we capture the

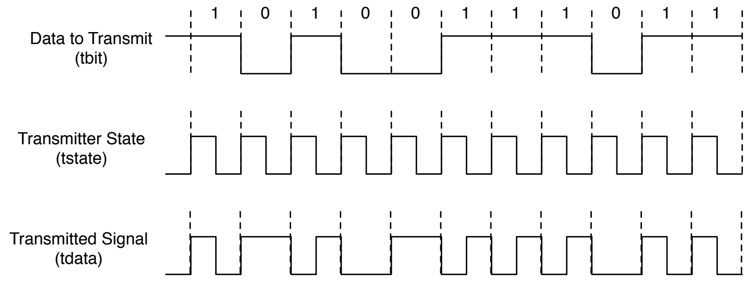

Fig. 5. Biphase Mark Protocol

setup time requirement by computing the time since the last transition on the input `tdata`. Note that the predicate `stable` is true if the input data (`tdata`) did not change on the last transmitter transition or if the receiver clock falls within the "stable" phase of the transmitter clock period. To capture a hold time requirement would require computing the time to the next possible transition on input `tdata`. This requires knowledge of the possible next states of the source of `tdata`, which is equivalent to knowing the state of `tready`. As previously mentioned, we chose not to implement this more complex constraint because the hold time can be subsumed by the setup time in a suitably designed flipflop. It is important to note that this constraint module cannot cause deadlock because the actual assignment to `rdata` allows either true or false as a next value.

4. Biphase Mark Protocol

The Biphase Mark protocol is a simple serial encoding that combines clock and data in a single signal. To understand the basic protocol, consider Figure 5, where the top stream is the signal to be transmitted while the middle stream is a digital clock (transmitter state) that defines the boundaries between the individual bits. These two signals are combined in the bottom signal according to the BMP protocol. In BMP, every bit period (called a *cell*) is guaranteed to begin with a transition marking a clock event. The value of the bit is determined by the presence (to encode a 1) or absence (to encode a 0) of a transition in the middle of the bit period. Thus, 0's are encoded as the two symbols 00 or 11, while 1's are encoded as 01 or 10. To make the correspondence between our model and figure clearer, we have noted the model signals (tbit, tstate, and tdata) in parenthesis.

4.1. Model

Our encoder module, illustrated in Figure 6, models the transmitter's portion of the protocol. Both transmitter and receiver have two states:

```
STATE : TYPE = [0..1];
```

In state 0, the initial state, the transmitter toggles `tdata` while in state 1 the transmitter signals the value of the cell by selectively toggling `tdata`. Our model assumes that the two halves of a cell are of identical length—recall that the transmitter's progress of time is enforced by `tclock_rt` (Figure 3). The model is generalizable to support asymmetric cells with a slightly more complex `tclock_rt` module.[3]

Decoding a BMP signal requires detecting the transition at cell boundaries and the values of the two cell halves. One technique for detecting cell boundaries involves "scanning" the received signal at a frequency higher than the transmission frequency. The value of the second cell half is then determined by "sampling" after an estimated delay from the detected transition. The complicating factors are metastability, which

[3] Although Moore suggests that there are advantages to an asymmetric cell [Moo94], this is not generally done in practice because it alters the DC balance and raises the bandwidth of the signal.

```
tenc_bmp : MODULE =
BEGIN
  OUTPUT tdata       : BOOLEAN
  OUTPUT tstate      : STATE
  OUTPUT tready      : BOOLEAN
  INPUT  tbit        : BOOLEAN
  INITIALIZATION
    tstate = 0;
    tdata  = TRUE;
  DEFINITION
    tready = (tstate = 0);
  TRANSITION
  [
     tstate = 0 -->  tdata' = NOT tdata;
                     tstate' = 1;
   [] tstate = 1 --> tdata' = tbit XOR tdata;
                     tstate' = 0;
  ]
END;
```

Fig. 6. BMP Encoder

```
rdec_bmp  : MODULE =
BEGIN
  INPUT  tdata  : BOOLEAN
  OUTPUT rdata  : BOOLEAN
  OUTPUT rstate : [0..1]
  OUTPUT rbit   : BOOLEAN
  INITIALIZATION
    rstate = 0;
    rdata = TRUE;
    rbit = TRUE;
  TRANSITION
  [
      rstate = 0  --> rdata' IN {FALSE, TRUE}
                      rstate' = IF (rdata = rdata') THEN 0 ELSE 1 ENDIF;
   [] rstate = 1  --> rdata' IN {FALSE, TRUE};
                      rbit' = rdata /= rdata';
                      rstate' = 0;
  ]
END;
```

Fig. 7. BMP Decoder

adds uncertainty to the transition detection process, and clock errors, which limit the accuracy of delay estimation.

Logically, the BMP decoder, illustrated in Figure 7, is the dual of the encoder. In state 0, the decoder "scans" for a transition. In state 1, the decoder "samples" the second cell value to determine the received bit value (`rbit`). As described in Section 3, latching the input data `tdata` in `rdata` is a non-deterministic process which is "constrained" to select the correct value if the setup time requirement has been met.

```
timeout (min : TIME, max : TIME) : [TIME -> BOOLEAN]  =
  {x : TIME | min <= x AND x <=  max};

rclock_bmp_rt : MODULE  =
BEGIN
  INPUT  tclk     : TIME
  INPUT  rstate   : STATE
  OUTPUT rclk     : TIME
  INITIALIZATION
  rclk IN {x : TIME | 0 <= x AND x < RSCANMAX};
  TRANSITION
  [
   rclk <= tclk -->  rclk' IN IF (rstate' = 0)
                              THEN  timeout(rclk + RSCANMIN, rclk + RSCANMAX)
                              ELSE  timeout(rclk + RSAMPMIN, rclk + RSAMPMAX)
                              ENDIF;
  ]
END;
```

Fig. 8. BMP Receiver Clock

Most of the ingenuity in defining a correct implementation of BMP is in correctly defining the clocks. As described in Section 3, the transmitter clock is assumed to have a fixed frequency, while any errors due to jitter, frequency drift, and fixed frequency error are captured in the behavior of the receiver clock. The receiver clock, illustrated in Figure 8, has two basic rates depending upon the state of the decoder. When the decoder is in state 0, `rclock_bmp_rt` advances at a "scan" rate and when the decoder is in state 1, it advances at a "sample" rate. In most physical implementations, the "sample" period is simply a multiple of the "scan" period; however, our model allows greater flexibility. The range of timeout values while "scanning" is the closed real interval [`RSCANMIN, RSCANMAX`], and the range of timeout values while "sampling" is the real closed interval [`RSAMPMIN, RSAMPMAX`] (these uninterpreted constants will be constrained shortly). The *correctness* of the protocol depends directly upon the relationship between these ranges and the transmitter clock periods.

