A Formal Model of Clock Domain Crossing and Automated Verification of Time-Triggered Hardware

Julien Schmaltz[1]
Institute for Computing and Information Sciences
Radboud University Nijmegen, The Netherlands
Email: julien@cs.ru.nl

***Abstract*— We develop formal arguments about a bit clock synchronization mechanism for time-triggered hardware. The architecture is inspired by the FlexRay standard and described at the gate-level. The synchronization algorithm relies on a specific value of a counter. We prove or disprove values proposed in the literature. Our framework is based on a general and precise model of clock domain crossing, which considers metastability and clock imperfections. Our approach combines this model with the state transition representation of hardware. The result is a clear separation of analog and digital behaviors. Analog considerations are formalized in the logic of the Isabelle/HOL theorem prover. Digital arguments are discharged using the NuSMV model checker. To the best of our knowledge, this is the first verification effort tackling asynchronous transmissions at the gate-level.**

I. INTRODUCTION

Embedded systems comprise software applications, compilers, real-time operating systems, processors with memory management units and devices, as well as communication architectures. These different components form the layers of a stack. The top layer and the most abstract one is occupied by software applications. Going down in the abstraction, software applications together with an operating system are compiled into machine code and run on top of processing units and memories. Their gate level description constitutes the lowest layer of the stack. In the late 80's, Bevier *et al.* [2] demonstrated a first "stack proof", *i.e.* a proof of a simulation theorem between the top layer and the bottom layer. The application of this approach to realistic embedded systems remains a challenge [12]. Computer systems are often *distributed*. One verified stack is not enough. One needs to prove correctness of stacks and their communications.

These communications are inherently asynchronous as in practice clocks of interconnected devices are not constant over time. This clock distortion induces possible metastable states of registers. The proof of *distributed stacks* requires the analysis of these phenomena at the gate-level. Moreover, in the context of realistic worst case execution time analysis, it is also necessary to know the duration of the transmission. Towards this end, a pencil and paper proof of an entire distributed systems was developed [1]. From this study, we formalized in the logic of Isabelle/HOL [13] the bit transmission between independently clocked registers, assuming precise timing parameters and metastability [15].

The contribution of this paper is an important extension of these theoretical results and the definition and the application of a methodology for the verification of time-triggered hardware. We extend the Isabelle theory to allow for long-term jitter. This relieves the previous hypothesis about constant clock periods. The outcome is a general model of clock domain crossing. This model is combined with the semantics of transition systems used to describe hardware designs. This identifies constraints on the *digital design* that guarantee proper transmission assuming *analog behaviors*. These constraints can be solved by decision procedures. We use the integration in Isabelle of the model checker NuSMV [16]. We demonstrate this methodology on the verification of a time-triggered bus interface inspired by the FlexRay standard [5] and described at the gate-level. The statement of our theorem also includes the duration of the transmission. Our analysis identifies precisely the possible values of one crucial parameter of the bit clock synchronization mechanism. This proves and disproves values proposed in the literature.

The paper is structured as follows. In the next Section, we present our general model of clock domain crossing. We show in Section III how we use this model for hardware design verification. Section IV describes the time-triggered interface, which is verified in Section V. Related work is discussed in Section VI. Section VII concludes the paper.

II. A FORMAL MODEL OF CLOCK DOMAIN CROSSING

A. Signals and Clocks

Time is represented by the nonnegative reals ($R_{\geq 0}$). A signal s is represented by a function $s(t)$ from *real* time t to $\{0, 1, \Omega\}$: 1 and 0 mean "high" and "low" voltages; Ω means any voltage.

The clock period of unit u is noted τ_u. Periods are different from zero. The *date* of the c^{th} rising edge of clock clk_u of unit u is noted $e_u(c)$. It equals the product of c with the clock period: $e_u(c) = c \cdot \tau_u$.

Function e gives the ideal date of edges. In practice, it is impossible to guarantee constant clock periods. We assume that all clock periods of any clock deviate at most by a percentage δ of a reference clock period. This reference clock

[1]Part of this work was carried out while the author was affiliated with the University of Saarland, Saarbrücken, Germany. This work was funded by the German Federal Ministry of Education and Research (bmb+f) in the framework of the Verisoft project under grant 01 IS C38.

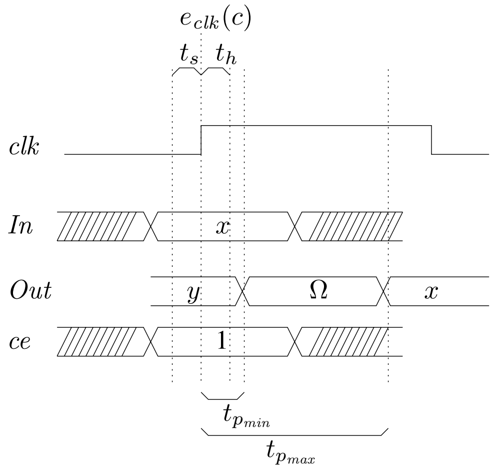

Fig. 1. Behavior of the register w.r.t clock edge c

is named clk_{ref}. Its period is τ_{ref}. Formally, we assume:

$$\Gamma_u \equiv 1-\delta \leq \frac{\tau_u}{\tau_{ref}} \leq 1+\delta \tag{1}$$

We are not interested in the deviation at each cycle, but in the number of cycles in which the number of ticks of two independent clocks may differ by at most one. Let π be that number. In this interval, the maximum drift between two clocks is obtained between the slowest and the fastest clocks allowed by Equation 1. Consequently, the ratio between the minimum and the maximum clock periods defines the lower bound of the drift. From Equation 1 and defining $\pi = \frac{1-\delta}{2\cdot\delta}$, we prove the following lemma:

Lemma 1: **Bounded Clock Drift.**

$$\Gamma_i \wedge \Gamma_j \rightarrow \frac{\pi}{\pi+1} \leq \frac{Min(\tau_i,\tau_j)}{Max(\tau_i,\tau_j)}$$

This property is preserved for any number less than π.

B. Analog Registers

Open intervals are represented using open squared brackets. Shifting an interval is noted $x+[y:z]$ instead of $[x+y:x+z]$.

