Easy Parameterized Verification of Biphase Mark and 8N1 Protocols

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

[1] Indiana University, Bloomington
geobrown@cs.indiana.edu
[2] Galois Connections
leepike@galois.com

Abstract. The Biphase Mark Protocol (BMP) and 8N1 Protocol are physical layer protocols for data transmission. We present a generic model in which timing and error values are parameterized by linear constraints, and then we use this model to verify these protocols. The verifications are carried out using SRI's SAL model checker that combines a *satisfiability modulo theories* decision procedure with a bounded model checker for highly-automated induction proofs of safety properties over infinite-state systems. Previously, parameterized formal verification of real-time systems required mechanical theorem-proving or specialized real-time model checkers; we describe a compelling case-study demonstrating a simpler and more general approach. The verification reveals a significant error in the parameter ranges for 8N1 given in a published application note [1].

1 Introduction

The Biphase Mark Protocol (BMP) and 8N1 Protocol are common physical layer protocols used in data transmission – BMP in CDs, Ethernet, and Tokenring and 8N1 in UARTs. Decoders for protocols such as these present challenging formal verification problems because their correctness depends upon reasoning about interacting real-time events. BMP was first verified using the Boyer-Moore Theorem Prover (Nqthm) [2]. Subsequently, it was verified using a Duration Calculus model in the PVS theorem prover [3], with the HyTech model checker [4, 5] and also using a combination of the Uppaal model checker and PVS [6]. In this paper, we show how a parameterized specification of BMP can be verified easily with the SAL tool set using its built-in bounded model checker in conjunction with a satisfiability modulo theories (SMT) decision procedure to complete induction proofs over infinite-state systems [7].[1]

Compared to interactive mechanical theorem proving – the usual method for parameterized verification – this approach is substantially simpler. For example,

⋆ The majority of this work was completed while this author was a member of the Formal Methods Group at the NASA Langley Research Center in Hampton, Virginia.

[1] The SAL specifications and a proof script are available at `http://www.cs.indiana.edu/~lepike/pub_pages/bmp.html`.

H. Hermanns and J. Palsberg (Eds.): TACAS 2006, LNCS 3920, pp. 58–72, 2006.

the proof of Vaandrager and de Groot using PVS requires 37 invariants whereas ours requires only five. Because invariants can be combined, a more meaningful (and striking) metric may be the number of user-directed proof steps required: their initial verification effort required more than 4000 steps whereas each of our five invariants is proved automatically by SAL. As another comparison, the verification reported by Hung has such complexity that PVS requires approximately five hours just to *check* the validity of the manually-created proof script. In our approach, the proofs are *generated* by the bounded model checker and decision procedure in just a few seconds. We emphasize the simplicity of the invariants necessary in our verification, the "push-button" technique used to prove them, and the robustness of the proofs under modifications to the underlying model. In fact, we demonstrate that with a few trivial changes to the model, the proof for BMP naturally leads to a similar proof for 8N1. Finally, in verifying the 8N1 decoder, we found a significant error in a published application note that incorrectly defines the relationship between various real-time parameters which, if followed, would lead to unreliable operation [1].

While the verification approach described in this paper can be orders-of-magnitude easier than mechanical theorem-proving, it is less general. In the models presented, we fix two small integer constants, and the constraints on the model are parameterized with respect to these. The fixed constants are sufficiently small, and the verification is sufficiently fast, so that these values can be enumerated, and the constraints can be checked for each value. Alternatively, an anonymous referee pointed out an alternative formulation of the constraints that allows for a fully parameterized verification of BMP (the limitation of the SAL's ICS decision procedure described in Section 5 prevents a fully parameterized verification of 8N1). We present a fully parameterized model similar to the one suggested by the referee in the SAL specifications provided on-line, and we briefly describe the approach in Section 7. We also make a more detailed comparison to other verifications, including those carried out using specialized real-time model checkers, in Section 7.

