Formal models of hardware peripherals

Jonathan Yao Håkansson

Niklas Rosencrantz

KTH Institute of Technology, B.Sc. project in Computer Science

{jyh,nik}@kth.se

1. Abstract

Security and privacy are important concerns. Once somebody is insecure they will almost always stay insecure.

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

Model Checking is formally defined as $M \models \varphi$ which means a method for formally verifying finite-state concurrent 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 state-spaces can often be traversed in minutes. The technique has been applied to several complex industrial systems such as the Futurebus+ and the PCI local bus protocols.

If your goal is to guarantee that the UART does not leak a crypto key, can you identify which parameters and internal states affect which memory addresses are accessed by the UART? What are the inputs of the component? What are the outputs? Is there internal state? How does the internal state change over time and, in particular, in one step? We try to look at the specification, it should report 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.

UART (Universal Asynchronous Receiver Transmitter) is a hardware peripheral (part of an SoC) that is memory mapped and available for use from the context of a program running on a microcontroller. 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 (no clock needed) over two signal wires, TX and RX, respectively. Simply put, UART is used from the application context within an SoC to send and receive arbitrary data to/from an external device. JTAG is used to verify a circuit and test device logic. An RS-232 interface has the following characteristics:

- Uses a 9 pins connector "DB-9" (older PCs use 25 pins "DB-25").
- Allows bidirectional full-duplex communication (the PC can send and receive data at the same time).
- Can communicate at a maximum speed of roughly 10KBytes/s.

Keywords: NuSMV, UART, RS232, model checking, formal verification

The case that we formalize is a connected UART RS-232 device in a security context. **DB-9 connector**

You probably already saw this connector on the back of your PC.

It has 9 pins, but the 3 important ones are:

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

Using just 3 wires, you can send and receive data.

Data is commonly sent by chunks of 8 bits (we call that a byte) and is "serialized": the LSB (data bit 0) is sent first, then bit 1, ... and the MSB (bit 7) last.

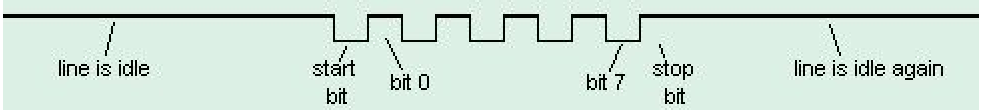
Asynchronous communication

This interface uses an asynchronous protocol. That means that 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 (speed, 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.

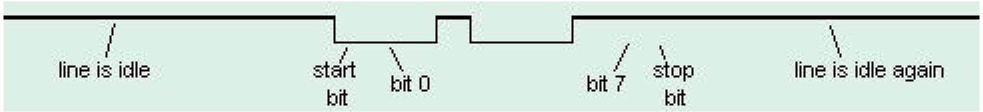
Let's see how looks the byte 0x55 when transmitted:



Byte 0x55 is 01010101 in binary.

But since it is transmitted LSB (bit-0) first, the line toggles like that: 1-0-1-0-1-0-1-0.

Here's another example:



Here the data is 0xC4, can you see it?

The bits are harder to see. That illustrates how important it is for the receiver to know at which speed the data is sent.

Physical layer

The signals on the wires use a positive/negative voltage scheme.

- "1" is sent using -10V (or between -5V and -15V).
- "0" is sent using +10V (or between 5V and 15V).

So an idle line carries something like -10V.