Registers consist of one input signal *In*, one clock signal *clk*, one control signal *ce* and one output signal *Out* (Fig. 1). A new value (x) is input to the register at cycle c, which is defined by interval $[e_u(c) : e_u(c+1)[$. During minimum propagation delay $t_{p_{min}}$ the output signal equals previous value y. Because the control signal is high, the output oscillates (*i.e.* is Ω) before stabilizing at new value x. If the control signal is low, the output does not oscillate and keeps its old value y.

If the input or the control signals do not have a constant value during the *setup time* (noted t_s) before edge c and during the *holding time* (noted t_h) after edge c, the register may become metastable. This means that its output may still be Ω after $t_{p_{max}}$. When this metastable state is resolved, the register reaches a defined value. Metastability cannot be avoided [9]. We assume that this resolution time is less than one clock period. Before giving our formal definition of analog registers, we define a few concepts.

```
$_aR_u(c, clk_u, ce_u, In_u, Out_u^0) \triangleq$
**if** $c = 0$ **then** $\lambda t.Out_u^0$ **else**
  **if** $\left\{\begin{array}{l} stadep(e_u(c)-t_s, e_u(c)+t_h, ce_u) \\ \wedge\ stadep(e_u(c)-t_s, e_u(c)+t_h, In_u) \end{array}\right.$ **then**
    **if** $ce_u(e_u(c)) = 1$ **then**
      $\lambda t.\left\{\begin{array}{lcl} _aR_u(c-1,\ldots)(e_u(c)) & : & t \in e_u(c)+]0:t_{p_{min}}] \\ \Omega & : & t \in e_u(c)+]t_{p_{min}}:t_{p_{max}}[ \\ In_u(e_u(c)) & : & t \in e_u(c)+[t_{p_{max}}:\tau_u] \\ \Omega & : & t \notin ]e_u(c):e_u(c+1)] \end{array}\right.$
    **else** ;; keep old value
      $\lambda t.\left\{\begin{array}{lcl} _aR_u(c-1,\ldots)(e_u(c)) & : & \forall t \in ]e_u(c):e_u(c+1)] \\ \Omega & : & t \notin ]e_u(c):e_u(c+1)] \end{array}\right.$
    **endif**
  **else** ;; metastability
    $\lambda t.\left\{\begin{array}{lcl} _aR_u(c-1,\ldots)(e_u(c)) & : & t \in e_u(c)+]0:t_{p_{min}}] \\ \Omega & : & t \in e_u(c)+]t_{p_{min}}:\tau_u[ \\ x \in \{0,1\} & : & t = e_u(c+1) \\ \Omega & : & t \notin ]e_u(c):e_u(c+1)] \end{array}\right.$
  **endif**
**endif**
```

Fig. 2. Definition of Analog Registers

The *metastability window* w.r.t. edge c of register u (noted MW_u^c) is defined by interval $e_u(c)+[-t_s:t_h]$.

A signal s is stable during time interval $[t_1:t_2]$ if it holds the value at time t_1 until time t_2. A signal s has a defined value during time interval $[t_1:t_2]$ if it never equals Ω during that interval. Formally, this is expressed as follows[1]:

$$stadep(t_1,t_2,s) \triangleq \exists b \in \{0,1\}, \forall t \in [t_1:t_2], s(t) = b$$

The formal definition of the analog behavior is given by function $_aR_u$ (Fig 2). We are interested in the output value of a register for all real times during cycle c. Function $_aR_u$ takes as arguments a cycle c, a clock signal clk_u, a clock enable control signal ce_u, an input signal In_u, and the initial output value Out_u^0. It generates a signal.

If no setup or holding time violation occurs, the register behaves normally. If the control signal is low, the register keeps its old value (at the previous cycle $c-1$); if the control signal is high the output keeps its previous value during $t_{p_{min}}$, then oscillates (*i.e.* is Ω) to finally reach its final value at time $e_u(c)+t_{p_{max}}$. If input signal In_u or control signal ce_u is not stable and defined during the metastability window, the register becomes metastable. The output equals the previous computation until $t_{p_{min}}$ (included) and Ω afterwards. At the end of the cycle, metastability has been resolved and the output equals an arbitrary but defined value. To make the function total, Ω is output for all times outside the cycle. To alleviate our notation, we shall write $_aR_u^c$ instead of $_aR_u(c, clk_u, ce_u, In_u, Out_u^0)$.

Formally, all timing parameters ($t_h, t_s, t_{p_{min}}, t_{p_{max}}$) are percentages. We assume that their sum is less than 1. Their value depends on the local clock period. In the remainder of this

[1]Note: *stadep* means **sta**ble, **de**fined, **p**redicate

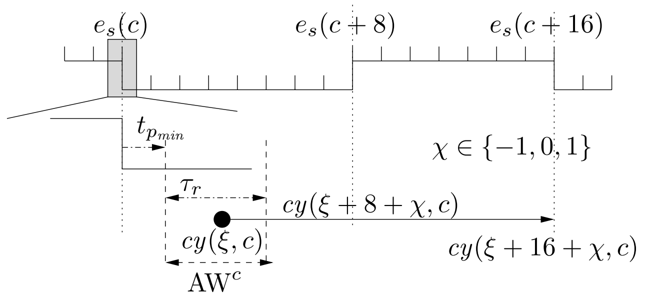

Fig. 3. Relating Receivers and Senders

paper, propagation delays are relative to the sender clock and setup and holding times are relative to the receiver clock period.

C. Relating Receivers to Senders

The relation between a sender and a receiver is pictured in Fig. 3. A sender starts sending three different bits at edge $\sharp c$, $\sharp c+8$, and $\sharp c+16$. If we take a closer look around edge c, the sender output is not modified before $e_s(c)+t_{p_{min}}$, when it moves from y to Ω (see Fig 1 for more details). If a receiver samples before that date, it will get the old value. In contrast, sampling *strictly after* that date will affect the receiver, either it will get metastable, or it will detect a new value. Let ξ be the first receiver edge after $e_u(c)+t_{p_{min}}$. As this edge is the first one to be affected by the behavior of the sender, we denote it as "marked with edge c", noted $cy(\xi,c)$. If there is no ambiguity, we may drop the first argument. Edge ξ occurs in time interval $e_s(c)+t_{p_{min}}+]0:\tau_r]$. This interval defines the "affected window w.r.t. edge c", noted AW^c. Formally, we have the following definition:

Definition 1: **Affected Cycle.** $cy(\xi,c) \equiv e_r(\xi)+t_h \in \mathrm{AW}^c$

Let us suppose that edge ξ is in AW^c.If the sender sends another bit within a number of cycles (α) less than our bound π, the corresponding affected window may be seen by the receiver with a potential error of one cycle, *i.e.* at $e_r(\xi+\alpha\pm 1)$. This means that subsequent marks are known with the same error. Fig 3 shows these marks for $\alpha=8$ and $\alpha=16$.