To motivate the design of this sort of protocol, consider Figure 1 where the top stream is the signal to be transmitted, while the middle stream is a digital clock that defines the boundaries between the individual bits. In a digital circuit, the clock signal is transmitted separately from the data; however, this is not feasible

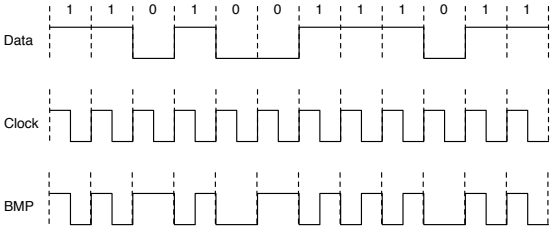

Fig. 1. Data and Synchronization Clock

in most communication systems (e.g., serial lines, Ethernet, SONET, Infrared) in which a single signal is transmitted. A general solution to this problem is to merge the clock and data information using a coding scheme such as BMP, illustrated as the lower stream. In BMP, every bit 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. 8N1 is a simpler encoding scheme in which a transition is guaranteed to occur only at the beginning of each *frame*, a sequence of bits that includes a start bit, stop bit, and eight data bits. Data bits are encoded by the identity function – a 1 is a 1 and a 0 is a 0. Consequently, the clock can only be recovered once in each frame in which the eight data bits are transmitted.

Thus, the central design issue for a *data decoder* is reliably extracting a clock signal from the combined signal. Once the locations of the clock events are known, extracting the data is relatively simple. Although the clock events have a known relationship to signal transitions, detecting these transitions precisely is usually impossible because of distortion in the signal around the transitions, clock jitter, and other effects. The transmitter and receiver of the data do not share a common time base, and hence the estimation of clock events is affected by differences in the reference clocks used. Constant delay is largely irrelevant; however, transition time and variable delay (e.g., jitter) are not. Furthermore, differences in receiver and transmitter clock phase and frequency are significant. Any correctness proof of a BMP (or 8N1) decoder must be valid over a range of parameters defining limits on jitter, transition time, frequency, and clock phase.

The remainder of this paper is organized as follows. In Section 2, the SAL tool set and the k-induction proof technique are described. In Section 3, we present the general SAL models of the transmitter, receiver, and data transmission used in the verifications. The specifics of the BMP model are provided in Section 4, and the small changes necessary for the 8N1 model are in Section 5. The verification of the two protocols is described in Section 6, and concluding remarks follow in Section 7.

2 Introduction to SAL

The protocols are specified and verified in the Symbolic Analysis Laboratory (SAL), developed by SRI, International [7]. SAL is a verification environment that includes symbolic and bounded model checkers, an interactive simulator, integrated decision procedures, and other tools.

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 and guarded transitions. Of the enabled transitions, one is nondeterministically executed at a time. Modules can be composed both synchronously (`||`) and asynchronously (`[]`), 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 one of the modules is nondeterministically chosen to be applied.

The language is typed, and predicate sub-typing is possible. Types can be both interpreted and uninterpreted, and base types include the reals, naturals, and booleans; array types, inductive data-types, and tuple types can be defined. Both interpreted and uninterpreted constants and functions can be specified. This is significant to the power of these models: the parameterized values are uninterpreted constants from some parameterized type.

Bounded model checkers are usually used to find counterexamples, but they can also be used to prove invariants by induction over the state space [8]. SAL supports *k-induction*, a generalization of the induction principle, that can prove some invariants that may not be strictly inductive. By incorporating a *satisfiability modulo theories* decision procedure, SAL can do k-induction proofs over infinite-state transition systems. We use SRI's ICS decision procedure [9], the default SAT-solver and decision procedure in SAL, but others can be plugged in.

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 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 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 inductive invariants. While arbitrary LTL safety formulas can be verified in SAL using k-induction, only state predicates may be used as lemmas in a k-induction proof. Lemmas strengthen the invariant. We have more to say about the proof methodology for k-induction in Section 6.

