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 have considered the security properties of hardware peripherals using the NuSMV model checker. Our aim has been to build formal models of hardware peripherals and use the models to check a security property. We have applied formal models of the universal asynchronous transmitter/receiver (UART) and the serial interface RS-232.

We have done formal verification of the mechanism altera_avalon_uart[1] including its security policy. We also documented our findings about actual hardware configurations based on our theory. The models we use are statechart diagrams and their corresponding formulas written in NuSMV. One of our conclusions is that connections based on ticket-granting security models 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.

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

Terminology and Definitions

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

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

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

MMU - Memory Management Unit.

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 tool that can prove or disprove symbolic formulas

Nios2 - Nios2 is a brand name for a softcore CPU to which one can download a custom CPU design.

RAM - Random Access Memory.

RS-232 - Recommended standard 232 (IEEE) 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.

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

uClinux - Linux distribution suitable for embedded systems and for FPGA.

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. The model checkers can automatically answer the question whether a model satisfies the requirements. We use NuSMV to conduct formal model checking of hardware peripherals. The hardware peripherals we model are the altera_avalon_uart 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.

We cover model checking of these important security issues using NuSMV, its background and a typical formal model of the UART including its security model, both written in NuSMV. We won’t cover an entire detailed realization and use a simplified model instead[2].

If a user is not secure he or she will almost always stay insecure about that particular context and about that setting. 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).

1.2. 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.

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

An inexperienced computer user could also leave the UART interface open for root access and this is the case we can demo with a configuration between two computers. It is also common that the UART port is left open for debugging purposes which means that a hacker can get a login prompt and maybe even login without a password and get a root shell.

People breaking into their own devices could also find a serial console left in for debug purposes by the manufacturer available on an unconnected UART.

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. 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.

2. Aim, Questions and Limitations

We aim to answer the following questions:

1. How reliable is the UART connector?
2. What are the inputs of the UART connector?
3. What are the outputs?
4. Is there internal state?
5. How does the internal state change over time and, in particular, in one step?

We look at the specification which 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. Our model represents some features or the hardware such as the minimal set of necessary features of the avalon_altera_uart[3].

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.

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

2.1. Interface and Registers

Our UART core provides an Avalon-MM slave interface to the internal register file. The user interface to the UART core consists of six 16-bit registers: control, status, rxdata, txdata, divisor, and end-of-packet. A master peripheral, such as a microprocessor, accesses the registers to control the core and transfer data over the serial connection.

Our UART 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.

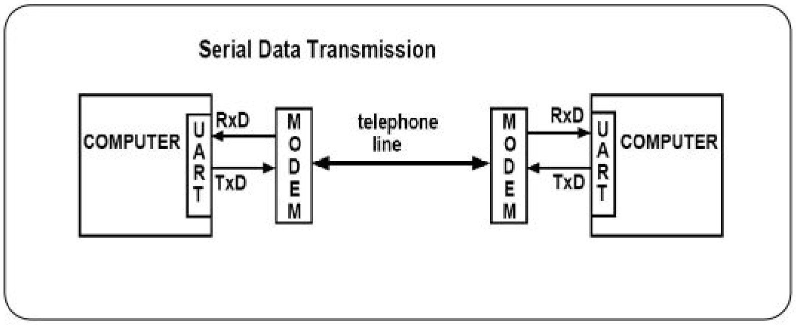


Figure 1. Block diagram of an UART serial transmission.

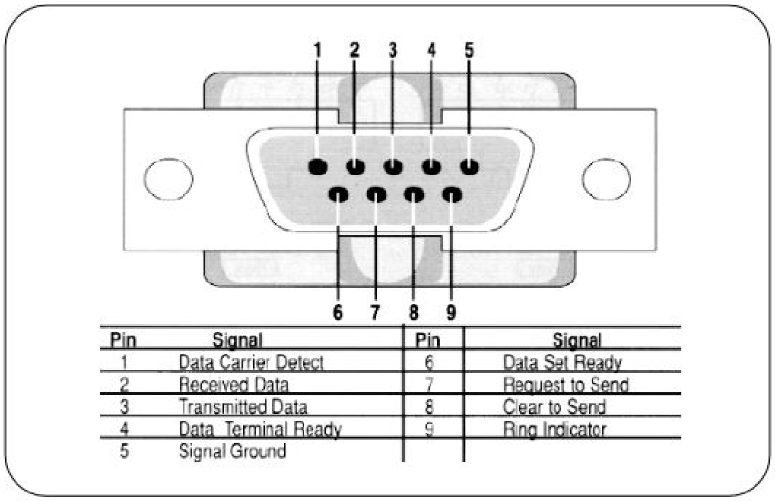
Our UART is, hence, used from the application context within an SoC to send and receive data to and from a hardware peripheral. The RS-232 interface has the following characteristics:

- 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 then name of the UART is avalon_altera_uart.

