KTH Computer Science
and Communication

Formal Models of Hardware Peripherals

Jonathan Yao Håkansson, Niklas Rosencrantz

Supervisor: Roberto Guanciale

Examiner: Örjan Ekeberg

CSC, KTH 2017

Abstract

In this project we build models and verify security properties of hardware peripherals. Our aim has been to build formal models of hardware peripherals with use of the model checker NuSMV to check security properties. We created a formal model of the universal asynchronous transmitter/receiver (UART) and a model of the main memory (RAM).

We do formal verification of these mechanisms. We document our theories and findings. The models consist of statechart diagrams and their corresponding formulas in NuSMV.

We analyze how peripheral devices interact with user processes and the main memory and what traits that interaction poses.

One of our conclusions is that connections with hardware are secure if we don't assume that there is a powerful malicious eavesdropper who can add, remove and eavesdrop on the data being sent.

Terminology and Definitions

Cartesian product - The set of all possible permutations of two other sets.

CPU - Central Processing Unit. It is the center of the computer and performs calculations.

DMA - Direct Memory Access. It is a way to release load off the CPU.

FPGA - Field-Programmable Gate Array. It is a hardware prototyping device.

FSM - Finite State Machine. It is a model of a mechanism which can enter a finite number of states.

HID - Human Interface Device, e.g. keyboard, touchscreen and mouse.

MMU - Memory Management Unit. It is an internal computer hardware that handles memory protection and faster translations between TLB and virtual paging.

Memory-mapped i/o - A method of using hardware by reading and writing to a special location in the computer's main memory.

NuSMV - A model checking language and a tool that can verify or make a counterexample of symbolic formulas

Nios II - Nios II is a brand name for a softcore CPU to which one can download a custom CPU design.

RAM - Random Access Memory. A data storage simply seen as a large array.

RS-232 - Recommended standard 232 (as recommended by IEEE). It is a serial interface for UART.

SDRAM - Synchronous Dynamic Random Access Memory.

SoC - System on a Chip. It is an integrated circuit that integrates all components of a computer or other electronic systems.

State space - The set of values which a process can take.

TLB - Translation Lookahead Buffer. It is a cache memory to facilitate virtual paging.

UART - Universal Asynchronous Receiver / Transmitter. It is a hardware for serial data transmission.

Virtual Paging - A mechanism to divide main memory into virtual page numbers that can be cached.

1. Introduction

In recent years, model checking has emerged as a means both to test and verify the correctness of hardware and software systems. Correctness in this context means that the model satisfies the requirements and that there is a formal proof of correctness. Model checkers can automatically verify whether a model satisfies the requirements. We use NuSMV to conduct formal model checking of hardware peripherals. The hardware peripherals we model are the UARt 16550 with the serial interface RS-232.

1.1. Benefits and Goal-settings

Many external hardware peripherals have no security model when connected to a computer and vice versa. An operating system will in many cases automatically trust a connected hardware peripherals such as a keyboard, mouse or even a network interface that is connected to the computer.

In general there are two kinds of proof techniques for verification and automated proof generation

1. Model checking in which a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution.
2. Automated theorem proving, in which a system attempts to produce a formal proof given a description of the system, a set of logical axioms, and a set of inference rules.

Our work is about model checking using NuSMV. We cover some of the background of model checking and we build a formal model of the UART including its security model, both written in NuSMV. We omit the most fine-grained details of UART and use a simplified model of the hardware[1]. A simplified model is preferable in this case to make it easy to understand for non-technical readers, to limit the time building the model, to fit our scope for an undergraduate thesis and to omit unnecessary details. [explain why you use a simplified model]

There are formal models to test software such as unit tests and integration tests and to guarantee data security. Hardware peripherals and their external connections are usually not part of such test and security models. The scope of our project is to build and verify formal models of a common external hardware peripheral that has been connected to the computer such as a typical UART RS-232 configuration. The configuration could mean many different devices (terminals, modem, serial lines, ...) or even emulated devices such as a system emulating an HID (a serially connected component behaving just like a keyboard or a modem while actually being something else).

[1] Altera_avalon_uart
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf#page=68

2. Background

We answer the following questions:

1. Why are peripherals a problem?
2. Why use model checking?
3. How reliable is the UART connector?
4. What are the inputs of the UART connector?
5. What are the outputs?
6. Is there internal state?
7. How does the internal state change over time and, in particular, in one step?
8. How is our work different than what has already been done?