3 Modeling

In this section, we discuss the general model of physical layer protocols, postponing the details of the BMP and 8N1 protocols to Sections 4 and 5, respectively. We model the protocols using three processes asynchronously composed – a transmitter (tx), a receiver (rx), and a global clock unit (clock). The general arrangement of the three major modules along with the details of the transmitter (`tx`) module are illustrated in Figure 2. The modules `tx` and `rx` model the transmitters and receivers of the protocols; the clock is a modeling artifact that records the passage of the global real time.

```
system : MODULE = clock [] rx [] tx;
```

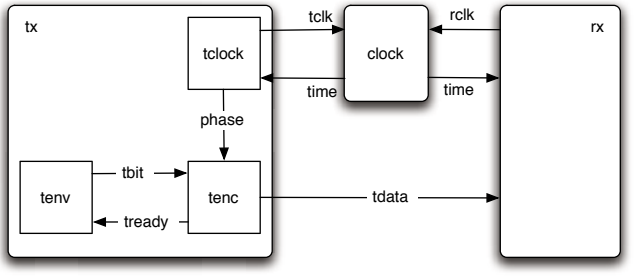

Fig. 2. System Block Diagram

The clock unit provides a single real output variable – `time` – and two inputs, `rclk` and `tclk`, which are the timeout variables of the receiver and transmitter, respectively. The basic idea, described as *timeout automata* by Dutertre and Sorea, is that the progress of time is enforced cooperatively (but nondeterministically) [10, 11]. The receiver and transmitter have *timeouts* that mark the real time at which they will respectively make transitions (timeouts are always in the future and may be updated nondeterministically). Each module is allowed to execute only if its timeout equals the value of `time`. When no module can execute, `clock` updates `time` to be equal to the next timeout. The SAL module below describes the transitions of the global clock.

```
TIME : TYPE = REAL;

clock: MODULE =
  BEGIN
    INPUT  rclk, tclk : TIME
    OUTPUT time : TIME
  INITIALIZATION time = 0
  TRANSITION
    [    time < rclk AND rclk <= tclk --> time' = rclk
      [] time < tclk AND tclk <= rclk --> time' = tclk ]
  END;
```

The transmitter consists of a local clock module (`tclock`) that manages the transmitter's timeout variable, an encoder (`tenc`) module that implements the basic protocol, and an environment module (`tenv`) that generates the data to be transmitted. These modules are synchronously composed.

```
tx : MODULE = tclock || tenc || tenv;
```

The environment and clock modules, defined in Figure 3, are protocol independent and used in both the BMP and 8N1 models. The `tenv` module determines when new input data should be generated and is regulated by `tenc` (which is protocol dependent and described in the following two sections). Whenever `tready` is true, a random datum is selected from `{0, 1}`; otherwise the old datum is preserved (the syntax "`var' IN Set`" defines the value of variable `var` after the transition to be a random value from the set `Set`).

The `tclock` module regulates the `tenc` module. To model periods when the value of a signal is either in transition or uncertain, we divide each period of the transmitter into a settling phase `TSETTLE`, in which the wire might have a value other than `Zero` or `One`, and a stable phase `TSTABLE`, in which the wire may only be `Zero` or `One`. In our models, `TSETTLE` and `TSTABLE` are uninterpreted constants; however they are parameterized, which allows us to verify the models for any combination of settling time and receiver clock error (described subsequently). 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. This means we verify the model under more pessimistic conditions than an actual implementation would face.

```
tenv : MODULE =
  BEGIN
    INPUT  tready  : BOOLEAN
    OUTPUT  tbit   : [0..1]
    INITIALIZATION tbit = 1;
    TRANSITION
      [    tready --> tbit' IN {0,1};
        [] ELSE   --> tbit' = tbit; ]
  END;

PHASE: TYPE = {Stable, Settle};

tclock : MODULE =
  BEGIN
    INPUT  time   : TIME
    OUTPUT tclk   : TIME
    OUTPUT phase  : PHASE
  INITIALIZATION
    phase = Stable;
    tclk  IN {x : TIME | 0 <= x AND x <= TSTABLE};
  TRANSITION
     [    time = tclk AND phase = Stable --> tclk' = time + TSETTLE;
                                             phase' = Settle;
       [] time = tclk AND phase = Settle --> tclk' = time + TSTABLE;
                                             phase' = Stable; ]
  END;
```