2.2. DB-9 Connector

Many computer users have probably seen a connector like this on the back of a PC.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figure 2. DB-9 connector.

The 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.

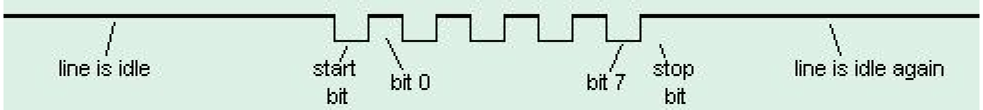
2.3. Asynchronous Communication

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.

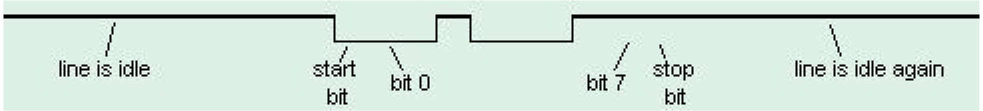
For example a byte 0x55 is transmitted as follows:



Byte 0x55 is 01010101 in binary.

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

Another example:



The data above is 0xC4.

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

The specific UART we have chosen is the one that is used in Altera FPGA, named altera_avalon_uart. In Quartus this UART looks according to the following:

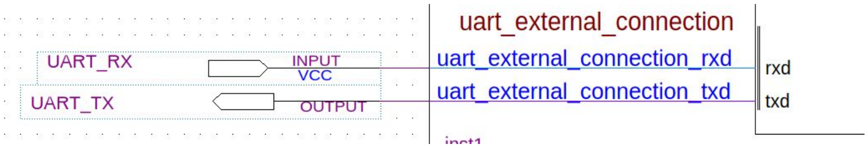


Figure 8. UART hardware in Qsys.

The product specification is according to the following:

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
Parity	Determines whether the UART core transmits characters with parity checking.
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.
Synchronizer stages	Synchronizer of stages

Include CTS/RTS	Include CTS/RTS pins and control register bits
Include end-of-packet	Include end-of-packet register
Baud rate (bps)	standard baud rates for RS-232 connections

The RS-232 settings are provided below for our connection from the Linux host that is used:

- Baud Rate: 115200
- Parity Check Bit: None
- Data Bits: 8
- Stop Bits: 1
- Flow Control (CTS / RTS): Off

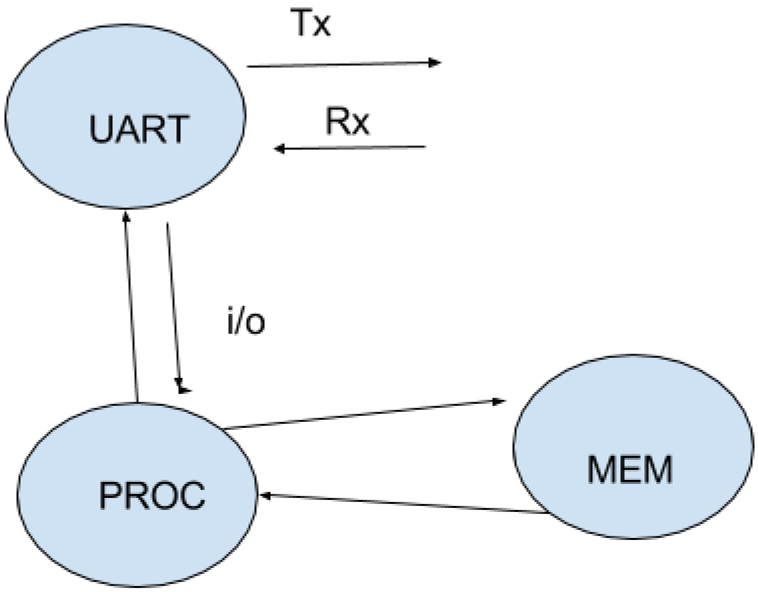
The RS-232 standard is used by many specialized and custom-built devices. This list includes some of the more common devices that are connected to the serial port on a PC. Some of these such as modems and serial mice are falling into disuse while others are readily available.