Formally, we have the following theorem:

Theorem 1: **Regular Affected Windows**

$$\begin{array}{ll} & \Gamma_{clk_r} \wedge \Gamma_{clk_s} \wedge 0<\alpha\leq\pi \wedge cy(\xi,c) \\ \rightarrow & \bigvee_{\chi\in\{-1,0,1\}} e_r(\xi+\alpha+\chi) \in \mathrm{AW}^{c+\alpha} \end{array}$$

Proof. We do a case analysis depending on the position of $\xi+\alpha$ regarding the affected window $\mathrm{AW}^{c+\alpha}$. If $e_r(\xi+\alpha)$ is (1) before $\mathrm{AW}^{c+\alpha}$, we prove $e_r(\xi+\alpha+1)\in\mathrm{AW}^{c+\alpha}$; (2) within $\mathrm{AW}^{c+\alpha}$, this proves the obvious case where $\chi=0$; (3) after $\mathrm{AW}^{c+\alpha}$, we prove $e_r(\xi+\alpha-1)\in\mathrm{AW}^{c+\alpha}$. □

D. Safe Sampling Window

In case of metastability, we always assume resolution to the negation of the expected input. Thus, an extra delay may be introduced. This is represented by the *metastability factor* (β). Metastability can only happen if the affected cycle (minus the setup time) appears while the sender output is undefined, *i.e.* before $e_s(c)+t_{p_{max}}$. In this case, the metastability factor returns 1. It returns 0 otherwise. Formally, the *metastability factor* is a function, which takes as arguments cycles ξ and c, and two clocks.

Definition 2: **Metastability Factor.**

$$\begin{array}{l} \beta(\xi,c,clk_s,clk_r) \triangleq \\ \quad \textbf{if } e_r(\xi)-t_s \leq e_s(c)+t_{p_{max}} \textbf{then } 1 \textbf{ else } 0 \end{array}$$

To alleviate the notation, we shall write β^ξ_c instead of $\beta(\xi,c,clk_s,clk_r)$.

To ensure that the receiver will not always sample Ω's, the sender keeps its output constant for several cycles (say k cycles). Consequently, there is only one metastability window and if k is big enough there exists a "sweet spot" in which the receiver can sample safely. Formally, the safe sampling window of length k w.r.t. cycle c (noted SSW^c_k) is denoted by interval $[e_u(c)+t_{p_{max}} : e_u(c+k+1)+t_{p_{min}}]$.

We prove that under our drift hypothesis, SSW^c_k entails up to $k-1$ receiver cycles (or k edges), even in case of metastability.

Theorem 2: **SSW's are large enough.**

$$\begin{array}{ll} & \Gamma_{clk_r} \wedge \Gamma_{clk_s} \wedge cy(\xi,c) \wedge n+1 \leq k \leq \pi \\ \rightarrow & \forall l \leq n, cy(\xi+\beta^\xi_c+l)+[-t_s:t_h] \in \mathrm{SSW}^c_k \end{array}$$

E. Clock Domain Crossing Correctness

Our main theorem proves that sampling in a safe sampling windows is correct. We assume that the sender creates a safe sampling window of length k, control and input bits must be stable and defined during all sender metastability windows, clock drift is bounded. We assume that cycle ξ is in SSW^c_k.

Theorem 3: **Correct Transfer.**

$$\begin{array}{ll} & \Gamma_r \wedge \Gamma_s \wedge cy(\xi,c) \text{ (*bounded drift, affected cycle*)} \\ \wedge & \text{(* } \mathrm{SSW}^c_k \text{ *)} \\ & ce_s(e_s(c)) = 1 \wedge \forall l \in [1:k], ce_s(e_s(c+l) = 0 \\ \wedge & c>0 \wedge n+1 \leq k \leq \pi \\ \wedge & \forall l \in [0:k+1], \text{(*input *)} \\ & stadep(e_s(c+l)-t_s, e_s(c+l)+t_h, In_s) \\ \wedge & \forall l \in [0:k+1], \text{(*control*)} \\ & stadep(e_s(c+l)-t_s, e_s(c+l)+t_h, ce_s))) \\ \wedge & \text{(*analog connection*)} \\ & \forall c, In_r = {}_aR_s(c, clk_s, ce_s, In_s, Out^0_s) \wedge \forall t, ce_r(t)=1 \\ \wedge & e_r(\xi)+[-t_s:t_h] \in \mathrm{SSW}^c_k \text{ (* good cycle *)} \\ \rightarrow & {}_aR^\xi_r(e_r(\xi+1) = In_s(e_s(c)) \end{array}$$

Proof. First, Theorem 2 gives us the position of receiver edges in the safe sampling window. Then, we case split on the position of the metastability window around cycle ξ. We set two reference points: $e_s(c+1)$ and $e_s(c+1+k)$. We prove the conclusion for 5 cases depending on the position of the metastability window regarding these points. □

III. Analog Transfer in a Digital World

Our model of clock domain crossing mentions analog entities only. The semantics is based on functions and a dense representation of time. Ultimately, we want to use this model to verify hardware designs described in another semantics, which is based on a discrete notion of time and transition functions. Before describing our approach, we define type conversion functions and rephrase Theorem 3 to match bits and not signals.