Fig. 3. Transmitter Environment and Clock

The decoders are protocol dependent, and are described in the following two sections. Each decoder is composed of a receiver clock, `rclock`, which enforces the timing discipline, and a decoder state machine, `rdec`.

```
rx : MODULE = rclock || rdec;
```

The receiver clock, operating at a multiple of the (nominal) transmitter clock frequency, is used to digitally sample the received signal. These samples are used to detect both transitions and level values which are in turn used to decode the received data. As described in Section 1, the received signal is not purely digital

in nature – there are substantial periods when the received signal is neither 1 nor 0 (i.e., it falls outside of specified voltage bands). Sampling the received signal in or near these transition bands can result in non-deterministic behavior. To model these transition bands, we let a wire have four possible values:

```
WIRE: TYPE = {Zero, One, ToZero, ToOne};
```

Only three values are required, but in practice it is convenient to use the two transition values (`ToZero`, `ToOne`) to store trajectory information. At the receiver we use non-deterministic transition rules of the form `var' IN sample(tdata)` where `sample(wire)` defines the set of possible values obtained when sampling a wire that may be in transition.

```
sample(w : WIRE) : [WIRE -> BOOLEAN] =
  IF (w = ToZero OR w = ToOne) THEN {Zero, One} ELSE {w} ENDIF;
```

The result is always binary but is chosen randomly from the set `{Zero, One}` whenever the wire has a transition value (`ToZero, ToOne`). Thus, the extra data transition values in the model do not "leak" to the receiver.

We do not model constant transmission delay – the settling phase need only capture the variable delay. While our proofs relate the state of the transmitter and receiver at an instant in time, the results hold for a delayed version of the transmitter state in the presence of a constant transmission delay.

As mentioned above, the transmitter clock period is constant (`TSTABLE` + `TSETTLE`). The receiver's clock is based upon this nominal period; however, in order to capture the effects of frequency mismatch and receiver clock jitter, the receiver's timeout period has a random error component that can affect every cycle. We model the transmitter clock as an integer number of unit length ticks (e.g., 16). The receiver clock error is defined on a per-tick basis. For a given nominal timeout of length T ticks, the actual receiver timeout value falls in the range

$$time + T * (1 - ERROR) \leq rclk \leq time + T * (1 + ERROR) ,$$

which we implement in SAL with the following timeout function:

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

As we shall show when we discuss the protocols, the receiver uses different nominal timeout periods depending upon its state. The value of `ERROR` is parameterized by protocol-specific linear inequalities that depend upon `TSETTLE`, which is constrained by the nominal clock periods – together they define the region of reliable operation.

4 Biphase Mark Protocol

Recall from Section 1 that the BMP protocol encodes every bit as two symbols – 00 or 11 for bit 0 and 01 or 10 for bit 1 – guaranteeing a transition at the beginning of every encoded bit (called a *cell*). Our encoder module, illustrated

below, is a straightforward translation where the first two guarded commands implement the basic protocol and are enabled only at end of the "stable" period discussed in Section 3; the third command is enabled only at the end of the "Settle" period and returns the output wire `tdata` to one of the two stable values (`One, Zero`). The `tready` signal, which controls the environment module `tenv` (Section 3), is defined as a function from the current state and phase that is true only when the encoder transitions from state 1 to state 0.