4.2. Verification

Our main goal is to prove that the Biphase decoder reliably extracts the data transmitted by the encoder. In a correct interleaving of the transmitter and receiver, they cycle through the states (0,0), (1,0), (1,1), (0,1), where the transmitter's state is the first element of the tuple and the receiver's state is the second. This sequence of states is illustrated in Figure 9 where the transmitter and receiver states (`tstate` and `rstate`, respectively) are superimposed upon example values for `tdata` – the "cell times" are denoted by dotted lines. When `tstate = 0`, the encoder is ready to begin transmitting a new cell. The `tenv` module (Figure 2) outputs `tbit`, which contains the value being transmitted in the current cell.

The main correctness theorem of BMP is expressible in the LTL temporal logic, where the `G` operator denotes that its argument holds in all states on a trajectory through the transition system. The theorem states that whenever a cell has been transmitted, the bit received is the bit sent.

```
BMP_Thm:  THEOREM system_bmp |- G((rstate = 0 AND tstate = 0) => (rbit = tbit));
```

For the state sequence described above to be maintained, and for correct data transmission to occur, the *only* system state in which the receiver latch should possibly exhibit metastability is when `rx` is scanning for an initial transition from `tx`.

Recall that the transmitter clock period consists of two phases, a "settling" phase (`TSETTLE`) and a "stable" phase (`TSTABLE`). The "settling" phase includes both variation in propagation delay and the receiver's

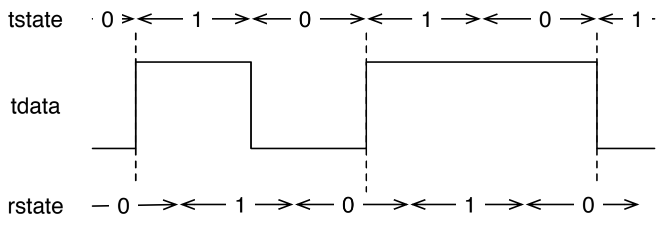

Fig. 9. Example State Sequence for BMP Protocol

setup requirement. For the receiver to reliably sample `tdata`, the transmitter clock must be in the "stable" phase. Furthermore, the receiver latch must correctly sample `tdata` at least once during state (1,0) to guarantee a transition to `rstate = 1`. These observations lead to the following constraints on the the "scan" time. (Recall again that constraints are formulated as uninterpreted constants from a parameterized type. Parameterized types are formulated using set-builder notation.)

```
RSCANMIN : {x : TIME | 0 < x};
RSCANMAX : {x : TIME | RSCANMIN <= x AND x < TSTABLE};
```

After the receiver transitions to `rstate = 1`, the timing constraints must guarantee that the next sample of `tdata` is correct. Hence, the the sampling time must fall in the "stable" transmitter clock phase of the second cell half (i.e. the period bounded below by `TPERIOD + TSETTLE` and above by `2*TPERIOD`). Given that the receiver may detect the initial transition as early as the beginning of the cell or as much as `TSETTLE + RSCANMAX` later, we derive the following constraints on the sampling clock period.

```
RSAMPMIN : {x : TIME | TPERIOD + TSETTLE < x};
RSAMPMAX : {x : TIME | RSAMPMIN <= x AND x + TSETTLE + RSCANMAX < 2*TPERIOD};
```

The main theorem is too challenging for SAL to validate directly in reasonable space and time, so supporting lemmas are required. When a k-induction proof attempt fails, two options are available to the user: the proof can be attempted at a greater depth, or supporting lemmas can be added to restrict the state-space. A k-induction proof attempt is automated, but if the attempt is not successful for a sufficiently small k (i.e., the attempt takes too long or too much memory), additional invariants are necessary to reduce the necessary proof depth. The user must formulate the supporting invariants manually, but their construction is facilitated by the counterexamples returned by SAL for failed proof attempts. If the property is indeed invariant, the counterexample is a trajectory that fails the induction step but lies outside the set of reachable states. The state-space considered can be constrained by auxiliary lemmas ruling out the counterexample.

We begin with an invariant that describes simple properties of the timeout variables. The invariant is inductive and hence can be proved by SAL at depth k=1.

```
l1 : LEMMA system_bmp |- G((tclk <= rclk + TPERIOD) AND (rclk <= tclk + RSAMPMAX));
```

The heart of the proof is an invariant that describes the relationship between the transmitter and receiver. We must relate them both temporally and with respect to their discrete state (e.g., `tstate` with `rstate` and `tdata` with `rdata`). The number of and the complexity of the supporting lemmas necessary to prove the main results is significantly reduced by proving a *disjunctive invariant* [Rus00] as mentioned in Section 2.2. A disjunctive invariant has the form $\bigvee_{i \in I} P_i$ where each P_i is a state predicate (predicates P_i and P_j need not be disjoint for $i \neq j$). Although disjunctive invariants represent a general proof technique, it is particularly easy to build a disjunctive invariant in SAL. The counterexamples SAL returns can be used to iteratively weaken the disjunction until it is invariant.

There are four disjuncts in the BMP invariant, illustrated in Figure 10. The disjuncts correspond to the four system states, the Cartesian product of two for the transmitter and two for the receiver, as illustrated in Figure 9. In addition to relating `tstate` and `rstate`, the invariant describes the relative difference between `tclk` and `rclk` as well as the "stability" of `tdata`. This invariant is proved at depth 4 using lemma `l1`. To give an intuition regarding how we build up the invariant, consider omitting the last disjunct from the invariant in Figure 10. If we attempt to prove `t0` with the final disjunct omitted, then SAL returns

```
t0:  THEOREM system_bmp |- G( ((rstate=0 AND tstate=0) AND
                                (rclk <= tclk + RSCANMAX) AND
                                (rdata = tdata) AND stable)
                            OR ((rstate=0 AND tstate=1) AND
                                (rclk < tclk) AND (rdata /= tdata))
                            OR ((rstate=1 AND tstate=1) AND
                                (tclk < rclk) AND (rdata = tdata))
                            OR ((rstate=1 AND tstate=0) AND
                                (rclk < tclk)));
```

Fig. 10. BMP Disjunctive Invariant

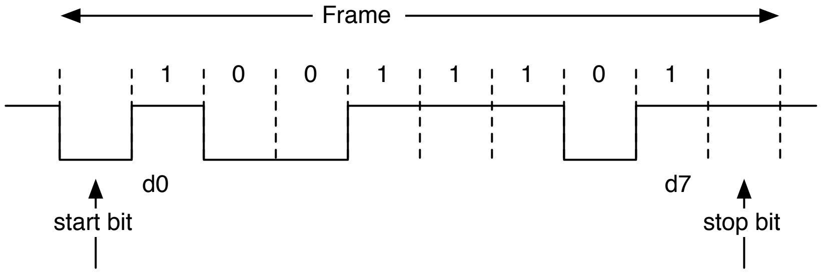

Fig. 11. 8N1 Data Transmission

a counterexample with three transitions in which the last system state contains the variable assignments `rclk = 9`, `rstate = 1`, `tclk = 12`, and `tstate = 0`, (among other assignments). From these assignments, we hypothesize a generalization of the relation between the clocks (`rclk < tclk`) and strengthen the invariant for the case in which `rstate = 1` and `tstate = 0`. Rerunning the proof confirms that the additional disjunct is sufficient to prove `t0`.