Our aim is to prove that certain security properties hold, given a configuration. We limit the scope of our project to communication via UART RS-232. We build a simplified model that still is realistic without too many details. The RS-232 specification is relatively easy in comparison to modern advanced peripherals. Therefore we choose to build a simplified model that can still be useful for different purposes such as prototyping a security property or serve as a template for adding more details later.

2.1. Why peripherals are a security problem

In this thesis, we investigate possible approaches for securing a computer system from abuse and similar cases. USB is not similar to UART in practice, but there are similarities in principle and both are used for data in serial transmission.

There are also similar scopes for other kinds of external hardware such as network controllers (NIC) and hardware peripherals that have built-in processors (GPU, FPGA). We have limited our scope to the UART RS-232. Other external hardware security models will be different in details but there will still be similarities for several properties.

A problem is that the protection rings can be subject to threat and sometimes not even be implemented.

A common problem with theorem proving and model checking today is that theorems and models become increasingly complicated due to more functionalities, more performance and more possible combinations. Therefore it has been proposed that induction proofs can be used instead of checking and proving all possible combinations of states [citation needed]. In practice an induction proof means verifying a security aspect for a base-case e.g. a 4-bit CPU and then proving it for example for a parallel connection for 2 parallel 4-bits CPU and then the property is proved for 8-bits and likewise for longer words by induction.

The practical problem definition is firstly proving the problem of unauthorized access to the example of a cryptographic key that should be for authorized owner or user only.

2.2. The Basics of Model Checking

A model checker is a software which automates formal model checking. Model checking is a formal verification that the model satisfied the requirements. There is no provability-checking algorithm. One can search for a proof of a first-order formula in such a systematic way that you'll find it if it exists, and you'll search forever if it doesn't. But if we've been searching for millions of years and it hasn't turned up yet, we still don't know whether the search will go on forever or end next week. These results were proved in the 1930s. The non-existence of an algorithm for deciding whether a formula is provable is called Church's theorem, after Alonzo Church[2].

If Φ is a true statement about all possible behaviors of M then the model checker confirms it (formal verification). If Φ is a false statement about M the model checker constructs a counterexample to Φ that satisfies !Φ.

The model usually includes transitions between states. In model checking, a transition system can be defined to include an additional labeling function for the states. If the transitions are labeled with elements of Σ, then the transition relation is a subset of $Q \times \Sigma \times Q$, while the transition function maps $Q \times \Sigma$ to 2^Q. If the transition system is deterministic and complete, then the transition function can be taken to map Q (or $Q \times \Sigma$) to Q.

For example, to say that there are transitions labeled $\sigma 1$ from $q1$ to $q2$ and $q3$, we would write:

$$\{(q1,\sigma 1,q2),(q1,\sigma 1,q3)\} \subseteq \rho \text{ or } \{q2,q3\} \subseteq \delta(q1,\sigma 1),$$

where ρ is the transition relation and δ is the transition function. It is also possible to adopt more compact notation and denote by something like $q1 \rightarrow_{\sigma 1} q2$ the existence of a transition from $q1$ to $q2$ labeled $\sigma 1$.

A sequence of transition steps can be executed as an actual use-case. This means that we have a method for formally verifying finite-state systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check if the specification holds or not. Extremely large statecharts can often be traversed in minutes. This technique has been applied to several large industrial systems.

Since our goal is to guarantee that the UART is a secure communication channel we will identify which parameters and internal states affect which memory addresses are accessed by the UART and model it.

[2] Alonzo Church's theorem

We consider the UART specification that reports the memory mapped registers (which are input/output and can affect the state of the UART). An additional input and output is the wire to which the UART is connected. Our model represents some features or the hardware such as the minimal set of necessary features of the avalon_altera_uart.