2.4. Scenario

Our scenario is that the UART can transmit and receive and there is a process with input and output to and from the UART and the process can also read and write to the memory of the computer. We build three models: UART, PROC and MEM.

We want to prove that K stays the same given that the process doesn't write to K.

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



2.5. 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.

3. The Security Traits of the UART

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.

3.1. Configuration

A simplistic outline of our theorem, that we are going to add more details to, can look as follows.

$$\begin{array}{c} \text{Conf} \\ \wedge \\ \models \text{Model} \\ \text{UART} \\ \wedge \\ \not\models \text{WRITE} \end{array}$$

The above theorem means that 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. By adding more details and specifics to the above theorem it can be useful without being overly simplistic and without being too complicated.

A simple overview of our model of hardware and its policy enforcement (the driver) can be seen in the following figure.

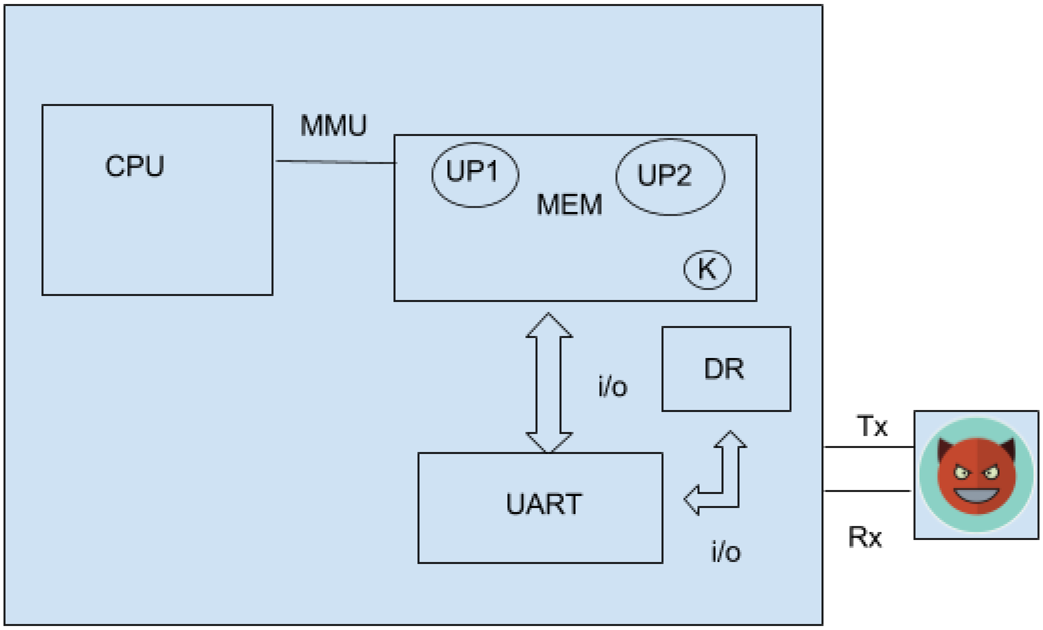


Figure 14. Hardware model overview.

The above is our simplified model of hardware host and the connected external hardware 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.

3.2. 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.

With the .sof file from Quartus and the uCLinux image we set up our environment:

```
$ nios2-download -g zImage

$ nios-terminal.exe 1
```

After the above procedures are done we have a development environment where two systems are connected via UART: One Altera DE2-115 which includes the UART connection and via a cable is connected to a peripheral system that tries to gain access via the RS-232 interface that UART enables.

3.3. Evaluation of Model Checking Tools

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

4. Specification and Methodology

Hardware circuits can be described in NuSMV. Hence, we 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.

4.1. Formal Verification

We include two models in our scope: The hardware devices and their states (1) and the software (2) (the driver and the operating system) delegated to be responsible for the security of the configuration. We include software for the reasons of most security vulnerabilities being due to software errors. In NuSMV a module can be instantiated as a VAR in other modules. The dot notation is used for accessing variables that are local to a module instance (e.g., m1.out, m2.out).