The main theorem is then proved at depth 2 using `t0`. Checking all of the lemmas and theorems with SAL requires under 10 seconds on a mid-range Linux workstation.

5. 8N1 Protocol

The techniques described in Section 4 extend naturally to other serial protocols such as 8N1, which is commonly implemented in UARTs. Figure 11 illustrates the 8N1 encoding scheme implemented by the sender's UART. A *frame* is a sequence of bits that includes a start bit, eight data bits, and a stop bit. Data bits are encoded by the identity function—a 1 is a 1 and a 0 is a 0. Unlike in BMP in which the receiver can resynchronize on each cell, in the 8N1 protocol, the receiver must estimate the transmitter's clock based upon the frame's start bit only—the start bit is the only guaranteed observable clock event.

5.1. Model

The transmitter and receiver each have 10 states, labeled 0 – 9, corresponding to the phases of data transmission in a frame. For convenience, we define a type synonym:

```
STATE : TYPE = [0..9];
```

The transmitter is idle (i.e., repeatedly transmits the stop bit) in state 9, transmits the start bit in state 0, and transmits data bits in states 1-8. Likewise, the receiver begins in state 9 awaiting a start bit, decodes

```
tenc_8N1 : MODULE =
BEGIN
  OUTPUT  tdata  : BOOLEAN
  OUTPUT  tstate : STATE
  OUTPUT  tready : BOOLEAN
  INPUT   tbit   : BOOLEAN
  INITIALIZATION
    tstate = 9;
  DEFINITION
    tready = state /= 8;
    tdata = (tstate = 9) OR tbit;
  TRANSITION
  [
      tstate = 9 --> tstate' = IF tbit' THEN 9 ELSE 0 ENDIF;
   [] tstate < 9 --> tstate' = tstate + 1;
  ]
END;
```

Fig. 12. 8N1 Encoder (real-time)

```
rdec_8N1 : MODULE =
BEGIN
  OUTPUT rstate : STATE
  OUTPUT rbit   : BOOLEAN
  OUTPUT rdata  : BOOLEAN
  DEFINITION
    rdata = rbit
  INITIALIZATION
    rbit  = TRUE;
    rstate = 9;
  TRANSITION
  [
      rstate = 9  --> rbit' = TRUE;
   [] rstate = 9  --> rbit' = FALSE;
                      rstate' = 0;
   [] rstate /= 9 --> rbit' IN {FALSE, TRUE};
                      rstate' = rstate + 1;
  ]
END;
```

Fig. 13. 8N1 Decoder (real-time)

data at the end of states 0-7, and receives the stop bit in state 8. An execution is therefore correct if it has the following sequence of states, where "$\ldots$" denotes indefinite idling and "$\cdots$" denotes the removal of a finite number of states in the sequence for readability:

$$(\texttt{tstate}, \texttt{rstate}) = (9,9), (9,9), \ldots, (0,9), \ldots, (0,0), (1,0), (1,1), \cdots, (8,7), (8,8), (9,8), (9,9), \ldots$$

The transmitter's encoder `tenc_8N1` (Figure 12) has one input variable, `tbit`, which is the output of the `tenv` module. The encoder has three output variables: `tdata` is the data transmitted to the receiver, `tstate` is the transmitter's state, and `tready` is the encoder's signal to `tenv` whenever a new bit is required from the environment. The environment is the same as described for BMP in Section 4. The encoder module

```
rclock_8N1 : MODULE  =
  BEGIN
    INPUT tclk   : TIME
    INPUT rstate : STATE
    OUTPUT rclk  : TIME
  INITIALIZATION
     rclk IN {x : TIME | 0 <= x AND x < RSCANMAX};
  TRANSITION
  [
  rclk = time(rclk, tclk) -->
                    rclk' IN IF (rstate' = 9) THEN
                                 timeout(rclk + RSCANMIN, rclk + RSCANMAX)
                             ELSIF (rstate' = 0) THEN
                                 timeout(rclk + RSAMPMIN, rclk + RSAMPMAX)
                             ELSE
                                 timeout(rclk + RPERIODMIN, rclk + RPERIODMAX)
                             ENDIF;
  ]
END;
```

Fig. 14. 8N1 Receiver Clock (real-time)

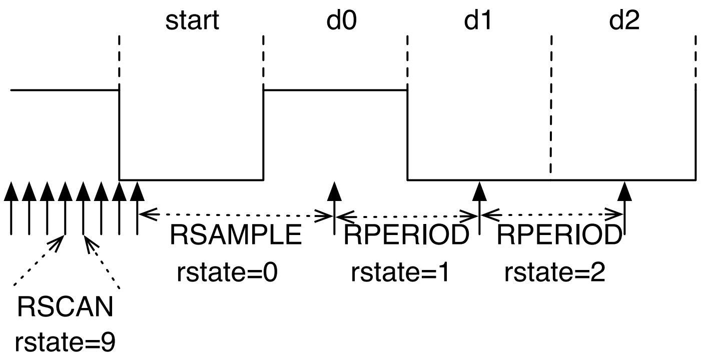

Fig. 15. 8N1 Data Sampling

has two guarded transitions. The first guarded transition defines the behavior in the idle state (9), which is exited when the environment produces a start bit. The second transition simply advances the transmitter's state; the ready signal is suppressed in state 8 to indicate to the environment that a stop bit (`TRUE`) will be inserted in the data stream. The 8N1 decoder is illustrated in Figure 13.

In summary, the BMP encoder (decoder) and the 8N1 encoder (decoder) are conceptually similar. In each protocol, the encoder provides a guaranteed transition to a "start" state followed by one or more transitions to data states. While BMP encodes the data value relative to the signal in the start state, 8N1 provides a level encoded signal.

The 8N1 receiver clock, illustrated in Figure 14, is more complex than the BMP clock with three distinct measurement intervals. The start transition is detected with a "scan" interval, data bit d0 is read with a "sample" interval and subsequent bits are read with a "period" interval. The relationship between these intervals and the states of the decoder are illustrated in Figure 15, which shows the start of a data frame along with the first three data bits.

```
t0 : LEMMA system_8N1 |- G(
   % idle
      ((rstate =  9) AND (tstate = 9) AND (tdata AND rbit) AND stable AND
       (rclk - tclk <= RSCANMAX))
   OR % start bit sent, not detected
      ((rstate = 9) AND (tstate = 0) AND (NOT tdata AND rbit) AND
       (rclk - tclk <= RSCANMAX - TSTABLE))
   OR % --- unwind 18 other cases
      rec_states(8, tstate, rstate, tdata, rbit, rclk, tclk, stable));

rec_states(n: STATE, tstate: STATE, rstate: STATE,
           tdata: BOOLEAN, rbit: BOOLEAN, rclk: TIME,
           tclk: TIME, stable: BOOLEAN): BOOLEAN =
      IF n = 0
      THEN disjunct_invar(n, tstate, rstate, tdata, rbit, rclk, tclk, stable)
      ELSE    disjunct_invar(n, tstate, rstate, tdata, rbit, rclk, tclk, stable)
           OR rec_states(n - 1, tstate, rstate, tdata, rbit, rclk, tclk, stable)
      ENDIF;
```