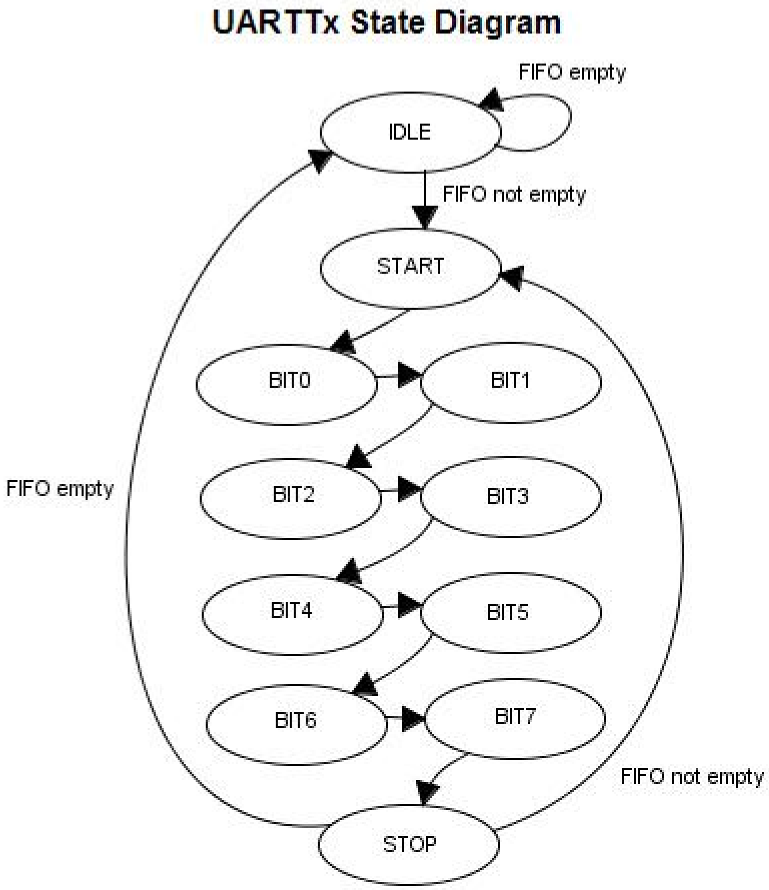
4. Method

4.1. Participants

Jonathan Yao Håkansson and Niklas Rosencrantz.

4.2. Assessments and Measures

UART enables serial character bitstreams between a computer system or an FPGA and an external peripherals. 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.



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.

8.1. Practical examples and problems

A main practical problem is that the operating system often will trust any random physical device that is physically connected to the computer. The operating system will automatically run a program from a trusted physical device, and that program can install malware. Hence, we are not safe when connecting physical peripherals to a computer.

An example of actual abuse done by hardware peripherals is the “BadUSB” which was a USB device that unauthorized emulated an authenticated user’s keyboard and could issue unauthorized commands by impersonating an authorized user. We examine the ways of securing a computer system from such abuse and similar cases.

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.

Bruce Schneier, a famous expert in IT security, has specified a common and general IT security problem as follows:

“Any security expert can invent a cipher that he himself can’t break.”

Schneier’s observation is that everybody can outsmart themselves and therefore you can also be intruded because intruders can get around your security solution in ways you didn’t prepare for. We study the hardware security applications of this problem and the existing and proposed solutions. Recently there has also been reports in the news and media that important and vital Government departments and functions of the state have been intruded in this manner by injecting malware from external hardware peripheral, a hardware peripheral which mistakenly has been trusted.

The common data security patterns often use the placeholder actor names Alice, Bob, Carol, Eve, Mallory, Peggy, Victor, Trent etc for different roles and functions. Typically Alice and Bob are the ones sending data and the other guys are actors with different functions. Will they ever be 100 % safe?

A common problem with theorem proving and model checking today is that theorems and models become increasingly complicated due to more functionalities, more performances and more possible combinations. Therefore it has been proposed that induction proofs can be used instead of checking and proving all possible combinations. In practice an induction proof means proving 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.

We first show that the cryptographic key is accessible and unsafe by policy and mechanism. Then we show that a formal model enforced by a mechanism can make the cryptographic key formally private and secure.

9. Goals

The main goal is to build a behavioral model of a real UART RS-232 peripheral and use model checking to check the assumptions so that a connected system can be trusted.

We arrange the following sub-goals to complete our main goal.

1. **We introduce, describe and explain the background and explain the problem that we're going to solve:** One problem is that device specifications of 600+ pages have no obvious formal proof and that hardware was not been 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 (like now) or too complicated to be feasible.