2.3. State of the art in model checking

Previous related work on UART prove the correctness of a UART driver in a context of C and assembly using HOL4 (Duan, Regeher, 2010). Is uses theorem proving to prove software properties. When we model hardware circuits the circuits can be described in the language SMV and the tool NuSMV. NuSMV is able to process files written in an extension of the SMV language. In this language, it is possible to describe finite state machines by means of declaration and instantiation mechanisms for modules corresponding to synchronous composition, and to express a set of requirements in CTL and LTL. NuSMV can work batch or interactively, with a textual interaction shell. (Sebastiani et. al)

We use NuSMV to build a behavioral model of a real UART RS-232 peripheral: The altera_avalon_uart, and we use model checking to check the requirements whether a connected peripheral can be trusted.

SMV programs are composed by a number of modules, Each module contains state variable declarations, variable initializations defining the initial states and assignments defining the transition relation.

A transition relation specifies a constraint on the values that a variable can assume in the next state , given the value of variables in the current state:

```
proc != read -> next (Rx) = Rx;
```

The above means that Rx (the receiver of the UART) stays the same unless the process is reading.

In logic, linear temporal logic (LTL) is a modal temporal logic with modalities referring to time. In LTL, one can encode formulae about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true or a condition has been true for some time, etc. LTL is a subset of CTL, computational tree logic, which additionally allows branching time and quantifiers. LTL is sometimes called propositional temporal logic, abbreviated PTL.

A safety property is distinct from a liveness property where one proves that something desirable eventually will happen.

2.4. Properties of the 16550 UART

UART enables serial character bitstreams between a computer system or an FPGA and an external peripheral. The UART implements the RS-232 protocol timing, and provides adjustable baud rate, parity, stop, and data bits. The feature set is configurable, allowing designers to implement just the necessary functionality for a given system. The UART provides a memory-mapped interface that allows peripherals (such as a processor) to communicate with the UART by reading and writing control and data registers.

The UART transmitter consists of a7-, 8-, or 9-bit txdata holding register and a corresponding 7-, 8-, or 9-bit transmit shift register. The shift register write the txdata holding register via the Avalon-MM slave port. The transmit shift register is loaded from the txdata register automatically when a serial transmit shift operation is not currently in progress. The transmit shift register directly feeds the TXD output. Data is shifted out to TXD LSB First. The two registers provide double buffering. A master peripheral can write a new value into the txdata register while the previously written character is being shifted Out. The master peripheral can monitor the transmitter's status by reading the status register's transmitter ready (TRDY), transmitter Shift Register empty (tmt), and transmitter overrun error (TOE) bits.The transmitter logic automatically inserts the correct number of start, stop, and parity bits in the serial TXD data stream as required by the RS-232 specification. It is important that the busy signal has no hazards.

We specifically build the formal model of the states of the UART and the states of the system that should be used for formal verification and model checking. Model checking is an examination of all the possible states of a design to determine if any of them violate a specified set of properties, similar to a mathematical proof that covers all possible cases. Model checking is done through mathematical analysis where properties such as assertions are checked against a specification.

It’s theoretically possible to use model checking to fully verify the functionality of design, it is typically used for sub-blocks of a design.

Equivalence checking is an alternative method of formal verification where one compares two different implementations of a design to see if they are functionally equivalent.

The UART serial receiver consists of two registers: Bitwise shift register and bitwise holding register. In the case of altera_avalon_uart, the memory management master peripheral reads the holding register. The holding register is loaded from the shift register when a new byte is received. These two registers provide double Buffering. The rxdata register can hold a previously received character while the subsequent character is being shifted into the receive shift Register. A master peripheral can monitor the receiver's status by reading the status register's read-ready (RRDY), receiver-overrun error (ROE), break detect (BRK), parity error (PE), and framing error (FE) bits. The receiver logic automatically detects the correct number of start, stop, and parity bits in the serial RXD stream as required by the RS-232 Specification. The receiver logic checks

for four exceptional conditions, frame error, parity error, receive overrun error, and break, in the received data and sets corresponding status register bits.

We build and check such formal models that can achieve security with a hardware peripheral. We briefly compare model checking with theorem proving. We answer related questions and prioritize the known related problems and relations with software such as operating system and application programs. We describe the delegation of responsibilities in such configurations, what to protect and what the potential hardware and software vulnerabilities are.

The specific UART in this project is a hardware peripheral (part of an SoC) that is memory mapped and available for use in the context of a program running on an FPGA. It requires configuration before use, which is generally achieved by writing values into a memory mapped configuration register. The UART can be used to send and receive arbitrary data asynchronously over two signal wires, TX and RX, respectively.