Fig. 16. 8N1 Disjunctive Invariant

5.2. Verification

The fundamental theorem for the 8N1 protocol is

```
Uart_Thm : THEOREM system_8N1 |-  G(rstate = tstate AND rstate /= 9 => rbit = tbit);
```

Verification of the 8N1 protocol is analogous to that of BMP. A key component is the set of linear constraints on the time constants. As with BMP, the scan time must be less than the stable portion of the transmitter period.

```
RSCANMIN : {x : TIME | 0 < x};
RSCANMAX : {x : TIME | RSCANMIN <= x AND x < TSTABLE};
```

Similarly, the first sample after the start transition must fall within the stable phase of the second transmission period.

```
RSAMPMIN : {x : TIME | TPERIOD + TSETTLE < x};
RSAMPMAX : {x : TIME | RSAMPMIN <= x AND x < 2 * TPERIOD - TSETTLE - RSCANMAX};
```

Subsequent samples by the receiver must also fall within the stable phase of subsequent transmission periods.

```
RPERIODMIN : {x : TIME | 9 * TPERIOD + TSETTLE < RSAMPMIN + 8 * x};
RPERIODMAX : {x : TIME | RPERIODMIN <= x AND
                   TSETTLE + RSCANMAX + RSAMPMAX + 8 * x < 10 * TPERIOD};
```

The theorems of the proof are also similar to those of BMP. We developed two lemmas defining basic properties of the clocks:

```
l1 : LEMMA system_8N1 |- G(tclk <= (rclk + TPERIOD) OR stable);
l2 : LEMMA system_8N1 |- G(rclk <= tclk + RSAMPMAX OR rclk <= tclk + RPERIODMAX);
```

As with BMP, we developed a disjunctive invariant enumerating the states defined by `rstate` and `tstate`. While the BMP invariant had four disjuncts, the 8N1 invariant has 20. Essentially, each disjunct characterizes a possible configuration of the system—recall that each of the sender and receiver has 10 states in 8N1. The lemma, built using a helper function, is shown in Figure 16.
Checking the proof takes approximately a few seconds on a mid-range Linux workstation.

```
    RSTARTMAX : TIME = TSTART * (1 + ERROR);
    RSTARTMIN : TIME = TSTART * (1 - ERROR);
    RSCANMAX : TIME = 1 + ERROR;
    RSCANMIN : TIME = 1 - ERROR;
    RPERIODMAX : TIME = TPERIOD * (1 + ERROR);
    RPERIODMIN : TIME = TPERIOD * (1 - ERROR);
```

Fig. 17. Receiver Parameters Defined with respect to Error

5.3. Calculating Error Constraints

Our proof of 8N1 is verified with respect to bounds on the various timing constants. In a practical implementation, the receiver scan period is defined relative to the nominal transmitter bit period and the receiver start and bit periods are integer multiples of this. What an implementor ultimately cares about is the trade off between settling time (generally due to signal dispersion over a given transmission medium) and frequency error.

Here, we show how the bounds that we have verified can be used to derive error and settling-time bounds in a form that is more convenient for a protocol implementer. These derived bounds are somewhat more restrictive than what we have verified since we require the maximum allowable frequency error to be symmetric about the nominal frequency. As before, let TPERIOD be the nominal period duration. We introduce another uninterpreted constant in the operational model representing the nominal duration the receiver waits after receiving the start bit (TSTART).

```
TSTART : TIME;
```

Now, let ERROR be an uninterpreted constant from TIME, then we define the constants in Figure 17 in terms of ERROR. By replacing these defined terms in the parameterization of the types in Section 5.2, we compute the bound on the error. For example, RSTARTMAX is an uninterpreted constant from the following parameterized type:

```
RSTARTMAX : {x : TIME | RSTARTMIN <= x AND TSETTLE + RSCANMAX + x < 2 * TPERIOD};
```

Replacing RSTARTMIN and RSCANMAX by their definitions from Section 5.2, we get

```
RSTARTMAX : {x : TIME |     TSTART * (1 - ERROR) <= x
                        AND TSETTLE + 1 + ERROR + x < 2 * TPERIOD};
```

By replacing each term with its definition, the type parameters are defined completely in terms of TPERIOD, TSETTLE, and ERROR. Isolating ERROR in the system of inequalities gives bounds on ERROR. For the 8N1 protocol, ERROR is thus parameterized as follows:

```
ERROR : {x : TIME |
      0 <= x
  AND (9 * TPERIOD + TSETTLE < 8 * TPERIOD * (1-x) + TSTART * (1-x))
  AND ((8 * TPERIOD * (1+x) + TSTART * (1+x) + (1+x) + TSETTLE) < 10 * TPERIOD)};
```

This derived model can be verified using the same invariants proved at the same depth as in the verification described previously.

We discovered significant errors in an application note for UARTs [Max03]. For TPERIOD = 16 and TSTART = 23, the authors suggest that if TSTABLE is TPERIOD/2 (they call this the “nasty” scenario), then a frequency error of ±2% is permissible. In fact, even with zero frequency mismatch, the stable period is too short—if we assume “infinitely” fast sampling, it is possible to show that the settling time must be less than 50% of TPERIOD. In other words, the type parameterizing ERROR is empty when TSTABLE is TPERIOD/2. With our choice of time constants, the longest settling time must be less than 7 (43.75%). In reading the article, it becomes clear that the authors neglected the temporal error introduced by sampling the start bit. They describe a “normal” scenario with TSETTLE = TPERIOD/4 and assert that a frequency

```
tclock_fs : MODULE =
BEGIN
  INPUT rstate : STATE
  INPUT tstate : STATE
  TRANSITION
    [ tstate = rstate --> ]
END;
```

Fig. 18. 8N1 Transmitter Clock (Finite-State)

error of $\pm 3.3\%$ is permissible. As our derivation above illustrates, the frequency error in this case is limited to $\pm 3/151 \approx \pm 1.9\%$.

6. System Design

While the protocol models described in Sections 4 and 5 are useful in understanding implementation timing requirements, they are not compositional with larger system designs. For instance, consider a system model that is the composition of a synchronous hardware model of the sender and receiver, respectively, and a real-time communication protocol between them. Supposing someone wished to prove some safety property of the entire system, one would be forced to strengthen an invariant similar to the ones presented in Figures 10 and 16 to include invariants of the synchronous hardware. Doing so (1) conflates two levels of abstraction—the synchronous hardware and the real-time protocol, (2) may overwhelm the model-checker (recall that SAT-solving is exponential in the size of the model), and (3) increases the difficulty to the user in formulating a non-compositional invariant for the entire system.

