Sigma J Eng & Nat Sci 10 (2), 2019, 245-252

Publications Prepared for the Sigma Journal of Engineering and Natural Sciences
Publications Prepared for the ICOMAA 2019 - International Conference on Mathematical Advances and Applications
Special Issue was published by reviewing extended papers

Research Article

DESIGN AND VERIFICATION OF PARITY CHECKING CIRCUIT USING HOL4 THEOREM PROVING

Elif DENİZ*[1], Kübra AKSOY[2], Sofiène TAHAR[3], Yusuf ZEREN[4]

[1]*Yıldız Technical University, Department of Mathematics, ISTANBUL;* ORCID:0000-0002-5742-9284
[2]*Yıldız Technical University, Department of Mathematics, ISTANBUL;* ORCID:0000-0002-4369-3834
[3]*Concordia University, Department of Electrical and Computer Engineering, Montreal-CANADA;* ORCID:0000-0002-5537-104X
[4]*Yıldız Technical University, Department of Mathematics, ISTANBUL;* ORCID:0000-0001-8346-2208

Received: 21.09.2019 Revised: 03.11.2019 Accepted: 11.11.2019

ABSTRACT

Digital data transmission is the most widely used way of modern communication. The data transmission from source to destination should be without loss of information. This is possible by using the method of parity generator and parity checker. A parity check is the process that ensures accurate data transmission between nodes during communication. In this paper, we present the design and formal verification of a parity checking circuit using Higher-Order Logic theorem proving. We use the HOL4 theorem prover to mathematically describe the parity checking specification as well as mathematical model of the circuit implementation. The formal verification of reliability shows that the circuit implementation satisfies the parity checking specification for all inputs and outputs.
Keywords: Parity checking, formal verification, reliability, higher-order logic, theorem proving, HOL4.

1. INTRODUCTION

Information security has become an essential concept for protecting our sensitive data and asset from insider and outsider attacks. Indeed, many methods have been developed in recent years for authentication and data integrity. Among them parity generator and parity checker circuits play an important role in error detection. A parity check is the process that ensures accurate data transmission between nodes during communication. A parity generator is a combinational logic circuit that generates the parity bit in the transmitter [1].

Parity bits are used for the purpose of detecting errors during transmissions of binary information [2]. The parity bit is added to every data unit (typically seven or eight bits) that are transmitted. Parity bits are extra signals which are added to a data word to enable error checking. There are two types of parity: even and odd. The odd parity bit system consists of counting the occurrences of bits whose value is 1 in the data stream. If the number is even, the parity bit value is set to 1, so the total count of occurrences of high bits in the entire stream including the parity bit is odd. The even parity bit method employs inverse logic. If the count of bits with a value of 1 is even in the data stream, the parity bit value is set to 0 making the total count of high bits in the entire stream including the parity an even number. If the count of bits with a value of 1 is odd, the parity bit is set to 1 so the entire stream has an even number of high bits. If the parity bit and the

parity value of the data do not match, the received data is rejected and the receiving device requests the sender to retransmit the data.

The verification of parity checking circuit is imperative for the accuracy of data transmission. Traditionally, reliability verification of parity checking circuits is done using simulation or prototyping proof methods. Simulation has been the most commonly used technique for verification of circuits. However, simulation cannot be guaranteed to produce complete and accurate results due to the inherent nature of numerical simulation coupled with the usage of computer arithmetic. A prototype is a draft representation built to test ideas for design, behavior and flow in a system [3]. Although prototypes are useful tools for resolving a large number of potential issues, this method is time consuming and expensive. Due to the above mentioned reasons, computer simulation and prototyping are not suitable to verify critical circuits. On the other hand, formal methods [4] overcome the above limitations of simulation and prototyping by providing mathematical proofs for the system specification, analysis and verification.

Formal methods are techniques and tools to verify the structure and behavior of systems based on mathematical models. Methods for the formal verification and specification of systems are an important tool for the development of correct systems. They have been applied to the verification of hardware, software and control systems. In particular, reliability verification based on probabilistic techniques for the prediction of reliability related parameters are very critical for the design of more secure systems. The two most widely used formal methods are model checking and theorem proving [5]. Model checking is an automatic verification approach for systems that can be expressed as a finite-state machine [6]. Theorem proving, is an interactive verification approach that allows us to mathematically reason about system properties by representing the behavior of a system in a certain logic [7]. In particular, higher-order logic provides a very high level of expressiveness that allows the specification and verification of complex mathematical models of systems.