Note that our implementation of the transmitter clock (Section 3) assumes the two halves of a cell are of identical length.[2] To modify the model in order to support asymmetric cells requires a small change to the `tclock` module to make the timeout period state-dependent.

```
tenc : MODULE =
  BEGIN
    INPUT  phase       : PHASE
    OUTPUT tdata       : WIRE
    OUTPUT tstate      : [0..1]
    OUTPUT tready      : BOOLEAN
    INPUT  tbit        : [0..1]
    LOCAL  ttoggle     : WIRE
  INITIALIZATION
    tdata  = One;
    tstate = 1;
  DEFINITION
    tready  = phase = Stable AND tstate = 1;
    ttoggle = IF (tdata = Zero) THEN ToOne ELSE ToZero ENDIF;
  TRANSITION
    [    phase = Stable AND tstate = 1 --> tdata' = ttoggle;
                                           tstate' = 0;
      [] phase = Stable AND tstate = 0 --> tdata' = IF (tbit = 1)
                                           THEN ttoggle ELSE tdata ENDIF;
                                           tstate' = 1;
      [] phase = Settle -->
         tdata' = IF tdata = ToOne THEN One ELSIF tdata = ToZero
                                            THEN Zero ELSE tdata ENDIF; ]
  END;
```

Recall from Section 3 that to model wires in transition, we use a two-phase clock model for the transmitter. At the beginning of a clock cycle, the transmitter either leaves its output `tdata` at its current value (`Zero` or `One`) or initiates a transition to the other stable value by setting `tdata` to the appropriate intermediate value (`ToOne` or `ToZero`). After an appropriate settling time, the wire is restored to a stable value.

The Biphase receiver is composed of two modules – a receiver clock `rclock` which enforces the timing discipline and a decoder state machine `rdec`. The receiver clock enables state transitions when `time = rclk` and it determines the next receiver timeout based upon the decoder's next state (either scanning for an edge or sampling data). Notice that the timeouts are selected randomly from ranges that are bound by the receiver clock error. The values of the various constants are discussed shortly.

[2] Although Moore [2] suggests that there are advantages to an asymmetric cell, this is not generally done in practice because it alters the DC balance and transmitted bandwidth of the signal.

```
rclock : MODULE  =
  BEGIN
    INPUT time   : TIME
    INPUT rstate : [1..2]
    OUTPUT rclk  : TIME
  INITIALIZATION
    rclk IN { x : TIME | 0 <= x AND x < RSCANMAX };
  TRANSITION
    [ time = rclk -->
        rclk' IN IF (rstate' = 2)
                 THEN timeout(time + RSCANMIN, time + RSCANMAX)
                 ELSE timeout(time + RSAMPMIN, time + RSAMPMAX) ENDIF; ]
  END;

rdec  : MODULE =
  BEGIN
    INPUT  tdata  : WIRE
    OUTPUT rdata  : WIRE
    OUTPUT rstate : [1..2]
    OUTPUT rbit   : [0..1]
  INITIALIZATION
    rstate  = 2;
    rdata = One;
    rbit  = 1;
  TRANSITION
    [    rstate = 1  -->
           rdata' IN sample(tdata);
           rbit' = IF (rdata = rdata') THEN 0 ELSE 1 ENDIF;
           rstate' = 2;
      [] rstate = 2  -->
           rdata' IN sample(tdata);
           rstate' = IF (rdata = rdata') THEN 2 ELSE 1 ENDIF; ]
  END;
```

The decoder has two states – in state 2, the decoder scans for an edge while in state 1 it determines the value of the transmitted bit.

We define the nominal transmitter clock period `TPERIOD`, the length of a half-cell, as an constant integer number of units. The nominal number of ticks from the beginning of the cell until the middle of the next half-cell is the constant `TSAMPLE`. In practice, verification of the model is sufficiently fast that it's feasible to run the verification for any choice of `TPERIOD` and `TSAMPLE`.

```
TIME    : TYPE = REAL;
TPERIOD : TIME = 16;
TSAMPLE : INTEGER = 23
```

The receiver runs at two rates – when it is "scanning" for an edge, its clock rate is nominally 1 time unit. After detecting an edge, the receiver waits until the middle of the next half cell. The actual receiver clock (timeout) depends upon the per tick frequency error giving us the four constants used in generating the receiver timeout, as shown in Figure 4.

```
RSAMPMAX : TIME = TSAMPLE * (1 + ERROR);
RSAMPMIN : TIME = TSAMPLE * (1 - ERROR);
RSCANMAX : TIME = 1 + ERROR;
RSCANMIN : TIME = 1 - ERROR;
```