The serial interface uses an asynchronous protocol, i.e. no clock signal is transmitted along the data. The receiver has to have a way to "time" itself to the incoming data bits.

In the case of RS-232, that's done this way:

1. Both side of the cable agree in advance on the communication parameters (e.g. speed and format). That's done manually before communication starts.
2. The transmitter sends "idle" (="1") when and as long as the line is idle.
3. The transmitter sends "start" (="0") before each byte transmitted, so that the receiver can figure out that a byte is coming.
4. The 8 bits of the byte data are sent.
5. The transmitter sends "stop" (="1") after each byte.

The specific UART we have chosen is the one that is used in Altera FPGA, named altera_avalon_uart. The product specification is according to the following:The RS-232 settings are provided below for our connection from the Linux host that is used.

The RS-232 standard is used by many specialized and custom-built devices. This list includes some of the more common devices that are connected to the serial port on a PC. Some of these such as modems and serial mice are falling into disuse while others are readily available.

The UART core implements RS-232 asynchronous transmit and receive logic. The UART core sends and receives serial data via the TXD and RXD ports. The I/O buffers on most Altera FPGA families do not comply with RS-232 voltage levels, and may be damaged if driven directly by signals from an RS-232 connector. To comply with RS-232 voltage signaling specifications, an external level-shifting buffer is required (for example, Maxim MAX3237) between the FPGA I/O pins and the external RS-232 connector. The UART core uses a logic 0 for mark, and a logic 1 for space. An inverter inside the FPGA can be used to reverse the polarity of any of the RS-232 signals, if necessary.

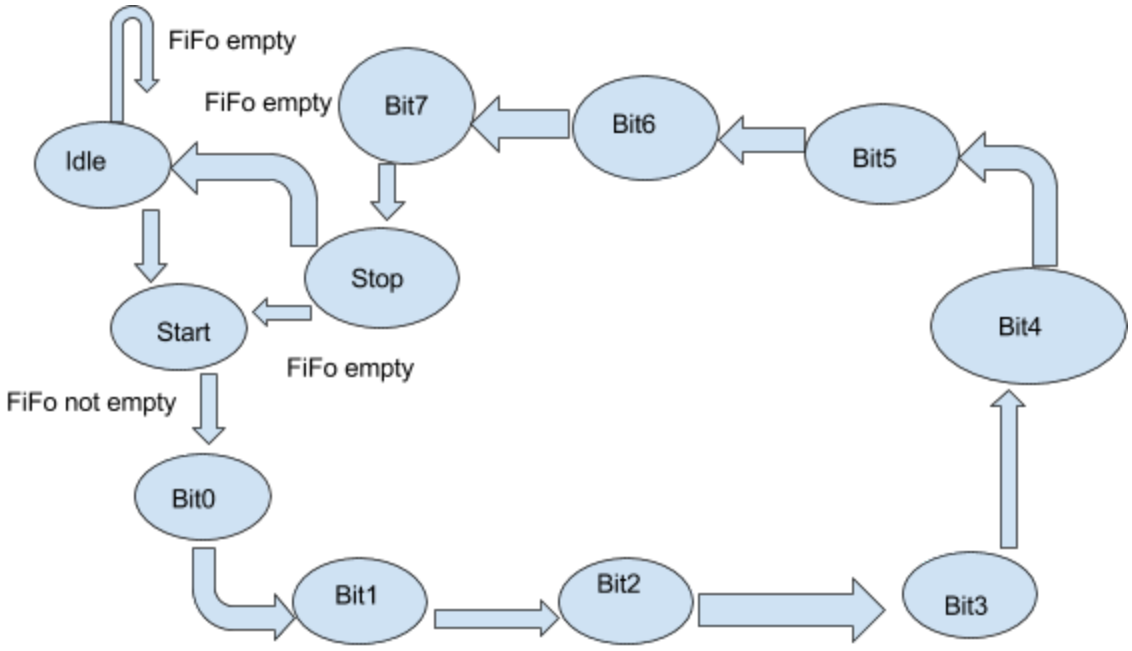


Figure 3. FSM diagram of the UART shift register.