Our implementation declares arrays of arrays with a couple of lower/upper bounds for the index and a type.A module declaration can be parametric and a parameter is passed by reference.

4.2. Model Checking

Model checking is formally defined as $M \vDash \Phi$ where M is an automation model and Φ is a logical formula in CTL and LTL.

$M \vDash \Phi$ means that Φ is true in every structure in which M is true. $M \vdash \Phi$ on the other hand, means Φ can be proved using M as the premises. (In both cases, M is a not necessarily finite set of formulas and Φ is a formula.)

First-order logic simultaneously enjoys the following properties: There is a system of proof for which

- If $M \vdash \Phi$ then $M \vDash \Phi$ (soundness).
- If $M \vDash \Phi$ then $M \vdash \Phi$ (completeness).
- There is a proof-checking algorithm (effectiveness).
- Fortunately there’s a fast-running algorithm.

That last point is in stark contrast to this fact: 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[4].

[4] Alonzo Church’s theorem

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 σ_1 from q_1 to q_2 and q_3, we would write:

$$\{(q1,\sigma1,q2),(q1,\sigma1,q3)\} \subseteq \rho \text{ or } \{q2,q3\} \subseteq \delta(q1,\sigma1),$$

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 $q_1 \rightarrow_{\sigma_1} q_2$ the existence of a transition from q_1 to q_2 labeled σ_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.

4.3. RS-232 Interface

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.

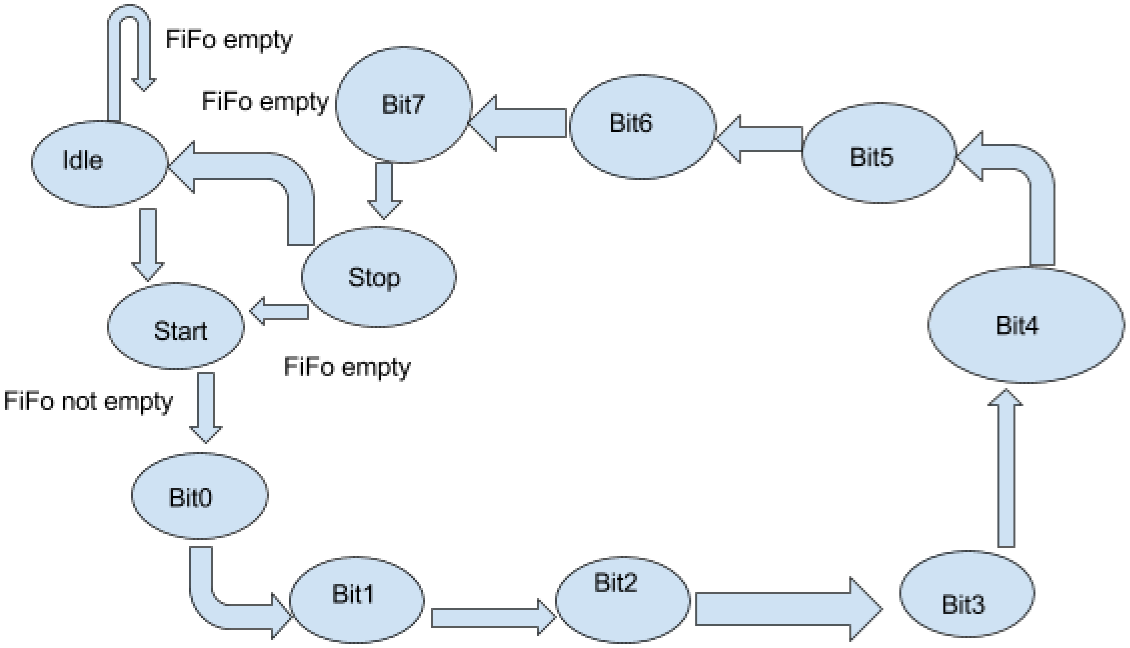


Figure 3. FSM diagram of the UART shift register.

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.

4.4. NuSMV modelling

The UART transmitter consists of a 7-, 8-, or 9-bit txdata holding register and a corresponding 7-, 8-, or 9-bit transmit shift register. Avalon-MM master peripherals 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.

4.5. UART Receiver Logic

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.