A. Type Conversions

The conversion from bit lists to signals is done by function γ. We do not give a particular definition to this function. We only assume that it produces a signal such that during the metastability window around cycle $i+1$, it outputs the value with index i in the bit list. This property is defined by predicate $bv2sp$:

$$bv2sp(\gamma, l_u) \equiv \forall t, i, t \in \mathrm{MW}_u^{i+1} \rightarrow \gamma(l_u) = l_u[i]$$

The conversion from signals to bits is done by function ζ, which takes as input a signal and a time. If the value of the signal at that time is a bit value, then this value is returned. Otherwise, *some* bit value is returned.

$$\zeta(s,t) \triangleq \textbf{if } s(t) \in \{0,1\} \textbf{ then } s(t) \textbf{ else } x \in \{0,1\}$$

B. Transfer Correctness in the Digital World

Let lists ce_s and In_s be the bit lists containing values given to the analog sender register. If they both satisfy predicate $bv2sp$, list element $ce_s[c-1]$ or $In_s[c-1]$ corresponds to the bit value given to the sender analog register at time $e_s(c)$.

Theorem 3 is embedded into a digital context in the following statement. We assume that (a) clock drift is bounded; (b) function γ correctly translates bit lists ce_s and In_s; (c) the digital control bits are high once and then low k times. Analog hypotheses are concerned with the connection of the sender with the receiver and the clock drift. Obviously, they cannot be "digitalized". Under these assumptions, we prove that the "digitalized" output of the **analog** receiver register equals the **digital** input of the sender at cycle c. In the remainder of this paper, we will denote the conjunction of the hypotheses of this theorem by $\mathcal{H}$.

Theorem 4: **Back to the Digital World.**

$$\begin{array}{ll}
 & \Gamma_r \wedge \Gamma_s \wedge n+1 \leq k \leq \pi \wedge cy(\xi, c) \\
 & \text{ (*bounded drift, cy(c)*)} \\
\wedge & \forall c, In_r = {}_aR_s(c, clk_s, \gamma(ce_s), \gamma(In_s), Out_s^0) \\
\wedge & \forall t, ce_r(t) = 1 \text{ (*analog link*)} \\
\wedge & bv2sp(\gamma, ce_s, clk_s) \wedge bv2sp(\gamma, In_s, clk_s) \\
 & \text{ (*modeling hypotheses*)} \\
\wedge & ce_s[c+\alpha-1] = 1 \wedge c > 0 \\
\wedge & \forall l \in [1:k], ce_s[c+l-1] = 0 \text{ (*sender OK*)} \\
\wedge & e_r(\xi) + [-t_s : t_h] \in \mathrm{SSW}_k^c \text{ (* good cycle *)} \\
\rightarrow & \\
 & \zeta({}_aR_r^{\xi}, e_r(\xi+1)) = In_s[c-1]
\end{array}$$

Proof. By definition of predicate $bv2sp$, $\gamma(In_s)$ and $\gamma(ce_s)$ are *stadep* for the required cycles. Theorem 3 concludes. □

C. Principle and Soundness

Fig. 4 illustrates our integration of our analog results in the analysis of digital designs. Our clock domain crossing model (CDC) is shown inside the dashed box. The remainder of the figure corresponds to digital designs that are actually used to synthesize hardware. These designs are not modified. Our model is simply inserted as a filter of the receiver inputs.

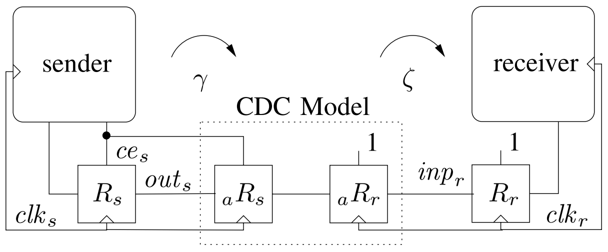

Fig. 4. Mixing Analog and Digital Signals

Digital designs are represented by their transition function, one application of which represents the computation of one clock cycle. The sender and the receiver parts are analyzed separately. The analysis of the sender does not need any analog arguments. It mainly consists of the proof that sender output out_s follows a specific frame format. The analysis of the receiver is done assuming correctness of the sender and that the connection of receiver input inp_r is done through our CDC model. We write that an element s_u of unit u has bit-value x at cycle c - *i.e.* after c applications of the transition function - as $s_u^c = x$. Formally, we assume that the value of input bit inp_r at hardware cycle c equals the output value of register ${}_aR_r$ at the date of edge $c+1$:

$$\forall c, inp_r^c = \zeta({}_aR_r^c, e_r(c+1)) \quad (2)$$

The left hand side represents the value that should be in register R_r at $c+1$. As the analog register is not part of the transition function of the receiver, one application of the latter compensates this difference. The right hand side is always a defined value. This Equation only holds when R_r is not metastable. As discussed in the next sub-section, it is always the case when we use Equation 2.

D. Proof Method

Our proof method uses Theorems 2, 1, 4 and Equation 2. We also need a mark $cy(\xi, c)$ which connects receiver cycle ξ with the beginning of a safe sampling window started at sender cycle c. From this mark, we obtain from Theorem 2 that there are $n+1$ receiver edges in the safe sampling window. These edges may be shifted by one cycle depending on the resolution of metastable states. Then, we obtain from Theorem 4 that register ${}_aR_r$ outputs out_s^c at the date of these edges. Because we are outside metastable behaviors, we obtain from Equation 2 inputs for the receiver. Once these inputs are known, the analysis is back to the digital world and decision procedures apply. We obtain subsequent inputs using a similar reasoning and the marks given by Theorem 1.

IV. Time-Triggered Bus Interfaces

We present an implementation of a time-triggered bus interface inspired by the FlexRay standard [5] for safety-critical automotive applications. This design has already been

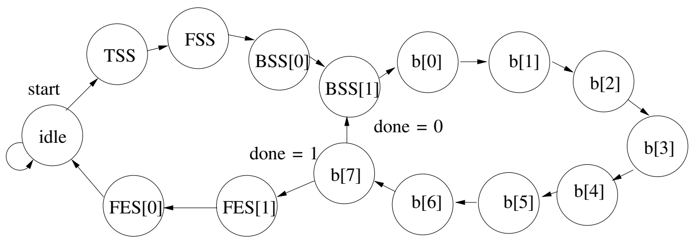

Fig. 5. Control Automaton

presented [1]. It can be translated to Verilog [14] and synthesized on FPGA. It will be available at the Verisoft Repository[2].