The state machine waits for the CPU or DMA to place an entry in the transmit FIFO. Once there is data present it transmits one Start bit, 8 data bits and finishes with one Stop bit. The bits are sent starting with bit 0. A design requirement is that two bytes can be sent back to back, so the STOP state must go directly to the START state if more data is available.

If the configuration is done properly and correctly, the UART connection can't allow manipulation of any protected user processes such as web browser or a command-line interpreter. A simple overview of our model of hardware, its policy enforcement (the driver) and a possible intruder can be seen in the following figure.

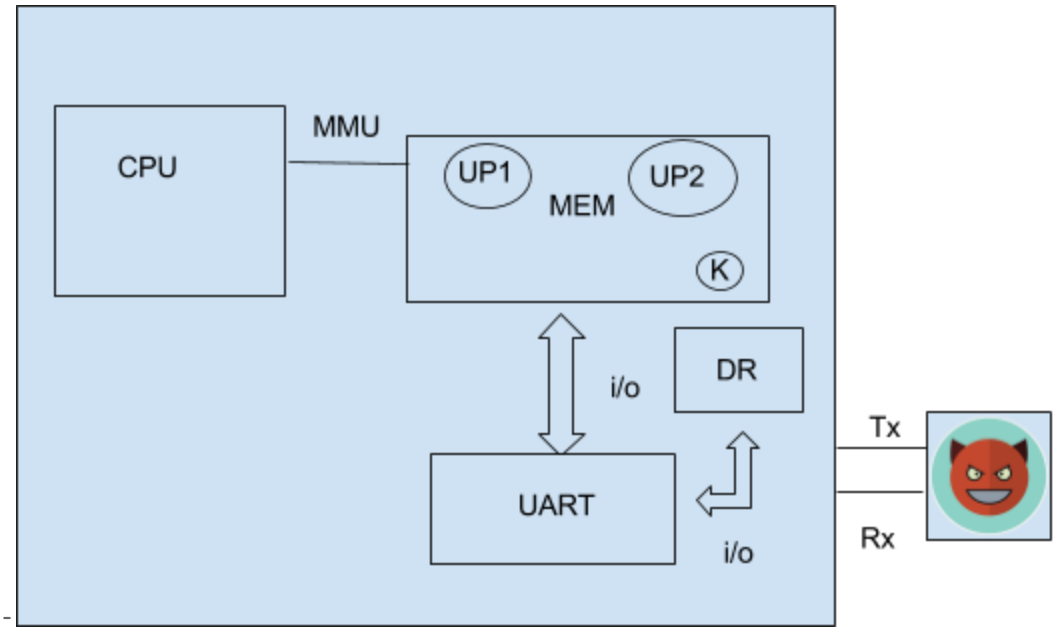


Figure 14. Hardware model overview.

The above is our simplified model of hardware host and the connected external hardware. In device drivers timing constraints are important and also embedded systems have important timing constraints, often deterministic, as part of their specifications. For example, the landing gears of a quadcopter should deterministically respond within a certain number of ms of receiving input (Duan, Regeher, 2010).

2.5. The DB-9 Connector

The DB-9 serial connector has 9 pins and the 3 relevant pins are:

- pin 2: RxD (receive data)
- pin 3: TxD (transmit data)
- pin 5: GND (ground)

Using these 3 pins, 2 connected peripherals can send and receive data. It also is used for null modem connections.

Data is commonly sent in bytes serialized by the LSB (data bit 0) that is sent first, then bit 1 and the following bits with the MSB (bit 7) last.

3. Specification and Methodology

The premise in this case is that there is a memory unit “K” (for instance a cryptographic key) that should not get overwritten or tampered with. We want to prove that it is always the case that K is not overwritten. We created one model for the main memory and one model for the UART. We prove that the following formulas are true:

LTL SPEC G (proc != write -> (memory.K = next (memory.K)))

LTL SPEC G (output != memory.data[1]) -> G (memory.K = next (memory.K))