Fig. 4. Receiver Rate Bounds

The limits on ERROR are related by a pair of linear inequalities to TSETTLE. Even if ERROR = 0, there is a practical limit on TSETTLE – reading of the second half-cell value must occur after the settling time but before the end of the cell. This results in the following type constraints for the uninterpreted constants TSETTLE and TSTABLE. Notice their values are both dependent on the value of TPERIOD.

```
TSETTLE : {x : TIME | (0 <= x) AND (x + TPERIOD < TSAMPLE) AND
                      (x + TSAMPLE + 1 < 2 * TPERIOD)};
TSTABLE : TIME = TPERIOD - TSETTLE;
```

Finally, we derive the frequency error bounds. Again, we examine where the reading of the second half-cell value occurs. It must occur after the mid-cell transition, but before the end of the cell. The earliest the reading may occur is RSAMPMIN after the beginning of the cell and the latest the reading may occur is TSETTLE + RSCANMAX + RSAMPMAX. This observation leads to a bound on ERROR.

```
ERROR : {x : TIME | 0 <= x AND TPERIOD + TSETTLE < TSAMPLE * (1-x) AND
                    TSAMPLE * (1+x) + (1+x) + TSETTLE < 2 * TPERIOD};
```

Note that the type for ERROR is parameterized by linear inequalities since TSAMPLE is an interpreted constant.

5 8N1 Protocol

In contrast with BMP, where the receiver clock is re-synchronized on every cell, 8N1, illustrated in Figure 5, is a frame-based protocol where re-synchronization occurs once per frame. Each frame consists of a start bit (0), eight data bits, and one or more stop bits (1), making 10 bits in total.

The 8N1 encoder module is very similar to the Biphase encoder. It contains additional transitions for the special cases of delivering the start and stop bits. The transmitter has ten states. In state 9, the encoder nondeterministically idles, or it sends a start bit by transitioning tdata to Zero and transitioning to state 0. In states 1 through 8, the encoder sends data bits that are generated by the tenv module described in Section 3. However, the interaction between the two modules differs slightly. tenv is directed to generate a new value for tbit when

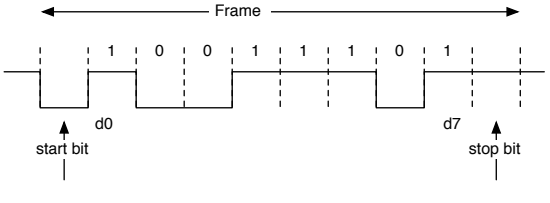

Fig. 5. 8N1 Code

`phase = Stable` and `tstate < 8` – the states during which data bits are sent. In state 8, the encoder generates a stop bit by transitioning `tdata` to `One`.

The 8N1 decoder model is also a simple adaption of the Biphase decoder model. The 8N1 decoder has ten states. In state 10, the decoder samples for the start bit; in state 9, it samples for the stop bit, and in the other states it samples for data bits. As with the BMP decoder, detection of the start of the frame causes the 8N1 decoder to wait until the middle of the first data bit to take its next sample, skipping over the start bit (which has already been detected).

The timing parameters are similar to those in the Biphase model. Again, a nominal transmitter clock period `TPERIOD` is defined. The 8N1 receiver runs at three rates. While scanning for an edge, its nominal bit rate is 1 time unit. After detecting a start bit, the receiver waits until the middle of the first data cell to sample the data. The constraints on sampling and scanning are the same as in Figure 4. To read the remaining data, the receiver waits for `TPERIOD` nominal ticks to sample the middle of the next cell. `RPERIODMAX` and `RPERIODMIN` bound the error of the receiver's clock.

```
RPERIODMAX : TIME = TPERIOD * (1 + ERROR);
RPERIODMIN : TIME = TPERIOD * (1 - ERROR);
```

As with BMP, we can derive bounds on both `TSETTLE` and `ERROR` (we discuss `TSETTLE` shortly). The basic intuition behind parameterizing `ERROR` is that the accumulated error at the point of reading the stop bit must fall in the stable part of the received signal. There are two bounds – the end of the stop bit and the beginning of the stable period of the stop bit. Together, these define bounds on the clock error value. Notice the similarity with the constraint for BMP.