In this paper, we work to use higher-order logic theorem proving [8] for the formal reliability verification of parity checking circuit. The main motivation of this work is to provide an accurate specification and verification of a parity checking circuit using higher-order theorem proving. In particular, we propose a new design of parity checking circuit and provide its formal specification and verification for all possible inputs, outputs and times in the HOL4 theorem prover [9].

There exist in the literature several work that address the design and verification of parity checking systems. Most of them verify the proposed hardware implementations using classical simulation methods (e.g., [10]). There exist also a few work that verify properties of the implementation using formal methods, in particular model checking (e.g., [11]). A formal specification and verification using HOL of a parity checking circuit is described in [8]. However, the circuit we propose has added features and is more efficient in terms of implementation by using fewer components compared to [8].

The rest of the paper is organized as follows: In Section 1, we present a brief overview of the HOL theorem prover. In Section 2 provides the formal specification of the parity checking circuit in HOL4. In Section 3, we describe the hardware implementation of the parity checking circuit and its formal model in HOL4. In Section 4 we discuss details of the formal verification of the parity checking circuit using HOL4. Finally, Section 6 concludes the paper.

2. HOL THEOREM PROVING

HOL is a higher-order logic based interactive theorem prover [8] intended for applications to both hardware and software. In fact, higher-order logic is a good formalism for mechanising other mathematical languages because it is both powerful and general enough to allow sound and practical formulations. The HOL system was originally developed by Gordon for reasoning about hardware systems, but in the following years the range of its applications has widened considerably. The HOL proof assistant is now used for mechanised theorem proving in many

areas, including design and verification of critical and real-time systems, program refinement, program correctness, compiler verification and concurrency.

Higher-order logic is used for quantification over arbitrary predicates and functions. In order to ensure secure theorem proving, the logic in the HOL system is represented in the strongly-typed functional programming language ML (Meta Language). Its procedures allow to interact with the theorem prover. The core of the HOL theorem prover is based on 5 axioms and 8 inference rules which are implemented as ML functions. Variables can be functions and predicates. Functions and predicates can take functions as arguments and return functions as values. HOL is capable of conducting proofs using mathematical reasoning. The soundness of HOL theorem proving is guaranteed as every new theorem must be created from these basic axioms and primitive inference rules or any other previously verified theorems/inference rules. This purely deductive aspect provides the guarantee that every sentence proved in the system is actually true.

The HOL theorem prover has many proof tools and automatic proof procedures [13] to assist the user in directing the proof. The user interacts with a proof editor and provides it with the necessary tactics to prove goals while some of the proof steps are solved automatically by the automatic proof procedures [14]. Though HOL includes decision procedures and semi-procedures for various fragments of its logic, much of the reasoning to prove deep mathematical results must ultimately be done manually. The manual reasoning manifests as a proof script written in HOL's Meta Language (ML), weaving together chains of forward inferences and backwards tactics.

In order to facilitate reusage of verified theorems, HOL allows its users to store a collection of valid HOL types, constants, axioms and theorems as a HOL theory in the computer. Once stored, HOL theories can be loaded in the HOL system and the corresponding definitions and theorems can be utilized right away. HOL theories are organized in a hierarchical fashion. Any theory may inherit types, definitions and theorems from other available HOL theories. The HOL system prevents loops in this hierarchy and no theory is allowed to be an ancestor and descendant of a same theory. The theorem prover includes very rich mathematical libraries such as boolean, integer, arithmetic, etc. Different mathematical concepts have been formalized and stored as HOL theories by the HOL users. For example, a mathematician can prove property for real numbers using the axioms of real number theory.

In HOL, we use predicate calculus to formally specify system behaviors. The infixed binary boolean operators $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ denote conjuction ('and'), disjunction ('or'), implication ('implies') and logical equivalence ('if and only if'), respectively. The quantified term '!*t*' means 'for all *t*' and '?*t*' means 'there exist a *t*'. The two truth-values representing truth and falsity are represented by the boolean constant symbols T and F, respectively.