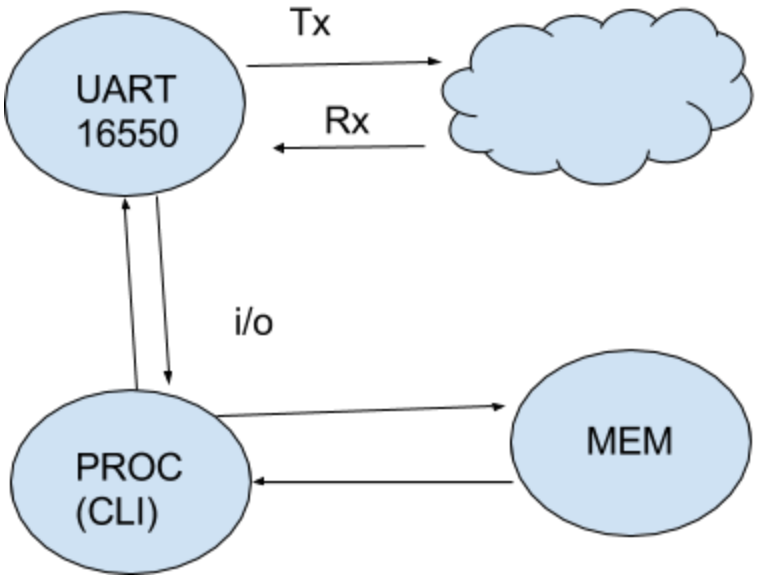
Our idea was to look for a counter-example. We also asked: is there a bad behaviour? What we are checking is a safety property i.e. “nothing bad ever happens”.

3.1. Scenario

Our scenario is that we have a cryptographic key, “K” that should not get overwritten. The UART can transmit and receive and that there is a user process writing input and reading output to and from the UART. The process can also read and write to the main memory of the computer. There are three models being built to represent the scenario: UART, PROC and MEM.

We want to prove that K stays the same given that the process doesn’t write to K. We consider the following specification:

$$LTL\ \ SPEC\ G\ PROC.OUT \neq WRITE\ K \rightarrow G\ MEM.K = NEXT(MEM.K)$$



The FSM is the result of the synchronous composition of the subsystems the UART, the memory and the process. The new state space is the cartesian product of the ranges of the variables.

3.2. NuSMV programming and interaction

We wrote the models of UART and of main memory in SMV. We made the process and the CPU be variables. An SMV program is composed by a number of modules where each module contains state variable declarations, assignments defining the valid initial states and assignments defining the transition relation. In the assignments the lhs can non-deterministically be assigned to any value in the set of values represented by the rhs, for example a finite set of possible states.

The models were built in an assignment style with at least one initial state and all states having at least one next state.

When building our model we could step through the steps interactively to manually inspect what the NuSMV was checking. We did formal verification of the LTL formulas.

We completed the following subtasks to complete our main tasks.

1. We identified the problem to solve. One problem is that device specifications of 600+ pages have no obvious formal proof and that hardware was developed less by security aspects and more by functional properties and performance.

   A second problem is that the model we build might either become overly simplistic or too complicated to be feasible.

2. We built and formalized our model. Our goal has been to build and formally verify a model of secure hardware peripherals. Our networks, operating systems and applications must interact with secure hardware peripherals such as input/output devices (UART, USB, network controllers), interrupt controllers and coprocessors (GPU, FPGA). The results are from both theoretical models as well as verifying with real hardware peripherals e.g. Altera DE2-115 FPGA and its UART. The theorems and models we use and build are simplified to enable us conclude intermediate results at the intermediate level.

3. We conducted model checking. Using the appropriate selected tools we have written down the details of our theory and our model. We used automatic theorem proving with NuSMV to prove our theory.

4. We documented, discussed and reported relevant conclusions and communicate the results. We documented and elaborated about our findings. We also presented the results for an audience of intermediate level..

4. Results and Discussion

A well-known fact is that a computer mechanism that can be technically attacked (for example physically pulling out the RAM module and by such means extracting the data from the physical RAM memory) will not be 100 % safe or completely protected in principle. The safest usage of the UART today is to not leave to connector open.

If someone has physical access it is difficult to prevent abuse of the hardware. An attacker can for example pull out the dram modules (perhaps after spraying them with freezer spray) and just read out the cryptographic keys, use JTAG, and many promote possible attacks.UART doesn't have a security model in its current availability.