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

ICS is unable to handle the `ERROR` constant parameterized by an uninterpreted `TSETTLE` in this protocol. Thus, we parameterize `ERROR` by the worst-case settling time calculated by hand. For example, if we bound the uninterpreted constant `TSETTLE` such that `0 <= TSETTLE < TPERIOD/4`, then we calculate from the above formula that `0 <= ERROR < 3/151`.

As we mention in Section 1, we discovered significant errors in the analysis in an application note for UARTs [1]. The authors suggest that if `TSTABLE` is `TPERIOD/2` (they call this the "nasty" scenario), then a frequency error of $\pm 2\%$ 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` – otherwise it is impossible to sample the first data bit after the settling period but before the end of the bit period. 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 error of ±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 Verification

Our main goal is to prove that the Biphase and 8N1 decoders reliably extract the data from the combined signal they receive. The statement of the main correctness theorem for 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, and the `X` operator denotes that its argument holds in the next state.

```
BMP_Thm : THEOREM
  system |- G(rstate = 1 AND time = rclk =>
                (time /= tclk) AND (tstate = 1) AND X(rbit = tbit));
```

Informally, suppose that `rstate = 1`, and the time has come for the receiver to make a transition (`time = rclk`). At this time, the wire's value should correspond to the second half of a transmitted cell, and the transmitter should *not* be changing the value of the wire at this time. Furthermore, in the next state – just after the receiver has sampled the wire – the receiver should record the same data bit as the receiver had encoded in that cell. Thus, `rbit = tbit` should hold.

The main theorem for the 8N1 decoder is essentially the same. The only substantial difference is that we prove that the decoder must reliably extract the data over an entire frame (i.e., for states `rstate < 9`).

For both BMP and 8N1, supporting lemmas are necessary to prove the main theorem. 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, and the state-space can be appropriately constrained by an auxiliary lemma based on the counterexample. The following lemmas are built by examining the counterexamples returned from proof attempts for the main theorem and the successive intermediary lemmas.

For both models, we begin by proving three simple preliminary invariants that describe the behavior of the transmitter in both models, irrespective of the receivers. The first invariant, `l0`, states that either the wire is in its settling phase, or it is high or low. Invariants `l1` and `l2` constrain the transmitter's timeout `tclk` during the stable and settle phases: it will never be updated more

than TSTABLE and TSETTLE, respectively. Each lemma is inductive, so it is proved at a depth of one.

```
l0 : LEMMA system |- G(phase = Settle OR tdata = One OR tdata = Zero);
l1 : LEMMA system |- G(phase = Stable => (tclk <= (time + TSTABLE)));
l2 : LEMMA system |- G(phase = Settle => (tclk <= (time + TSETTLE)));
```

One additional lemma is proved for the 8N1 transmitter stating that the stable value of the stop bit is One. This lemma is proved at a depth of 13, using invariants l0 - l2 as lemmas.

The essential part of the proof is an invariant describing 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* [12]. 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 $i \neq j$). Disjunctive invariants are easier to generate iteratively than conjunctive invariants. If a disjunctive invariant fails to cover the reachable states, additional disjuncts can be incrementally added to it (in a conjunctive invariant, additional conjunctions must hold in all the reachable states). Although this is 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 seven disjuncts in the both the BMP and the 8N1 disjunctive invariants. To get an idea about how the invariants are constructed, consider the typical state predicate from the BMP model below. In general, each disjunct states the phase, relates tstate and rstate, and then describes the relative difference between tclk and rclk:

```
... OR ((phase = Settle) AND (rstate = tstate + 1) AND
        (rclk - tclk - TPERIOD > 0) AND
        (tclk + TPERIOD + TSTABLE - rclk > 0)) OR ...
```

Using lemmas l0, l1, l2 described above, the BMP disjunctive invariant is proved at depth five. Using these lemmas and lemma l3, the 8N1 disjunctive invariant is proved at depth three. All that remains is to prove the main theorems, BMP_Thm and the corresponding theorem for the 8N1 decoder. Using the respective disjunctive invariants as lemmas, the former is proved at depth two, and the latter is proved at depth six.