We allow variables to range over functions and the arguments of predicates to be functions. As we see below, the values needed to model sequential devices will sometimes be functions from time to data values. Predicates on such values will be predicates whose arguments are functions; such as predicates are called higher-order. In this paper, we also use arithmetic and boolean theory in order to verify parity checking circuit. In the next section, we provide the formal specification of the parity checking circuit in HOL.

3. FORMAL SPECIFICATION OF THE PARITY CHECKING CIRCUIT IN HOL4

Formal specifications are mathematically descriptions used to verify the implementation of systems such as software and hardware. They describe the behavior of a system or a set of properties representing system requirements. A specification in HOL consists of a combination of predicates, functions and abstract types. In general, formal verification involves proving that a system implementation satisfies the specification in a certain way. The specification, implementation and properties are expressed as assertions in the given logic. A specification

implies a property that the system should exhibit, for example that a binary adder does indeed produce the mathematical sum of its inputs [15].

The parity checking device we consider has two input lines, (*reset* and *inp*) and one output line, (*out*) as depicted in Figure 1.

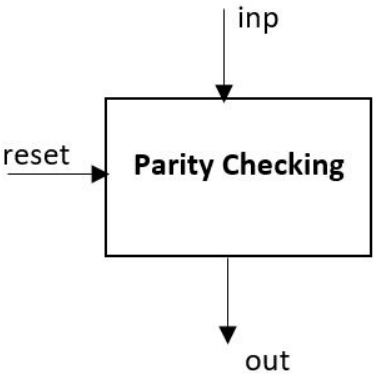

Figure 1. Parity checking

Its behavior can be specified by defining a predicate which holds if and only if *reset*, *inp*, *out* satisfy following specification of parity checking:

*If **reset** is true at time t, then the value of the output at time t is also true. If **reset** is true at time t or t+1, then the value of output at time t+1 is true, otherwise it is equal to the value of the input at time t.*

This specification can be mathematically interpreted as follows; where t represents the clock time.

$$\vdash (\forall t.\ reset\ (t) \Longrightarrow out\ (t) = True) \wedge$$
$$(\forall t.\ if\ [reset\ (t) \vee reset\ (t+1)]\ then\ out(t+1) = True\ else\ out(t+1) = inp\ (t)).$$

This is formalized in HOL by the definition:

```
⊢ SPEC_def (reset, inp, out) <=> (!t. reset t ==> (out (t) =T) ∧ ((!t. out
(t+1) =  if reset (t) ∨ reset (t+1) then T else inp t)
```

4. FORMAL MODEL OF IMPLEMENTATION PARITY CHECKING CIRCUIT

The schematic diagram in Figure 2 shows the design of a circuit that is intended to implement the above specification. It is composed of two multiplexers (if-then-else selectors), two registers (memory storage elements) and a constant one generator. The circuit works by storing the parity of the sequence input so far in the lower of the two registers. Each time t, the input is stored in the lower register (REG2) which output is fed into the lower multiplexer (MUX2). The other input of the MUX2 is the constant one. At the same time, the reset input acts as the selector and input of the upper multiplexer (MUX1). The second input of MUX1 is the previous value of reset stored in the upper register (REG1). The output of MUX1 acts as the selector of MUX2 which output is the output whole of the parity checking circuit.