A. Protocol Overview

We consider an arbitrary number of units connected through a shared bus. Each unit can send and receive messages. Idle units put one's on the bus. Let $\mathrm{TSS} = 0$ be the transmission start sequence, $\mathrm{FSS} = 01$ be the frame start sequence, $\mathrm{BSS} = 10$ be the byte start sequence and $\mathrm{FES} = 01$ be the frame end sequence. Let $\langle a, b\rangle$ be the concatenation of bit vectors a with b. A message m of l bytes is encapsulated into a frame $f(m)$ with the following format:

$$f(m) = \langle \mathrm{TSS}, \mathrm{FSS}, \mathrm{BSS}, m[0], \ldots, \mathrm{BSS}, m[l-1], \mathrm{FES}\rangle$$

Each bit of a frame is sent 8 times.

B. Sender Module

The sender implements the protocol by the control automaton in Fig. 5. As specified by the protocol, in each state the corresponding bit is generated 8 times. The sender is connected with the shared bus through a register named R_s with control enable bit ce_s. This paper focuses on the verification of message reception. We do not detail the sender implementation any further.

C. Receiver Implementation: Bit Clock Synchronization

The receiver module implements the same state-automaton as the sender. In each state, the receiver is expecting to receive the corresponding bit of the frame. Beside the automaton, the relevant part of this receiver consists of the input stage pictured in Fig. 6. The first two registers form a "synchronizer" used to remedy to metastability. A five majority vote is performed. Signal $sync$ is used to detect the synchronization sequence BSS. It is high if and only if the current voted bit does not equal its previous value and the state automaton is either in state *idle* or in state BSS[1]. When $sync$ is high counter cnt is reset to 000 in the next cycle. The state automaton is clocked by signal *strobe*, which is high each time the counter reaches value 010 and the automaton is not synchronizing, *i.e.* when signal $sync$ is low. Each time *strobe* is high, the voted bit is stored in shift register BYTE. When the last bit has been stored (*i.e.* automaton is in state b[7]) and signal *strobe* is high, signal $rb.we$ turns high and BYTE is written to the main receiver buffer.

[2]http://www.verisoft.de/VerisoftRepository.html

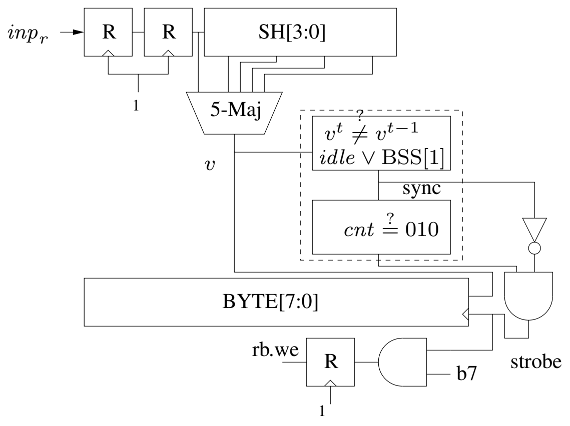

Fig. 6. Input Stage

This implementation uses the synchronization mechanism described in the FlexRay standard. Synchronization is performed by resetting the counter when receiving the BSS sequence. Our implementation differs slightly from the FlexRay guidelines. The standard suggests to reset the counter to 010 and to strobe when it reaches 101. We reset to 000 and strobe at 010. So, we strobe one cycle earlier. In [1], the counter is reset to 000 and *strobe* is high when *cnt* is 100.

V. Formal Verification

A. Sender Correctness

The sender is proven to effectively generate each bit 8 times. This discharges the digital hypotheses of Theorem 4. Formally, this is defined as follows:

Definition 3: **Correctness of ce_s.**

$$\begin{array}{lcl} \mathrm{WF}_{ce}(ce_s, L, k, c) & \equiv & \forall i < L, ce_s[c + 8\cdot i] = 0 \\ & \wedge & \forall j \in [1:k], ce_s[c + 8\cdot i + j] = 0 \end{array}$$

We prove that the sender generates frames with the specified format. For the purpose of this paper, we are only concerned with synchronization bits, *i.e.* the BSS sequence. This is expressed by the following predicate:

Definition 4: **Partial Correctness of In_s.**

$$\mathrm{WF}_{In}(In_s, L, c) \equiv \forall i < L, \forall y \in [0:7] \left\{ \begin{array}{ll} & In_s[c + 80 * i + 16 + y - 1] = 1 \\ \wedge & In_s[c + 80 * i + 24 + y - 1] = 0 \end{array} \right.$$

B. Receiver Correctness Statement

The correctness of a time-triggered interface is achieved if for the transmission of any byte of a frame there exists a hardware cycle from which the interface recovers that byte. This requires the proof that (1) depending on the position of the BSS[0]-mark ($cy(\mathrm{BSS}[0])$) the state automaton strobes the right voted bits, and (2) this happens soon enough to match the sender output. The first statement expresses that the automaton indeed synchronizes and the second that this synchronization is good enough to sample properly. The final proof uses theses two statements to prove by induction over the number of bytes that the whole frame is recovered. We assume that the state automaton is initially is state "idle" and that the first mark

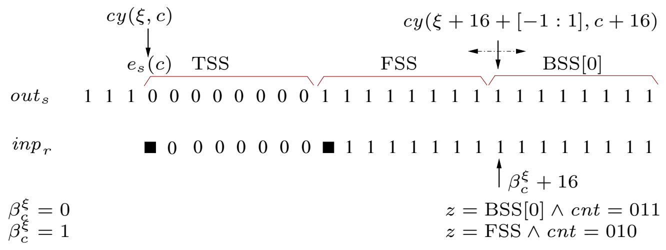

Fig. 7. Initial Transmission Phase

is known. The final statement is Theorem 5 below. The first line of the conclusion states that for all bytes we detect a mark for BSS[0]. The second lines states that at this cycle the control automaton could be in different states with different values of its counter. This different values come from previous bytes which may have suffered from clock drifts. The last lines state that under this uncertainty the automaton samples byte $\sharp i$ properly. This theorem also proves lower and upper bounds on the time at which the last byte is recovered. Using the mark of the conclusion, these bounds can be expressed as functions of the reference clock and the time ($e_s(c)$) when the first bit is put on the bus by the sender.