RS232 doesn't specify higher-level about the wire format. It can be a teletype, AT commands for modems, IP over SLIP/PPP, MODBUS, or something else e.g. X.25

4.1. Comparison of Model Checking Tools

Hardware circuits can be described in NuSMV. We built a behavioral model of a 16550 UART peripheral and we use model checking to check the requirements whether a connected peripheral can be trusted. We chose the appropriate modelling tool NuSMV and also tried Java Path Finder and the Spin model checker to perform model checking of a simplified connected hardware consisting of an UART RS-232 peripheral connected to a computer system and its operating system. We handled the question of responsibility for security: Is the operating system or the hardware responsible?

We evaluated the tools for model checking, of which we finally decided to select NuSMV for reasons of wanting to learn a new tool and learning a new language, as well as the tool is well established and suitable for our needs.

Our UART is, hence, used from the application context within an SoC to send and receive data to and from a hardware peripheral. The RS-232 interface has the following characteristics:

- A 9 pin connector, i.e. DB-9 connector
- Bidirectional full-duplex communication (the PC can send and receive data at the same time).
- Can communicate at a maximum speed of roughly 10 KBytes/s.

The case that we formalize is a connected UART RS-232 device in a security context. The manufacturer is Altera and the name of the UART is 16550.

One can use Java Path Finder, NuSMV or the Spin model checker for model checking.

We evaluated three frameworks to conduct our model checking of UART and RS-232.

1. Java Path Finder, JPF, is used to verify executable Java ByteCode programs. JPF was created by NASA AMES Research Center. Main focus of JPF uses and executes Java ByteCode and can store states, match restore program states. Main usage for JPF is Model Checking of concurrent programs. You can use JPF as model check of distributed application, model checking of user interfaces, low level program inspection, program instrumentation.

2. SMV is language that is well-suited for modelling hardware circuits. It is a language for verification of finite state systems (FSM). The tool NuSMV conducts checking of finite state machines and checks if specifications are correct against CTL called temporal logic.NuSMV supports both CTL and LTL. NuSMV is a BDD-based (Binary Decision Diagram) model checker that allows to check finite state systems against specifications in the temporal logic CTL. The software is freely available at http://nusmv.irst.itc.it/ where also a tutorial and manual is available. This program makes it able to have finite systems from completely synchronous to completely asynchronous. Data types are thought to be used as finite state machines and it have only datatypes as Boolean, scalar types, bit vectors and fixed arrays. NuSMV has following features Interaction, Analysis of invariants, Partitioning methods, LTL Model Checking, PSL Model Checking, SAT-Based Bounded Model Checking.

3. Spin-Model Checking is well-suited for modeling of concurrent systems. It is a verification system that can be used as verification tool for asynchronous process systems. The Main focus of Spin is process interactions and provide abstract from internal sequence computations. Some formal methods that Spin have are:
   NuSMV was easier to use than Spin/Promela

- An intuitive program-like notation for design choices.
- A concise notation for general correctness requirements.
- A methodology for establishing logical consistency.
  Spin accepts a verification language PROMELA (Holzmann, 1997) specified in syntax of Linear Temporal Logic.

Spin is used as an open-source model-checker that has been used for the formal verification of distributed software systems. It was invented by Bell Labs Computing Sciences Research Center in 1980. It was released in the public domain in 1991 and got awarded 2002 by ACM for system software.

It is used to trace logical design error in distributed systems like operating systems,data communications protocols and looking for the logical consistency in the system. To describe the specification it is using a high level language that is named Promela. The abbreviation is for PROcess or PROtocol Meta Language.

The language grants dynamic creation of concurrent processes that can communicate over synchronous or synchronous message channels along shared memory.

This tool is a usage of correctness properties. These Properties to be specified in different ways, like system or process invariants usage of assertions on the point of Linear Temporal Logic (LTL).

Spin can help with automatically report on deadlocks ,unspecified receptions, race conditions.
The simulation regarding the specification can be two properties interactive or random.
Spin can create a C-program that can make a fast complete verification of the system state space.

The techniques that are proof are both exhaustive and partial. It is founded on depth-first or breadth-first search.

Promela(PROcess or PROtocol Meta Language) is the input language of spin. Its specification has three parts.

-Processes. It has data and channel declarations that specify a performance. Are declared by proctype.
-Data objects are expressed globally or locally to processes. The syntax looks like C.
-Message channels are expressed globally or locally to inside the process. Usage to exchange data between processes.