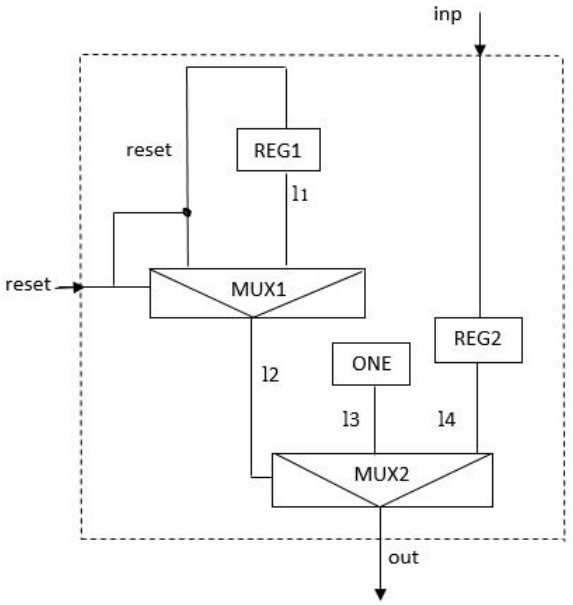

Figure 2. Parity checking circuit implementation

The parity checking circuit is built by connecting together all five components as described above. The internal lines connecting them are *l1, l2, l3* and *l4*. The way we describe this circuit in HOL4 is by a conjuction of predicates which describe the components and the way they are linked together by using existensial quantification over the intermediate signals *l1, l2, l3, l4*. The devices making up this schematic will be modelled with predicates. In following, we describe the predicates of each components of the circuit.

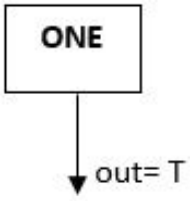

Figure 3. ONE component

The predicate ONE is true when for all times *t*, the output signal (*out*) is T (Figure 3). This is formalized in HOL by the definition:

```
⊢ ONE_def = ONE (out: num→bool) = (!t. (out t = T)).
```

The multiplexer acts as an if-then-else selector where the output *out* a function over time of type num → bool. Figure 4 represents the block diagram of a multiplexer. It has two inputs, a selector of input and an output. When the selector signal *sel* is 0, then the first input (*inp1*) is connected to the output (*out*). The second input (*inp2*) is connected to *out* when *sel* is equal to 1.

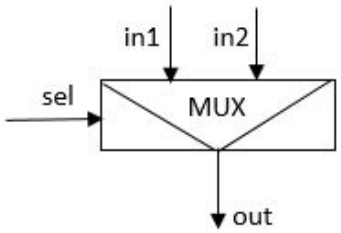

Figure 4. MUX component

The multiplexer can be mathematically interpreted as follows:

$\vdash \forall t.$ *if sel (t) = True then out (t) = in1(t)*
else out (t) = in2(t).

This is formalized in HOL by the definition:

```
⊢ MUX_def = MUX (sw, in1, in2, out: num→bool) =
                !t. out t = if sel t then in1 t else in2 t
```

The registers are unit-delay elements (Figure 5); the values output at time *t+1* are the values input at the preceding time *t*, except at time 0 when the register outputs F.

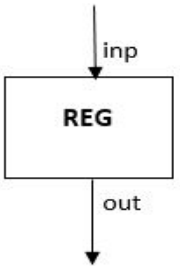

Figure 5. REG component

The register can be mathematically interpreted as follows:

$\vdash \forall t.$ *if* $t = 0$ *then out (t) = false*
else out (t) = in(t-1).

This is formalized in HOL by the definition:

```
⊢ REG_def = REG (inp, out: num→bool) =
                !t. out t = if (t=0) then F else inp (t−1)
```

The schematic diagram Figure 2 can now be represented as a predicate by conjucting the relations holding between the various signals and then existentially quantifying the internal lines.

This implementation is formalized in HOL by the definition:

```
⊢ IMP_def = IMP (reset, inp, out: num→bool) =
            ?l1 l2 l3 l4.  REG (reset, l1) ∧
                           MUX (reset, reset, l1,l2) ∧
                           ONE l3 ∧
                           REG (inp, l4) ∧
                           MUX (l2, l3, l4, out).
```

5. FORMAL VERIFICATION OF THE PARITY CHECKING CIRCUIT

Formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. In HOL, the proof is not always automatic. The user makes the proof according to his/her own customized reasoning. The system checks the user steps to prove a theorem. The user has to tell the system which lemmas should be used and how, within his/her own strategy.

So, a proof strategy in HOL means designing a sequence of deduction in order to prove a theorem with a customized inference mechanism (tactics and tacticals).

Tactics corresponding to conversions and inference rules are defined in HOL. For example, rewriting tactics, such as REWRITE_TAC, add the assumptions of the goal to the given list of theorems and are fundamental in goal directed proofs. Higher-order tactics can implement powerful general proof strategies. When a tactic solves a subgoal, the package computes a part of the proof and presents the user with the next subgoal.