In this section, we develop a finite-state specification for the 8N1 protocol and demonstrate how to compose it with a larger system specification. For a simple example, we model parallel-to-serial and serial-to-parallel converters (shift registers) normally associated with UART transmitters and receivers, respectively. The finite specification of the protocol can be composed with a finite-state specification of the synchronous hardware, and because the entire specification is finite-state, it can be model-checked using BDDs.

The use of this finite-state protocol specification naturally leads us to demonstrate that a real-time infinite-state model refines the finite-state one. We prove the refinement using infinite-bmc. Furthermore, the temporal refinement proof can be decomposed from the remainder of the system specification, making the refinement tractable (note we are doing *temporal* refinement, not data refinement). While this section demonstrates our refinement approach using the 8N1 protocol, a temporal refinement of BMP is available on-line.[1]

The section is organized as follows. In Section 6.1, we present our finite-state specification of the clocks and decoder. Section 6.2 fills in the system design with shift registers for the transmitter and receiver. Finally, we present some algebraic laws for formulating the necessary set of refinement theorems and apply these laws to the 8N1 protocol refinement in Section 6.3.

6.1. 8N1 Specification

The fundamental difference between the infinite state implementation and finite state specification for the 8N1 protocol is the definition of clocks. In the infinite state model, the clocks are synchronized by linear real-time constraints. In the finite state specification, the clocks directly reference the state variables of the transmitter and receiver and hence use global state predicates to impose this ordering. The 8N1 specification consists of three new modules—`rclock_8N1_fs`, `tclock_fs`, and `rdec_8N1_fs`, which are the respective analogues of `rclock_8N1_rt`, `tclock_rt`, and `rdec_8N1`. The temporal refinement is over these modules—intuitively, these are the modules that are crucially real-time—the clocks by linear real-time constraints and the decoder by metastability effects due to crossing the clock boundary.

In the finite-state specification, the transmitter may transition between states whenever the states of the

```
rclock_8N1_fs : MODULE  =
BEGIN
  INPUT tstate : STATE
  INPUT rstate : STATE
  TRANSITION
    [ rstate /= tstate OR tstate = 9 --> ]
END;
```

Fig. 19. 8N1 Receiver Clock (Finite-State)

```
rdec_8N1_fs : MODULE =
BEGIN
  INPUT  tdata  : BOOLEAN
  OUTPUT rstate : STATE
  OUTPUT rbit   : BOOLEAN
  INITIALIZATION
    rbit  = TRUE;
    rstate = 9;
  TRANSITION
  [   rstate  = 9              -->
   [] rstate  = 9 AND NOT tdata -->  rbit' = FALSE;
                                     rstate' = 0;
   [] rstate /= 9              -->  rbit' = tdata;
                                     rstate' = rstate + 1;
  ]
END;
```

Fig. 20. 8N1 Decoder (Finite-State)

transmitter and receiver are equal. We capture this requirement with the `tclock_fs` module in Figure 18. The receiver's clock in Figure 19 is similarly modeled, although it is allowed to transition only when the states of the transmitter and receiver are unequal or the transmitter has not yet sent a start bit. The two clocks serve to constrain the inter-leavings of the sender and receiver.

The receiver's decoder `rdec_8N1_fs` (Figure 20) has a design similar to `rdec_8N1`. Of its four variables, `tdata` is the data sent by the transmitter (i.e., the output variable of `tenc`), `rstate` is the receiver's state, and `rbit` records the value received from the transmitter. The module has three transitions: the first allows the receiver to idle waiting for the start bit, the second transitions to the "start state", and the last transition governs states in which it is sampling for data. The data is ready to be consumed when the receiver is in one of states 1-8.

The transitions of `rdec_8N1_fs` are under constrained—even when the transmitter provides a start bit, the receiver may continue to stutter in state 9. Stuttering is an essential characteristic of physical reality where changes to signals take time to propagate and to be correctly sampled. In the design of the implementation, we introduce a constraint process that limits this stuttering based upon real-time properties of the local clocks and which guarantees that the stuttering is finite.

Theorem `Uart_Thm` of Section 5 is easily verified using SAL's BDD-based model checker. We also used the BDD-based model checker to demonstrate that the possible sequences of system system states are exactly characterized by the sequence listed in Section 5 by proving the following three theorems:

```
StateThm1 : THEOREM system_8N1_fs |-
  G(rstate = tstate OR ((rstate + 1) MOD 10) = tstate);
```

```
tenv_reg : MODULE =
BEGIN
  INPUT tstate : STATE
  LOCAL treg   : REG
  OUTPUT  tbit : BOOLEAN
  TRANSITION
  [
      tstate = 9 --> treg' IN {reg: REG | TRUE}
   [] tstate < 8 --> tbit' = treg[7-tstate]
   [] tstate = 8 -->
  ]
END;
```

Fig. 21. Transmitter's Environment Instantiated by a Shift Register

```
StateThm2 : THEOREM system_8N1_fs |-
  G(FORALL (i : STATE) : (i = rstate AND i = tstate) =>
      X(   (i = rstate AND (i + 1) MOD 10 = tstate)
        OR (i = rstate AND i = tstate AND i = 9)));

StateThm3 : THEOREM system_8N1_fs |-
  G(FORALL (i : STATE) : (i = rstate AND (i + 1) MOD 10 = tstate) =>
      X(   ((i + 1) MOD 10 = rstate AND (i + 1) MOD 10 = tstate)
        OR (i = 9 AND i = rstate AND 0 = tstate)));
```

Finally, SAL's deadlock checker ensures liveness of the model; given the assumption that the receiver eventually detects a start bit, the system will eventually return to an "idle" state.

```
Liveness: THEOREM system_8N1_fs |- G(F(rstate /= 9) => F(tstate = 9 AND rstate = 9));
```

6.2. Temporal Refinement for System Design

In the system specifications, we instantiate the transmitter's environments in both the finite-state and infinite-state models with a parallel-to-serial shift register. Likewise, we augment the receiver with a serial-to-parallel shift register module by synchronously composing it with the other receiver modules.

We will suppose that the transmitter's environment receives data as 8-bit bytes, and the words are sent to the receiver via the 8N1 protocol. The receiver then stores the incoming words in its own 8-bit register to be read as bytes by other hardware on the receiver-side. To begin, we define a SAL type, `REG`, the type of 8-bit arrays:

```
REG : TYPE = ARRAY [0..7] OF BOOLEAN;
```

The transmitter's environment instantiates the abstract transmitter encoder defined in Figure 2 with the encoder defined in Figure 21. A new register is loaded in state 9, and data bits are read from the register in states 0-7.

In the receiver specification, we now need a byte array to hold the incoming data bits. We define the array using a new module, `rdec_reg`, as defined in Figure 22. Module `rdec_reg` is synchronously composed with the `rdec_8N1` and `rclock_8N1_fs` modules. Initially, the register is empty; that is, each bit is set to `FALSE`. The register is a shift register; incoming bits are stored at index 0, and as new bits arrive, old bits are shifted to the right.