2. **We build and formalize our model:** Our goal is to build and proof-check 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 proofs anas well as checking the models that we build and verifying with real hardware peripherals. The theorems and models we use and build are simplified to enable us conclude intermediate results at the intermediate level. We should combine functional invariants, for example the invariant that our “protected” is always “protected” from unauthorized use, varying the inputs.

3. **We do model checking:** Using the appropriate selected tools we write the details of our theory and our model. We can use automatic theorem proving to prove our theory. We can also combine and compare the same scope by checking a formal model of the hardware peripherals and the connections, as exemplified by a simplified model of a connected UART RS-232.

   A known problem with model checking is called the state explosion problem, which is the problem that the number of states in a system grows exponentially by the number of different parameters. It has been told that induction proofs and inductive and recursive techniques could be able to get around the state explosion problem by proving or checking only a small number of cases and then the proof or model is generalized for all possible cases similar to an inductive or recursive proof technique.

4. **We document, discuss, report, draw relevant conclusion(s) and communicate the result(s):** We document and elaborate about our findings. We will also present to result for an intended audience of intermediate level, less advanced than current research and more advanced than trivial.

10. Specification and methodology

We specify the problem and limit ourselves to a small and simplified model of an UART RS-232 consisting of the parts

We used three programs to prove our UART 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 on concurrent programs. You can use JPF as model check of distributed application, model checking of user interfaces,low level program inspection,program instrumentation.

2. NuSMV is Computer program that is a tool for verification of finite state systems. This program are checking finite state machines and checks if specifications are correct against CTL called temporal logic. 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 you will also be able to find a tutorial and manual. 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,bit vectors,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 are 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

- 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 specified in syntax of Linear Temporal Logic.

We have evaluated these languages for the scope of our project. The most appropriate program to prove this problem is NuSMV because of finite state machines and after that the most likely think is JPF because of our wide java knowledge.

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.

10.1. Scope and limitations

We limit the scope of our project to the UART RS-232 and 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.

We specifically build formal models 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.

11. Milestones

11.1. Introduction learning and preparations

Read, learn and practice formal models, select and learn the tools to use and divide the work between the collaborators.

11.2. We formalize our theory and our model

Formalize the theories and build the model(s) using the tools at hand. A simplistic outline of our theorem, that we are going to add more details to, can look as follows.

$$\begin{array}{rl} & \text{Conf} \\ & \quad\wedge \\ \models & \text{Model} \\ & \text{UART} \\ & \quad\wedge \\ \not\models & \text{WRITE} \end{array}$$

The above theorem means in plain English that if the configuration is done properly and correctly, the UART can’t manipulate any protected user processes such as web browser or a command-line interpreter. We are adding more detail and specifics to the theorem so that it can be useful without being overly simplistic and without being too complicated.

A simple overview of our model of hardware can be seen in the following figure.

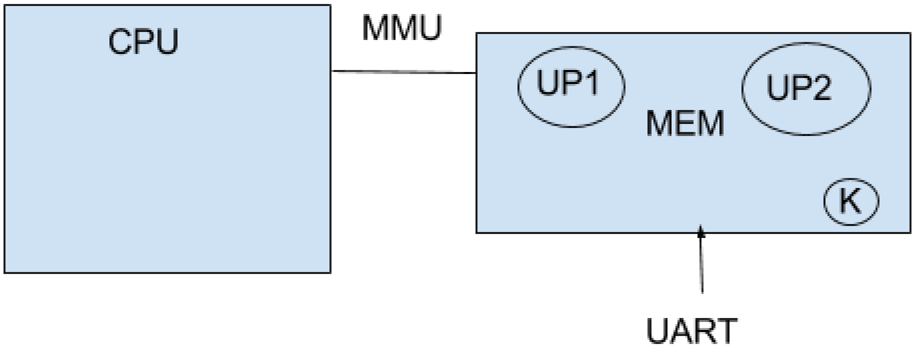


Figure 2. Hardware model overview

The above was our simplified model of hardware host and the connected external hardware device to illustrate the problem being solved. We added more details to our view in our continued project so that our modelling became more detailed and realistic.

11.3. Choosing the appropriate tools

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?