A goal is verified in HOL by simplifying the proof statement, based on already existing theorems and definitions. In order to verify a goal in HOL, we establish a formal specification of intended behavior and a formal description of the implementation and then this formulation of a proof goal can be proved either implementation implies specification or implementation equivalence the specification.

We present a goal for formal verification of the parity checking circuit in HOL as follows:

```
⊢ ! reset inp out. IMP (reset, inp, out) ==> SPEC (reset, inp, out).
```

The first step in proving this goal is to rewrite with definitions followed by a goal using *PURE_REWRITE_TAC*. It is used to simplify the proof goal by using the explicitly given predicts in the list of theorems supplied as an argument. In order to decompose the result we use *strip_tac*. It splits a goal by eliminating one outermost connective. The remaining subgoal can be proved with the simplifier *rw*. A very powerful rewriting tactic for working with goals with assumptions is *RW_TAC*. It not only rewrites the the goal with the given list of theorems but also uses the assumptions to simplify it (whereas *REWRITE_TAC* ignores the assumptions). Finally, we can use for verifying goals related to Boolean theory is called *PROVE_TAC.*

Thus, the goal is proven. Below is the complete HOL proof script.

```
PURE_REWRITE_TAC [ IMP_def ; SPEC_def ; ONE_def ; MUX_def ;
REG_def [ ] ; THEN rpt_strip_tac THEN rw [ ] THEN rw [ ] THEN
PROVE_TAC [ ]
```

6. CONCLUSION

In this paper, we have used HOL4 theorem prover for the reliability verification of parity checking circuit. We provided the formal specification of the circuit behavior in higher-order logic. We developed a new hardware implementation of parity checking circuit and presented its formal model in HOL. We have proven in HOL4 that the developed implementation satisfies the specification of the parity checking circuit for all possible inputs, outputs and time. As a future work, we plan to formally verify the equivalence of our implementation with other designs in HOL4.

Acknowledgement

This research was partially supported by TUBITAK 2221 program.

REFERENCES

[1] Hamming R.W., (2005) The Art of Doing Science and Engineering. *Taylor-Francis e-Library*, California, USA.

[2] Mano M.M., (2004) Digital Logic and Computer Design. *Prentice Hall*, New Jersey, USA.

[3] Jensen L.S., Ozkil A.G., and Mortensen N.H., (2016) Prototypes in engineering design: Definitions and strategies, *14th International Design Conference, In DS 84: Proceedings of the DESIGN*, 16-19 May 2016, pp. 821-830, Dubrovnik, Croatia.

[4] Hall A., (2007) Realising the Benefits of Formal Methods. *J. UCS*, 13(5), pp. 669-678.

[5] Hasan O., and Tahar S., (2015) Formal Verification Methods, In Encyclopedia of Information Science and Technology, *IGI Global*, pp. 7162-7170.

[6] Baier C., and Katoen C.P., (2008) Principles of Model Checking. *MIT Press*, London, England.

[7] Gordon M. J., (1989) Mechanizing programming logics in higher order logic, In Current trends in hardware verification and automated theorem proving, *Springer*, pp. 387-439.

[8] Gordon M. J., and Melham T. F., (1993) Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic. *Cambridge University Press*, London, England.

[9] HOL4, (2019) https://hol-theorem-prover.org/.

[10] Stojčev M., and Stanković T., (2001) Parity Error Detection in Embedded System, *Proceedings of the 2nd International Conference on Informatics and Information Technology*, 20-23 December 2001, pp. 293-307, Skopje, Macedonia.

[11] Umezawa Y., and Shimizu T., (2005) A formal verification methodology for checking data integrity. In Design, Automation and Test in Europe, *IEEE*, pp. 284-289.

[12] Harrison J., (1996) *Formalized Mathematics*, Turku: Turku Centre for Computer Science, Finland.

[13] Hasan O., (2008) Formal Probabilistic Analysis using Theorem Proving, PhD Thesis, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada.

[14] Stavridou V., (1993) Formal Methods in Circuit Design. *Cambridge University Press*, London, England.