Theorem 5: **Transmission Correctness.**

$$\begin{array}{l} \mathcal{H} \wedge z^\xi = \mathit{idle} \wedge cy(\xi, c) \\ \forall c, inp_r^c = \zeta({}_aR_r^c, e_r(c+1)) (\text{* A/D mix*}) \\ \text{WF}_{ce}(ce_s, L, k, c) \wedge \text{WF}_{In}(out_s, L, c) \end{array}$$

$$\begin{array}{lll} \rightarrow & & \\ \forall i < l, & \exists \nu, & cy(\nu, c + 16 + 80 \cdot i) \text{ (* mark *)} \\ \wedge & & z^\nu = (\text{FSS} \vee b[7]) \wedge cnt^\nu = (001 \vee 010) \\ & \vee & z^\nu = \text{BSS}[0] \wedge cnt^\nu = (011 \vee 100) \\ \wedge & & z^{\nu+78+w} = \text{BSS}[0] \wedge cnt^{\nu+78+w} = 010 \\ \wedge & & \text{BYTE}^{\nu+79+w} = \langle out_s^{c+16+80\cdot i+8\cdot(j+2)} \rangle \end{array}$$

$$\text{where } w \in [0:3] \text{ and } j \in [7:0]$$

The proof is done by induction over i. For space reason, we only detail the base case, which is pictured in Fig. 7. The first two lines show the output of the sender and how it is seen by the receiver. Black boxes indicate possible metastability.

Because of clock drift, the BSS[0]-mark may appear on the receiver side 15, 16 or 17 cycles after ξ. There is a potential metastability at cycle ξ. Depending on the value reached after resolution – that is depending on the value of β_c^ξ – the receiver automaton reaches different state and counter values when the BSS[0]-mark is detected. In the figure, we show these values at $\beta_c^\xi + 16$, where the automaton is either in state BSS[0] with a counter at 011 or in state FSS with a counter at 010. In the following sections, we prove that the receiver recovers a byte for all these possibilities.

C. Traversing Synchronization Edges

Our reasoning is illustrated in Fig. 8. The first two lines show the output of the sender and how it is seen by the receiver. Black boxes indicate possible metastability. Question marks are used to denote unknown values.

We fix the initial step of the lemma to match the date of the detection of the BSS[0]-mark. We consider the case where the receiver is in state BSS[0] with a counter value at either 011 or 100.

According to Theorem 1 and assuming that the BSS[0]-mark is known, the BSS[1]-mark has three possible dates. The potential metastability around that edge has the same three dates. We consider bits sampled by the receiver at these dates unknown. Another source of uncertainty resides in factor β. It is already represented by metastability. Therefore, at most three bits are unknown. Depending on the values of these three bits, the automaton will spend more or less time in the states of BSS. There is synchronization if the lower and the upper bound on this number of cycles allow proper sampling. This bounds are defined by the next lemma which imposes that the automaton reaches state $b[0]$ with counter value 011 in at least 15, and at most 18 cycles.

Let t be the date of the affected cycle of BSS[0]. If the three unknown bits are 0 (see line 3 in Fig. 8), signal $sync$ is high at $t + 7 + 4 = t + 11$. The counter is reset, and signal $strobe$ is high at $t + 11 + 3 = t + 14$. In the next cycle, the automaton reaches state $z^{t+15} = b[0]$. For any lower value of the counter, the automaton will reach this state earlier.

If the unknown bits are 1, signal $sync$ is high at $t + 10 + 4 = t + 14$. If the counter was 100 initially, then it has reached value 010 and $strobe$ is high. At the same time, signal $sync$ is high, the automaton stays in BSS[0] and the counter is reset. At cycle $t + 17$, $strobe$ is high and the automaton reaches $b[0]$ with a correct counter value at $t + 18$. For any larger value, the automaton requires more cycles to reach this state.

From Theorem 4 and considering the possible values of β_c^ξ, we know for sure 6 bits of BSS[0] (from $t + 1$ to $t + 6$) and 6 bits of BSS[1] (from $t + 9$ to $t + 14$). Assuming that only these input values are known, the rest of the proof is purely digital. It is expressed by the following lemma, the proof of which is fully automatic.

Lemma 2: **From BSS[0] to BSS[1].**

$$\begin{array}{ll} & \forall u \in [0:6], inp^{t+1+u} = 1 \text{ (* 6 bits of BSS[1]*)} \\ \wedge & \forall v \in [0:6], inp^{t+9+v} = 0 \text{ (* 6 bits of BSS[0]*)} \\ \wedge & ((z^t = \text{BSS}[0] \wedge cnt^t = 010 \vee 011) \text{ (* states and *)} \\ \vee & (z^t = \text{FSS} \wedge cnt^t = 001 \vee 010)) \text{ (*misalignment*)} \\ \rightarrow & \bigvee_{w \in [0:3]} z^{t+15+w} = b[0] \wedge cnt^{t+15+w} = 011 \end{array}$$

Proof. By NuSMV. □

D. Strobing Correct Bits

The next lemma states that whatever happens in the traversal of "synchronization" edges, strobe points always hit correct voted bits. We consider hypotheses similar to the previous lemma. The BSS[0]-mark matches the start point of the lemma. The automaton could be in state BSS[0] with counter at 011 or 100, or in state FSS with counter at 001 or 010. The reasoning is illustrated in the right part of Fig. 8, where voted bits are shown instead of the input.

Formally, we prove that register BYTE contains the correct frame 79 to 82 cycles following the first bit of the synchronization sequence. The proof shows the exact values of the counter that allow proper transmission.