Our main four problems and questions were (1) that our models were either overly simplistic or much too complicated. We worked on building models that were realistic simplifications without being overly simplistic.

Another problem (2) we to choose the right tools for model checking, 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.

A third problem (3) we had was to correctly identify whether it is hardware of software that should perform the protection mechanism. The responsibility of protecting a system might be the responsibility of the operating system and not the hardware.

Our fourth question (4) was to what extent our project should or could create a new model and a new solution or if we mainly must use existing models and existing solutions.

11.4. Methods for model checking

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

11.5. Results and conclusions

We report and account for our results and conclusions. We document our tasks, our methodology and our findings in a relatively detailed report. We also will speak and answer questions with supervisors and members of other projects about our project and about our documentation.

We mention an overview of where the current technology is heading, some new ideas such as induction proof and recursive techniques instead of proving and instead of checking all possible cases.

We expect to conclude that a computer user or the data will not be 100 % safe or completely protected in principle. We hope to conclude a result what the best and safest usage of hardware peripherals is today and where the current development in our discipline is going.

12. SMV mutex program

```
MODULE main

VAR

state1: {n1, t1, c1};

ASSIGN

init(state1) := n1;
```

```
next(state1) :=
case
   (state1 = n1) & (state2 = t2): t1;
   (state1 = n1) & (state2 = n2): t1;
   (state1 = n1) & (state2 = c2): t1;
   (state1 = t1) & (state2 = n2): c1;
   (state1 = t1) & (state2 = t2) & (turn = 1):  c1;
   (state1 = c1): n1;
   TRUE : state1;
Esac;
VAR
state2: {n2, t2, c2};
ASSIGN
init(state2) := n2;
next(state2) :=
case
   (state2 = n2) & (state1 = t1): t2;
   (state2 = n2) & (state1 = n1): t2;
   (state2 = n2) & (state1 = c1): t2;
   (state2 = t2) & (state1 = n1): c2;
   (state2 = t2) & (state1 = t1) & (turn = 2):  c2;
   (state2 = c2): n2;
   TRUE : state2;
esac;
VAR
```

```
turn: {1, 2};


ASSIGN

init(turn) := 1;

next(turn) :=

case

   (state1 = n1) & (state2 = t2): 2;

   (state2 = n2) & (state1 = t1): 1;

   TRUE : turn;

esac;

SPEC

EF((state1 = c1) & (state2 = c2))

SPEC

AG((state1 = t1) -> AF (state1 = c1))

SPEC

AG((state2 = t2) -> AF (state2 = c2))
```

7. References

[1] UART RS-232 https://en.wikipedia.org/wiki/RS-232

[2] RS-232 specification http://www-ug.eecg.toronto.edu/msl/nios_devices/dev_rs232uart.html

[3] https://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[4] http://nusmv.fbk.eu/NuSMV/

[5] http://spinroot.com/spin/Doc/ieee97.pdf

[6] Swedish penalty code 8:8 (TWOC, unlawful dispossesion, “egenmäktigt förfarande”) http://www.regeringen.se/49bb67/contentassets/72026f30527d40189d74aca6690a35d0/the-swedish-penal-code#page=31

[7] Principles of Model checking - by Baier and Katoen https://mitpress.mit.edu/books/principles-model-checking

[8] Buffer overflow http://www.csc.kth.se/utbildning/kth/kurser/DD2395/dasak06/dokument/F4/F4.pdf

[9] Computer Security Dieter Gollmann

[10] Ross Anderson, Security Engineering

[7] Principles of Model checking - by Baier and Katoen https://mitpress.mit.edu/books/principles-model-checking

[8] Buffer overflow http://www.csc.kth.se/utbildning/kth/kurser/DD2395/dasak06/dokument/F4/F4.pdf

[9] Computer Security Dieter Gollmann

[10] Ross Anderson, Security Engineering

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

[12] http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C11_SerialInterface.htm

[13] http://chibios.sourceforge.net/docs/hal_stm32f4xx_rm/group___u_a_r_t.html

[14] Model checking overview http://www.cs.cmu.edu/~modelcheck/tour.htm