In summary, the finite-state system specification is the following composition:

```
rdec_reg : MODULE =
BEGIN
  INPUT rbit : BOOLEAN
  INPUT rstate : STATE
  OUTPUT rreg : REG
  INITIALIZATION
    rreg = [[i:[0..7]] FALSE]
  TRANSITION
  [
      rstate < 8 --> rreg' = [[i:[0..7]] IF i=0 THEN rbit' ELSE rreg[i-1] ENDIF]
   [] ELSE       -->
  ]
END;
```

Fig. 22. Receiver's 8-Bit Register Module

```
tx_8N1_fs : MODULE = tclock_fs || tenc_8N1_fs || tenv_reg;
rx_8N1_fs : MODULE = rclock_8N1_fs || rdec_8N1 || rdec_reg;
system_8N1_fs : MODULE = tx_8N1_fs [] rx_8N1_fs;
```

And the infinite-state specification just adds shift registers to the model presented in Section 5:

```
tx_8N1 : MODULE = tclock_rt || tenc_8N1 || tenv_reg;
rx_8N1 : MODULE = rclock_8N1_rt || rdec_8N1 || rdec_reg;
system_8N1 : MODULE =  tx_8N1 [] rx_8N1;
```

With the system specification in place, we can now prove a correctness theorem stating that at the end of each frame, the transmitter's and receiver's registers are equivalent at the end of each transmission frame:

```
Thm_Uart_Reg : THEOREM system_8N1_fs |- G(rstate = 8 => rreg = treg);
```

The theorem is proved in the finite-state model using BDDs in a few seconds on a mid-range Linux workstation. However, if the infinite-state model is similarly refined, invariants must be proved about the registers, in addition to the timing constraints. Although the invariants could conceivably be strengthened, the model and associated proof become non-compositional. That is, it should be possible to augment the transmitter and receiver models, respectively, with additional synchronous hardware specifications without having to correspondingly augment the proof that the physical-layer communication protocol behaves correctly. The synchronous hardware specifications are conceptually independent of the protocol.

Therefore, as mentioned in the introduction to this section, we demonstrate how to do a temporal refinement from the finite-state specification to the infinite-state specification. The refinement ensures the infinite-state implementation inherits the system-level safety property (i.e., in this case, that the registers are equivalent at the end of each frame) from the finite-state specification. The temporal refinement ensures compositionality because, as shown below, only the portion of the model mentioning real-time variables and constraints needs to be refined. The approach is summarized in Figure 23.

6.3. Refinement Proof

With both a finite-state specification and an infinite-state implementation in hand, we can now prove the latter refines the former. More specifically, every possible interleaving of the transmitter and receiver in the

Specification	*Implementation*
• Finite-state model.	• Infinite-state model (time is modeled by $\mathbb{R}$).
• Nondeterministic interleaving semantics.	• Linear real-time constraints.
• Safety properties *proved* by BDD model checking.	• Safety property *inherited* from the specification.
• Compositional with other finite-state specifications.	• Refinement proof of the real-time portion, via inf-bmc.

Fig. 23. Temporal Refinement Summary

implementation is a possible interleaving of the specification. Paraphrasing Abadi and Lamport, **I** implements **S** if every externally visible behavior of **I** is also allowed by **S** [AL91]. To prove that **I** implements **S** it is sufficient to prove that if **I** allows the behavior

$$\langle\langle(e_0, z_0), (e_1, z_1), (e_2, z_2), ...\rangle\rangle$$

where each e_i is an externally-visible state, and where each z_i is an internal state (the remainder of the state), then there exist internal states y_i such that **S** allows

$$\langle\langle(e_0, y_0), (e_1, y_1), (e_2, y_2), ...\rangle\rangle$$

In general, an Abadi-Lamport style refinement proof can be difficult and may rely upon the introduction of history and prophecy variables [AL91]. For the class of protocols we consider in this paper, the refinement mappings are straightforward. The basic transformation rule that we apply is *guard weakening* over SAL's guards to show that the guards of the implementation imply the guards of the specification. Consider a module consisting of the following set of guarded transitions:

$$G_0 \rightarrow S_0 [] \ldots [] G_N \rightarrow S_N$$

where each G_i is a predicate over the state variables, and each S_i is a set of possibly nondeterministic next-state variable assignments. If G_i implies G'_i for each i, then the following set of guarded commands allow a superset of behaviors and are therefore a specification for the former:

$$G'_0 \rightarrow S_0 [] \ldots [] G'_N \rightarrow S_N$$

Theorems of the form $G_i \Rightarrow G'_i$ are the *refinement conditions*. The semantics of synchronous and asynchronous composition in SAL are such that the effect of weakening any guard is to expand the set of legal behaviors [dMOR+04]; we have covered the case of asynchronous composition above. In the case of synchronous composition, observe that the following equivalence holds:

$$(G_0 \rightarrow S_0 \parallel G_1 \rightarrow S_1) \equiv (G_0 \wedge G_1 \rightarrow S_0; S_1)$$

(Recall from Section 2.2 that the variable-assignment separator ; is commutative.) Weakening either G_0

```
    rdec_thm1 : THEOREM system_8N1 |- G(rstate = 9 AND X(rbit) => X(rbit) = rbit);

    rdec_thm2 : THEOREM system_8N1 |- G(rstate = 9 AND X(NOT rbit)
                                   AND X(rstate = 0) => NOT tdata);

    rdec_thm3 : THEOREM system_8N1 |- G(FORALL (i : [0..8]) :
                  rstate = i AND X(rstate = i+1) => X(rbit) = tdata);
```

Fig. 24. Refinement Theorems for the Receiver's Decoder

or G_1 has the effect of weakening $G_0 \wedge G_1$. Indeed, by weakening either guard to `True`, its associated module can be eliminated (effectively abstracting away the module's outputs). The key observation is that an unconstrained input is treated by SAL as having any possible value—eliminating a module and its associated outputs increases the non-determinism of those signals. Thus, in a synchronous composition `P || Q`, guard weakening can be generalized to module elimination: `P` and `Q` are both specifications of `P || Q`. A corollary to the equivalence above allows us to "split" a process:

$$(G_0 \to S_0) \equiv ((G_0 \to \mathit{skip}) \parallel (\mathit{TRUE} \to S_0))$$

With the above algebraic rules in hand, we proceed to refine the system design of the 8N1 protocol. The refinement is principally over the clock modules of the finite-state and infinite-state models. Recall that `system_8N1` is the composed module `(tx_8N1 [] rx_8N1) || constraint`. From the law pertaining to module elimination described above, the `constraint` module (from the implementation) can be eliminated, and the composition `tx_8N1_fs [] rx_8N1_fs` is a specification of `system_8N1`. For the transmitter implementation module `tx_8N1`, we prove the single guard in `tclock_rt` (Figure 3) implies that the single guard in `tclock_fs` (Figure 18) holds:

```
    tclock_thm : THEOREM system_8N1 |- G(tclk = time(tclk, rclk) => tstate = rstate);
```

The proof is automated in SAL using infinite-bmc induction. The proof requires a small invariant bounding the maximum and skew between the clocks as a function of the transmitter's and receiver's respective states.