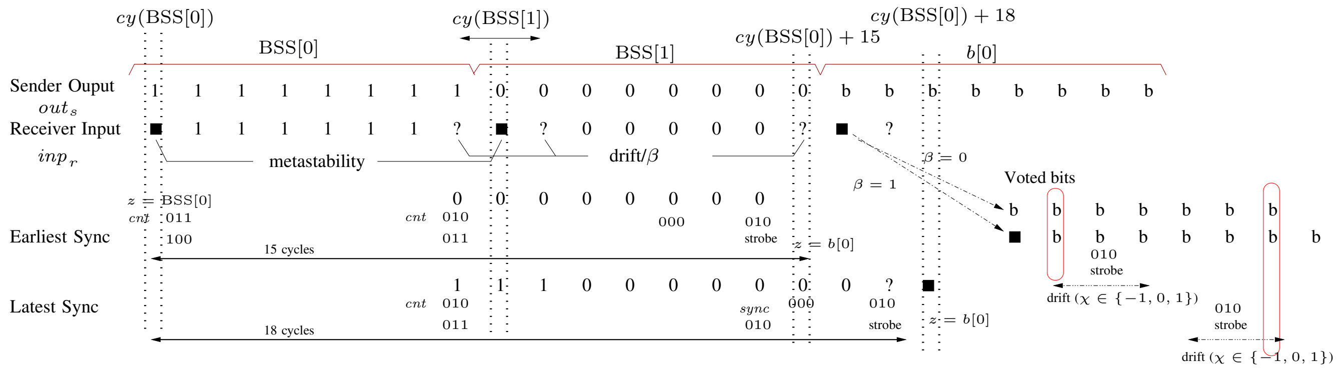

Fig. 8. Earliest and latest possible synchronization points

Lemma 3: **Strobing Correct Bits.**

$$\begin{array}{ll} & \mathcal{H} \wedge cy(t, c+16) \text{ (* BSS[0]-mark is known*)} \\ \wedge & \forall c, inp_r^c = \zeta({}_aR_r^c, e_r(c+1)) \text{ (*mixing analog-digital*)} \\ \wedge & \mathrm{WF}_{ce}(ce_s, L, k, c) \wedge \mathrm{WF}_{In}(out_s, L, c) \text{ (*sender OK*)} \\ \wedge & ((z^t = \mathrm{BSS}[0] \wedge cnt^t = 011 \vee 100) \text{ (* states and *)} \\ \vee & (z^t = \mathrm{FSS} \wedge cnt^t = 001 \vee 010)) \text{ (*misalignment*)} \\ \rightarrow & \end{array}$$

$$\bigvee_{w \in [0:3]} \mathrm{BYTE}^{t+79+w} = \langle out_s^{c+16+8\cdot(j+2)} \rangle, j \in [7:0]$$

Proof. From the previous lemma, state $b[0]$ can be reached at four different dates. Therefore, there are four different dates for strobe points. Moreover, due to clock drift these strobe points can be shifted by one cycle. Finally, because of potential metastability when starting to send $b[0]$, the voted bits have two different positions. For each one of these, we prove that strobe points hit good voted values.

Figure 8 illustrates the proof for counter value 010. In the third line, it considers the shortest traversal of the synchronization sequence, *i.e.* $z^{t+15} = b[0]$. The longest traversal is shown on the next line, *i.e.* $z^{t+18} = b[0]$. In the third line, we assume no metastability when sampling $b[0]$, or good resolution of it ($\beta = 0$). The last line considers the opposite case and the voted bits are shifted by one cycle. The "ideal" strobe appears at $t + [15:18] + 7$. This date can shift by one cycle depending on clock drift. This is represented by the values of χ.

Majority voting delays the input by four or five cycles depending on metastability resolution. From Theorems 2 and 4 (for $k = 7$), we know that sending the same bit eight times implies that the receiver always samples seven times properly. Using Theorem 1, we extend this result to subsequent bits. Assuming a mark ξ, we prove the following formula:

$$\bigvee_{\chi \in \{-1,0,1\}} \forall x \in [4:10], v^{\xi+\alpha+\chi+\beta_{c+\alpha}^{\xi+\alpha+\chi}+x} = out_s[c+\alpha-1]$$

If the mark is the BSS[0]-mark, this formula gives us when the bits of a byte are known to be correct. For all possible traversal durations of the synchronization sequence, we must find an α and an x such that strobe points match these good voted bits. This is expressed by the following equality, where the left hand side corresponds to strobe points and the right hand side to the cycles at which the voted bit is correct.

$$cy(c+16) + [15:18] + 8 \cdot j + 7 = cy(c+16) + \alpha + \beta + \chi + x$$

We set $\alpha = 8 \cdot (j+2)$.

The minimum x is required when the right hand side is maximized and the left hand side of the equality is the earliest cycle. This means that the receiver is one cycle behind the sender. Because clock ticks differ at most by one, this implies that χ cannot take value 1.The right hand side is therefore maximized with $\beta = 1$ and $\chi = 0$. We need to find x such that:

$$cy(c+16) + 15 + 8 \cdot j + 7 = cy(c+16) + 16 + 8 \cdot j + 1 + 0 + x$$

A solution is $x = 5$. We see here that there is still one possibility ($x = 4$). This means that counter value 010 would also be provable. This value is a limit, *i.e.* the earliest working synchronization point.

The maximum x is required when the right hand side is minimized and the left hand side of the equality is the latest cycle. This means that the receiver is one cycle ahead of the sender. Again, because of the bound on clock drift, this implies that $\chi \neq -1$. The right hand side is therefore minimized with $\beta = 0$ and $\chi = 0$. Here, we need to find x such that:

$$cy(c+16) + 18 + 8 \cdot j + 7 = cy(c+16) + 16 + 8 \cdot j + 0 + 0 + x$$

A solution is $x = 9$. Counter value 011 would push the x to the limit 10 and constitute the latest synchronization point. Note that this value is equivalent to the one proposed by the FlexRay standard [5]. Value 100 proposed in [1] would be outside this limit, and is therefore not adequate.□

E. Induction Step

The proof of the induction step is very similar. The induction hypothesis gives the BSS[0]-mark for byte i and the possible dates when the automaton reaches the end of byte i, *i.e.* state $b[7]$ and counter at 010. We extend Lemma 3 to be satisfied if the automaton is in state $b[7]$. If the transmission is not completed (bit *done* is low), the BSS[0]-mark of byte $i+1$ has three possible dates at which the state automaton satisfies the hypotheses of our extended Lemma 3. We apply this lemma for all these possible dates.