7 Discussion

We have described a general model of physical layer data transmission, and we have used this model to verify the correctness of BMP and 8N1 under parameterized timing constraints. We also present an error in a published application

note discovered during the verification. The verification is highly-automated using k-induction implemented with a SMT decision procedure and a bounded model checker in SAL.

As mentioned in Section 1, a referee suggested an alternative approach that fully parameterizes the BMP verification. The central idea is to leave `TPERIOD` and `TSAMPLE` as uninterpreted constants and then constrain the times when the receiver scans and samples directly in terms of `TPERIOD`, `TSETTLE`, and `TSTABLE`:

```
RSCANMIN : {x : TIME | 0 < x};
RSCANMAX : {x : TIME | RSCANMIN <= x AND x < TSTABLE};
RSAMPMIN : {x : TIME | TPERIOD + TSETTLE < x};
RSAMPMAX : {x : TIME | RSAMPMIN <= x AND x < 2 * TSTABLE - RSCANMAX};
```

Thus, no error term is necessary, and the verification is fully parameterized. The BMP specification otherwise remains the same, and its proof of correctness succeeds using the same lemmas, proved at the same depth. The constraints on the error can be easily recovered by hand from the fully parameterized verification by replacing the constants `RSCANMIN`, `RSAMPMIN`, and `RSCANMAX` in the type definitions above by their definitions from Figure 4. The referees also point out that because the error bounds are not explicit in this fully parameterized model in SAL, it is less general than the verification by Vaandrager and de Groot using mechanical theorem-proving [6]. Although the parameterized verification in SAL with error bounds recovered by hand is neither fully automated nor machine-checked, it is a more economical approach than mechanical theorem-proving.

As compared to real-time model checking, our SAL verification appears to be more parameterized than the verifications reported by Ivanov and Griffioen in Hytech [4] and at least as parameterized as the one suggested (but not described) by Henzinger, Preussig, and Wong-Toi, also using Hytech, in which the verification is fully automatic [5]. The tool TReX has similar capabilities to HyTech [13]. Note, however, that SAL is not specifically a real-time model checker.

The verification technology employed in SAL is recent, and only a few non-trivial verifications using it exist [11, 14]. This work, along with recent work by one of the authors, is the first known application of these techniques to the verification of physical-layer protocols [15].

Acknowledgments. We thank Leonardo de Moura, John Rushby, and our three anonymous TACAS referees for their careful comments and suggestions.

References

1. 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`.
2. 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.

3. 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)*, pages 88–98. IEEE Computer Society Press, 1998.
4. S. Ivanov and W. O. D. Griffioen. Verification of a biphase mark protocol. Technical Report CSI-R9915, University of Nijmegen Computing Science Institute, 1999.
5. 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.
6. 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.
7. 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.
8. 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.
9. Leonardo de Moura, Sam Owre, Harald Ruess, John Rushby, and N. Shankar. The ICS decision procedures for embedded deduction. In *2nd International Joint Conference on Automated Reasoning (IJCAR)*, volume 3097 of *LNCS*, pages 218–222, Cork, Ireland, July 2004. Springer-Verlag.
10. Bruno Dutertre and Maria Sorea. Timed systems in SAL. Technical Report SRI-SDL-04-03, SRI International, 2004.
11. 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.
12. 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.
13. Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. TReX: A tool for reachability analysis of complex systems. In *Computer-Aided Verification, CAV'01*, pages 368–372, London, UK, 2001. Springer-Verlag.
14. 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.
15. Geoffrey M. Brown. Verification of a data synchronization circuit for all time. Unpublished, 2005.