The theorem `tclock_thm` is the only refinement condition required for `tx_8N1`: the other two modules, `tenv_reg` and `tenc_8N1`, are identical in the specification and implementation. As for the module `rx_8N1`, recall that the module is the parallel composition `rdec_8N1 || rclock_8N1_rt || rdec_reg`. The refinement condition and proof for `rclock_8N1_rt` (Figure 14) is exactly analogous to that for `tclock_rt`. We can ignore `rdec_reg` since the module is identical in the specification and implementation. This leaves only `rdec_8N1` to refine.

To complete the refinement proof, we prove a refinement condition that slightly generalizes guard weakening. The module `rdec_8N1` (Figure 13) is actually less deterministic than is `rdec_8N1_fs` (Figure 20), but its nondeterminism is limited by the constraint module. In simple guard weakening, the next-state variable assignments in the specification and implementation are identical, but for the modules `rdec_8N1_fs` and `rdec_8N1`, they are not. In this case, the hypothesis of the refinement condition is a conjunction of the implementation's guard and the next-state variable values produced by the implementation's transition. Likewise, the antecedent of the refinement condition is a conjunction of the specifications guard and the next-state variable values produced by the specification's corresponding transition. (A weaker condition—for example, one in which the hypothesis is not strengthened with all of the next-state variable values from the implementation—still proves refinement.) Applying this reasoning to `rdec_8N1` and `rdec_8N1_fs`, we prove three theorems, one for each of the three corresponding guarded transitions in the two modules, as shown in Figure 24. For example, consider `rdec_thm2`: this refines the second transition of `rdec_8N1_fs` (Figure 20) to the second transition of `rdec_8N1` (Figure 13). The hypothesis of the theorem is

```
    rstate = 9 AND X(NOT rbit) AND X(rstate = 0)
```

The first conjunct corresponds to the guard of `rdec_8N1`, while the second two conjuncts correspond to the next-state values of that transition. Together, these imply the guard of the second transition of `rdec_8N1_fs`:

```
    rstate = 9 AND NOT tdata
```

For simplicity, we do not reproduce the sentence `rstate = 9` in the consequent of the transition, only sentence `NOT tdata`. Each of the three refinement theorems is proved using the same disjunctive invariant mentioned in the refinement proofs for the clock modules. The theorems are proved using k-induction, where $k = 2$.

7. Discussion

In this paper, we have demonstrated, by example, how to use infinite-state bounded model-checking to verify the correctness of physical-layer protocols. Furthermore, we have given an approach for using these techniques in system-design, requiring one to reason about both synchronous and asynchronous subsystems. Our work compares favorably with previous efforts: theorem-proving verifications are tedious and require significant user expertise; model-checking verifications are not as parameterized and do not easily support system-level design.

Since our work is a case-study, we provide some "lessons learned" to inform similar future endeavors. In Section 2, we gave some metrics comparing our verification efforts to previous work. In our work, the vast majority of our time was spent generalizing the model. Developing the "right" modeling abstractions took a few weeks of effort. Instantiating the general model with another physical-layer protocol would probably take a day of effort (the effort required is speculative since we developed the general model by abstracting the specific protocols). Although we conjecture that carrying out the proof via SMT and model checking is at least an order-of-magnitude easier (measured in, say, engineer-hours of effort) than using mechanical theorem proving, it is still the most onerous aspect of the verification. The particular difficulty results from building up the main disjunctive invariant for a sufficiently small value of k. Developing the invariant for a new protocol may take on the order of a day or two of effort. The structure of the BMP and 8N1 invariants are quite similar; we expect invariants for other protocols to have a correspondingly similar structure.

7.1. Shortcomings

We highlight the following shortcomings of our approach: (1) ensuring soundness of the model, (2) by-hand algebraic reasoning for refinement conditions, and (3) SMT limitations.

7.1.1. Soundness

In our approach, there are two essential causes of potential unsoundness. The first cause is that infinite-bmc is complex and highly-automated. Mechanical theorem-proving offers a high-degree of confidence to the user that proofs are sound. Proof steps are usually "small" and in general, the theorem-prover checks rather than generates proofs. Comparatively, infinite-bmc combines both a model-checker and SMT solver. Both are complex software systems and used as "black-box" solvers in our work. Furthermore, SMT in particular is relatively recent technology and the Yices SMT solver is beta-quality software.

The second cause of potential unsoundness is that infinite-bmc cannot be used to prove liveness properties. In a deadlocked system, safety properties may hold vacuously. Liveness properties can prove that the system is not deadlocked. Note that in Section 6, we showed how to refine a finite-state specification into a infinite-state real-time implementation. We were able to prove liveness of the finite-state specification, but as noted by an anonymous reviewer of a recent paper by the authors [BP07], refinement does not preserve liveness.

In our work, we have attempted to mitigate both soundness shortcomings. As for the first soundness issue, we have indeed encountered some bugs in our work with SAL, as we were using the latest releases. One test for soundness is to use constraints that should not hold or to attempt to prove properties known to be false. In either case, SAL should return a counterexample. As for the second soundness issue, again, as an approximation to liveness, we can attempt to prove some safety property of the implementation that should be false (e.g., for both BMP and 8N1, `G(tstate > 0)` is false and SAL should return a corresponding counterexample). If the system is deadlocked, then the property may be proved. If the property is not proved, then either the system is not deadlocked, or the property is not k-inductive, for the value of k and lemmas used in the proof effort. We call these safety properties "poor man's liveness properties". Failed proofs of this sort for rather large values of k increase our confidence that the models are deadlock-free.

7.1.2. Algebraic Reasoning

While Section 6 demonstrated the feasibility of automating parts of refinement proofs utilizing SAL, the work was somewhat hampered by the lack of a well-developed algebraic theory for the SAL language and by the need to apply algebraic reasoning by hand. Automated model checking techniques have matured sufficiently to prove the tedious refinement conditions presented in this paper, but combining SAL with a system capable of algebraic reasoning (e.g., an interactive theorem prover) would both increase the assurance and decrease the tedium of deriving the implementation and refinement conditions from the specification.

7.1.3. SMT Limitations

In our work, two limitations of SMT solvers in general inhibited our results. The first limitation is the class of theories solvable and the second limitation is the complexity of SAT solving.

Regarding the first limitation, while SMT solvers can determine satisfiability over an impressive combination of theories that continues to grow [BOS07], some theories (and unions of theories) are not decidable. For example, the theory of arithmetic, which includes variable multiplication, is not decidable. Our results in earlier work were not fully-parameterized because of this limitation; that is, we instantiated parameters with constants to ensure the constraints were linear [BP06]. Initially, the constraints for the BMP and 8N1 protocols included explicit error terms that accumulated over the execution of the protocol. An anonymous referee to that paper suggested a model in which the accumulated error is not explicit in the constraints but is captured by the other terms. The constraints presented in this paper are expressed this way.