Because it is hard to define what is good properties, promela has different types of exactness about (un)reachable states and path properties. The following:

-Basic (boolean expressions that must become TRUE).
-End-state(To separate valid end-states from deadlocks).
-Progress-state labels(To tag states that did something good eligible).
-Accept-state labels(It is used to correct fairness condition, an accept condition cannot be visited more than often).
-Never claims (Behaviour to avoid)
-Trace assertions(Specifies valid or invalid sequences of operations that processes can perform on message channels)

NuSMV had a library that provided a better use for the work.

In our methodology there are various types of formal verification that we can do. One is theorem proving, which is a proof of a relationship between a specification and an implementation as a theorem in a logic, proved within the framework of a proof calculus. It is used for verifying arithmetic circuits.

A second way of formal verification is model checking which is checking the specification in the form of a logic formula, the truth of which is determined with respect to a semantic model provided by an implementation. Model checking is starting to be used to check small modules in industry.

Equivalence checking is a third way that checks the equivalence of a specification and an implementation. Equivalence checking is the most common industry use of formal verification.

We have chosen to not perform equivalence checking in this project and instead working primarily with building a sufficient model for our needs and then perform model checking using the appropriate tools.

5. Concluding Remarks

6. Acknowledgements

The authors would like to thank their supervisor Roberto Guanciale for outstanding guidance during the work. We would also like to thank Patrick Trentin for letting us learn NuSMV and model checking in detail.

7. Bibliography

1. Sebastiani et. al. NuSMV : An OpenSource Tool For Symbolic Model Checking http://nusmv.fbk.eu/NuSMV/papers/cav02/pdf/cav02.pdf
2. Jianjun Duan, John Regehr 2010, Correctness Proofs for Device Drivers in Embedded Systems, https://www.cs.utah.edu/~regehr/papers/ssv10.pdf
3. Induction proofs for verification http://theory.stanford.edu/~arbrad/papers/iictl.pdf
4. Klauser, 2017, Altera 16550 UART driver https://github.com/tklauser/qemu/tree/altera-uart
5. Bruce Schneier, 2011, Yet Another "People Plug in Strange USB Sticks" Story https://www.schneier.com/blog/archives/2011/06/yet_another_peo.html
6. UART RS-232 https://en.wikipedia.org/wiki/RS-232
7. RS-232 specification http://www-ug.eecg.toronto.edu/msl/nios_devices/dev_rs232uart.html
8. BadUSB - On accessories that turn evil, 2014, Nohl, Krissler, Lell https://srlabs.de/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf
9. Holzmann, 1997, The Model Checker Spin http://spinroot.com/spin/Doc/ieee97.pdf
10. Buffer overflow http://www.csc.kth.se/utbildning/kth/kurser/DD2395/dasak06/dokument/F4/F4.pdf
11. Computer Security Dieter Gollmann
12. Altera, 2016, Embedded Peripherals IP User Guide https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf#page=68
13. http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C11_SerialInterface.htm
14. http://chibios.sourceforge.net/docs/hal_stm32f4xx_rm/group___u_a_r_t.html
15. Model checking overview http://www.cs.cmu.edu/~modelcheck/tour.htm
16. DESIGN AND IMPLEMENTATION OF CUSTOMIZABLE http://www.ijraet.com/pdf17/31.pdf
17. http://www.ijcst.org/Volume7/Issue4/p2_7_4.pdf
18. The serial driver layer http://www.linuxjournal.com/article/6331

19. tty layer http://marcocorvi.altervista.org/games/lkpe/tty/tty.htm
20. Klauser, 2010. altera_uart.c
http://lxr.free-electrons.com/source/drivers/tty/serial/altera_uart.c
21. http://lxr.free-electrons.com/source/drivers/tty/serial/serial_core.c
22. NuSeen (2017) http://nuseen.sourceforge.net/

8. Appendix 1

8.1. UART specification

Name	altera_avalon_uart
Version	16.1
Author	Altera Corporation
Description	No description
Group	Interface Protocols/Serial
User Guide	https://documentation.altera.com/#/link/sfo1400787952932/iga1401317331859
Release Notes	https://documentation.altera.com/#/link/hco1421698042087/hco1421697689300
Data bits	Determines the widths of the txdata, rxdata, and endofpacket registers.
Stop bits	Use to terminates a receive transaction at the first stop bit.
Include end-of-packet	Include end-of-packet register