4.6. NuSMV modelling

Our 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:

```
TRANS

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

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

The NuSMV modules are supposed to represent circuits. Therefore it is not recommended to model a process or something else that is not a circuit. We created one model for the main memory and one model for the UART. The composition of modules is synchronous by default: all modules move at each step. We proved that the following formula is true given the premises:

LTLSPEC G (output != memory.data[1][0]) -> 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”. A safety property is distinct from a liveness property where one proves that something desirable eventually will happen.

4.7. NuSMV interaction

NuSMV is typically used interactively according to the following

```
NuSMV > reset; read_model -i models.smv ; go ;pick_state -v;simulate -v

Trace Description: Simulation Trace

Trace Type: Simulation

  -> State: 1.1 <-

       proc = read

       input = 0ud8_255

       output = 0ud8_255

       memory.K = 0ud8_255

       memory.data[0][0] = 0ud8_255
```

Step	uart0.Rx	uart0.Tx	uart0.input	uart0.output	proc	mem.K
0	0ud8_0	0ud8_0	0ud8_255	0ud8_255	read	0ud8_255
1						
2						
3						
4						
5						

4.8. Computer Security Patterns

Bruce Schneier, a famous expert in IT security, has specified a common and general IT security problem according to the following:

> *"Any security expert can invent a cipher that he himself can't break."*[5]

[5] Bruce Schneier...

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 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.

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.

4.9. Software Programming Model UART Core

We can select UART as a console when we build Linux for embedded systems. This flavor of linux is named uClinux.

In this example, the system contains a UART core named uart1 that is not necessarily configured as the stdout device. In this case, the program treats the device like any other node in the HAL file system.

5. Results and Conclusive Discussion

1. How reliable is the UART connector?
2. What are the inputs of the UART connector?

3. What are the outputs?
4. Is there internal state?
5. How does the internal state change over time and, in particular, in one step?

Any physical computer system can be accessed physically while being somewhat unprotected from technical attacks (for example physically pulling out the RAM module and by such means extracting the data from the physical RAM memory) and therefore we conclude that a computer user or the data will not be 100 % safe or completely protected in principle. The safest usage of the altera_avalon_uart today is to not leave to connector open. If the connector is open, like it was with the uClinux installation, root access is easy for anybody with physical access.

If someone has physical access it is game over and they can also pull out the dram modules (possibly after spraying them with freezer spray) and just read out the cryptographic keys, use JTAG, and many promote possible attacks.

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.

   A known problem with model checking is 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 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, less advanced than current research and more advanced than trivial.

We specified the problem and limited the scope to a small and simplified model of an UART RS-232 consisting of the parts.

UART doesn’t have a security model in its current availability. It seems up to the driver and os to delegate and react responsible.

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 stranger like X.25

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. NuSMV is well-suited for modelling hardware circuits. It is a language for verification of finite state systems (FSM). This language conducts checking of 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 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:

- 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.

6. Bibliography

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. http://nusmv.fbk.eu/NuSMV/
4. http://spinroot.com/spin/Doc/ieee97.pdf
5. Buffer overflow
   http://www.csc.kth.se/utbildning/kth/kurser/DD2395/dasak06/dokument/F4/F4.pdf
6. Computer Security Dieter Gollmann
7. Ross Anderson, Security Engineering
8. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf#page=68
9. http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C11_SerialInterface.htm

10. http://chibios.sourceforge.net/docs/hal_stm32f4xx_rm/group___u_a_r_t.html
11. Model checking overview http://www.cs.cmu.edu/~modelcheck/tour.htm
12. DESIGN AND IMPLEMENTATION OF CUSTOMIZABLE
    http://www.ijraet.com/pdf17/31.pdf
13. http://www.ijcst.org/Volume7/Issue4/p2_7_4.pdf
14. Security http://ijireeice.com/upload/2016/may-16/IJIREEICE%2071.pdf
15. The serial driver layer http://www.linuxjournal.com/article/6331
16. tty layer http://marcocorvi.altervista.org/games/lkpe/tty/tty.htm
17. altera_uart.c http://lxr.free-electrons.com/source/drivers/tty/serial/altera_uart.c
18. [http://lxr.free-electrons.com/source/drivers/tty/serial/serial_core.c
19.] NuSMV http://nuseen.sourceforge.net/