The second limitation is the complexity limitation inherent in SAT-based techniques, such as SMT. The satisfiability of boolean formulas is exponential in the worst case. The main limitation this caused for us is in unrolling the transition relation during k-induction. For models of our size, $k \geq 10$ resulted in SAL taking more than approximately five minutes to return a result (the execution time is exponential in the size of k, in the worst case). The mitigation is to strengthen the required invariants manually, as we demonstrated in Section 4.2 (if we had been able to unroll the entire transition relation, no invariant would have been required to prove our main theorem). Despite the inherent limitations to SAT-solving, k-induction for $k < 10$ significantly reduced the complexity of the invariants we would have had to state and prove. Although we did attempt to prove the property using only $k = 1$-induction in SAL, the complexity of the invariant would likely have been on the same order as the invariants required in mechanical theorem-proving efforts, as described in Section 2.

7.2. Future Work

As pointed out by an anonymous reviewer, although we present a methodology for temporal refinement, the real-time implementation can itself be refined using data refinement techniques. The data-refined implementation should still inherit the safety properties from the untimed, data-abstract specifications. Carrying out data refinement on our model is future work.

More broadly, the work presented herein fits into an end-to-end verification methodology for distributed real-time systems and in particular, *time-triggered systems*. Time-triggered systems are distributed systems in which the nodes are independently-clocked but maintain synchrony with one another [Kop97]. Because time-triggered systems can make hard real-time guarantees and possess fault-tolerance properties, they are being developed for next-generation fly-by-wire and drive-by-wire systems; some noteworthy examples include Honeywell's SAFEbus, TTTech's Time-Triggered Architecture (TTA), NASA's SPIDER, and FlexRay, being designed by an industrial consortium [Rus01]. Recent work is particularly focusing on how to model and verify these systems end-to-end—from the distributed fault-tolerant protocols to the hardware [Pik07, KP06, Sch07]. Our contribution is a simple way to verify the physical-layer of the distributed systems.

Integrating our results into an end-to-end verification would require demonstrating that the timing characteristics of a physical-layer protocol are sufficient to meet the timing constraints required in the *time-triggered model*, one abstraction layer above the physical layer. The essential feature of this model is that while the local clocks of individual nodes may not be perfectly synchronized, their disharmony is bounded [Pik07]. Integrating these results into a time-triggered system verification remains future work.

8. Acknowledgments

The authors wish to thank John Rushby and Bruno Dutertre for comments and suggestions on this research. Anonymous reviewers provided valuable comments on the conference papers upon which this article is based [BP06, BP07] as well as this article.

References

[AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theor. Comput. Sci.*, 82(2):253–284, 1991.

[BOS07] Clark Barrett, Albert Oliveras, and Aaron Stump. website, July 2007. Available at `http://www.smtcomp.org/`.

[BP06] Geoffrey M. Brown and Lee Pike. Easy parameterized verification of biphase mark and 8N1 protocols. In *The Proceedings of the 12th International Conference on Tools and the Construction of Algorithms (TACAS'06)*, pages 58–72, 2006. Available at `http://www.cs.indiana.edu/~lepike/pub_pages/bmp.html`.

[BP07] Geoffrey M. Brown and Lee Pike. Temporal refinement using smt and model checking with an application to physical-layer protocols. In *The Proceedings of the Fifth ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'07)*, 2007.

[DdM06] Bruno Dutertre and Leonardo de Moura. Yices: an SMT solver. Available at `http://yices.csl.sri.com/`, August 2006.

[dMOR$^+$04] Leonardo de Moura, Sam Owre, Harald Rueß, John Rushby, N. Shankar, Maria Sorea, and Ashish Tiwari. SAL 2. In *Computer-Aided Verification, CAV'04*, volume 3114 of *LNCS*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.

[dMRS03] Leonardo de Moura, Harald Rueß, and Maria Sorea. Bounded model checking and induction: From refutation to verification. In *Computer-Aided Verification, CAV'03*, volume 2725 of *LNCS*, 2003.

[DS04] Bruno Dutertre and Maria Sorea. Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata. In *FORMATS/FTRTFT*, pages 199–214, 2004.

[HPWT01] T. Henzinger, J. Preussig, and H. Wong-Toi. Some lessons from the Hytech experience. In *Proceedings of the 40th Annual Conference on Decision and Control*, pages 2887–2892, 2001.

[HRSV01] Thomas S. Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. Technical Report RS-01-5, BRICS, University of Aarhus, January 2001.

[Hun98] D. V. Hung. Modelling and verification of biphase mark protocols using PVS. In *Proceedings of the International Conference on Applications of Concurrency to System Design (CSD'98)*, Aizu-wakamatsu, Fukushima, Japan, March 1998, pages 88–98. IEEE Computer Society Press, 1998.

[Kop97] Hermann Kopetz. *Real-Time Systems.* Kluwer Academic Publishers, 1997.

[KP06] S. Knapp and W.J. Paul. Realistic worst case execution time analysis in the context of pervasive system verification. In *Program Analysis and Compilation, Theory and Practice: Essays Dedicated to Reinhard Wilhelm*, LNCS volume 4444, pages 53–81. Springer, 2006.

[Max03] Maxim Integrated Products, Inc. *Determining Clock Accuracy Requirements for UART Communications*, June 2003. Available at `http://www.maxim-ic.com/appnotes.cfm/appnote_number/2141`.

[Moo94] J Strother Moore. A formal model of asynchronous communication and its use in mechanically verifying a biphase mark protocol. *Formal Aspects of Computing*, 6(1):60–91, 1994.

[Pik07] Lee Pike. Modeling time-triggered protocols and verifying their real-time schedules. In *Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2007. Available at `http://www.cs.indiana.edu/~lepike/pub_pages/fmcad.html`.

[PJ05] Lee Pike and Steven D. Johnson. The formal verification of a reintegration protocol. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 286–289, New York, NY, USA, 2005. ACM Press.

[Rus00] John Rushby. Verification diagrams revisited: Disjunctive invariants for easy verification. In *Computer-Aided Verification, CAV'00*, volume 1855 of *LNCS*, pages 508–520, Chicago, IL, July 2000. Springer-Verlag.

[Rus01] John Rushby. Bus architectures for safety-critical embedded systems. In Tom Henzinger and Christoph Kirsch, editors, *EMSOFT 2001: Proceedings of the First Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323, Lake Tahoe, CA, October 2001. Springer-Verlag.

[Rus06] John Rushby. Harnessing disruptive innovation in formal verification. In *4th IEEE International Conference on Software Engineering and Formal Methods (SEFM)*. IEEE Computer Society, 2006. Available at `http://www.csl.sri.com/users/rushby/abstracts/sefm06`.

[SBS05] Sanjit A. Seshia, Randal E. Bryant, and Kenneth S. Stevens. Modeling and verifying circuits using generalized relative timing. In *ASYNC*, pages 98–108, 2005.

[Sch07] Julien Schmaltz. A formal model of clock domain crossing and automated verification of time-triggered hardware. In *Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2007.

[VdG04] F. W. Vaandrager and A. L. de Groot. Analysis of a Biphase Mark Protocol with Uppaal and PVS. Technical Report NIII-R0455, Nijmegen Institute for Computing and Information Science, 2004.