VI. Related Work

The first verification effort about physical layer protocols was carried out by Moore [11]. Moore developed a general model of asynchronous communications as a function in the logic of the ACL2 theorem prover [8]. Moore's model assumes distortion around sampling edges and do not allow for clock jitter. Sender and receiver modules are also represented by two functions. Moore's correctness criteria states that the composition of these three functions is an identity. He applied this approach to the verification of a Biphase-Mark protocol.

Moore's work inspired many studies around this protocol. Recently, Vaandrager and de Groot [17] modeled the protocol and analog behaviors using a network of timed-automata. Their model is slightly more general than Moore's and allows for clock jitter. They can derive tighter bounds for the Biphase-Mark protocol. Previously, timed-automata have been used to verify a low level protocol based on Manchester encoding and developed by Philips [3]. Another recent proof of the Biphase-Mark protocol has been proposed by Brown and Pike [4]. They developed a general model of asynchronous communications in the formalism of the tool SAL [10] developed at SRI. Their model includes clock jitter and metastability. Using k-induction, the verificaton of the parameterized specification of Brown and Pike is largely automatic. All these studies tackle *protocol specification* only. They prove functional correctness. We prove a more precise theorem about a gate-level hardware implementation and from which bounds on the transmission duration can be derived.

Regarding hardware verification, Hanna [6], [7] used predicates to approximate analog behaviors at the transistor level. The predicates can be embedded in digital proofs. His work is not specifically targeted to communication circuits and does not consider timing parameters, metastability or clock drift. We consider only gates and not their structure in terms of transistors.

VII. Conclusion and Future Work

Reliable transmission between two independent clocked devices is performed using bit clock synchronization, which is achieved by resetting a counter when detecting a synchronization sequence. This specific value is a crucial parameter. We have developed a general and precise model of asynchronous communications and defined a methodology to use this model for hardware design verification. We have proven the exact possible values for this parameter. This proves and disproves values proposed in the literature.

The model of clock domain crossing is about 2,000 lines and is available on the web[3]. The proof presented here was developed in about one man-year and is about 8,000 lines. Most of it is dedicated to the deduction of valid digital inputs from the analog transmission. This technique is independent of the design under verification. The analysis of similar designs will mainly amount to re-prove all digital lemmas.

The case study is extracted from a more complex design which includes a scheduler implementing a high-level clock synchronization algorithm. We are currently applying our approach to the verification of this component, moving towards a fully verified distributed system.

Acknowledgments

Tsverdyshev provided a precious technical support on the use of NuSMV within Isabelle. Shadrin wrote the translator from Isabelle to Verilog. This work initiates from the lecture "Computer Architecture 2 – Automotive Systems" given by Paul at Saarland University and notes taken by students[4].

References

[1] S. Beyer *et al.*, Towards the Formal Verification of Lower System Layers in Automotive Systems, In *Proc. of the 23rd IEEE International Conference on Computer Design (ICCD 2005)*, 2005.

[2] W. R. Bevier, W. A. Hunt Jr., J Strother Moore and W. D. Young, An Approach to Systems Verification, *Journal of Automated Reasoning* 5(4):411-428, 1989.

[3] D. Bosscher, I. Polak and F. Vaandrager, Verification of an Audio Control Protocol, In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, pp. 170–192, LNCS 863, Spinger 1994

[4] G.M. Brown and L. Pike, Easy Parameterized Verification of Biphase Mark and 8N1 Protocols, In *Proc. of 12th Intl. Conf. on Tools and the Construction of Algorithms (TACAS'06)*, pp. 58–72, LNCS 3920, Springer, 2006.

[5] FlexRay Communication System – Protocol Layer Specification v2.1, Rev A, FlexRay Consortium, December 2005.

[6] K. Hanna, Reasoning About Real Circuits, in *Proc. of the 7th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, pp 235–253, Springer-Verlag, 1994

[7] K. Hanna, Automatic Verification of Mixed-Level Logic Circuits, In *Proc. of the Second International Conference on Formal Methods in Computer-Aided Design (FMCAD '98)*, LNCS 1522, pp 133-166, Springer-Verlag 1998.

[8] M.Kaufmann, P.Manolios, and J.Moore, *Computer Aided Reasoning: an Approach.* Kluwer Academic Press, 2002.

[9] R. Männer, Metastable States in Asynchronous Digital Systems: Avoidable or Unavoidable ? *Microelectronics Reliability*, 28(2):295-307, 1988

[10] L. de Moura *et al.*, SAL 2, In *Computer-Aided Verification (CAV'04)*, pp. 496–500, LNCS 3114, Springer, 2004.

[11] J Strother Moore, A Formal Model of Asynchronous Communications and Its Use in Mechanically Verifying a Biphase Mark Protocol, *Formal Aspects of Computing*,6(1) 60–91, 1993.

[12] J Strother Moore, A Grand Challenge Proposal for Formal Methods: A Verified Stack, In *10^{th} Anniversary Colloquium of UNI/IIST*, B. K. Aichernig and T. S. E. Maibaum editors, LNCS 2757, pp 161-171, Springer, 2003.

[13] T. Nipkow, L.C. Paulson and M. Wenzel, Isabelle/HOL: A Proof Assistant for Higher-Order Logic, Vol. 2283 of LNCS, Springer, 2002.

[14] V. Sagdeo, *The Complete Verilog Book*, Springer, 1998

[15] J. Schmaltz, A Formal Model of Lower System Layer, In *Proc. of Formal Methods in Computer-Aided Design (FMCAD'06)*, San Jose, CA, USA, November 12-16, pp 191–192, IEEE/ACM, 2006.

[16] S. Tverdyshev, Combination of Isabelle/HOL with Automatic Tools. In *Proc. of 5th International Workshop on Frontiers of Combining Systems*, FroCoS 2005, LNCS 3717, pp 302-309, Vienna, Austria, September 19-21, 2005.

[17] F.W. Vaandrager and A.L. de Groot, Analysis of a Biphase Mark Protocol with UPAAL and PVS, *Formal Aspects of Computing* 18(4):433–458, Springer, 2006.

[3]www.cs.ru.nl/∼julien/, then Research

[4]www-wjp.cs.uni-sb.de/lehre/lehre.php