# Idea: Simulation Based Security Requirement Verification for Transaction Level Models

Johannes Loinig[1], Christian Steger[1],
Reinhold Weiss[1], and Ernst Haselsteiner[2]

[1] Institute for Technical Informatics, Graz University of Technology, Graz, Austria
{johannes.loinig,steger,rweiss}@tugraz.at
[2] NXP Semiconductors Austria GmbH, Gratkorn, Austria
ernst.haselsteiner@nxp.com

**Abstract.** Verification of security requirements in embedded systems is a crucial task - especially in very dynamic design processes like a hardware/software codesign flow. In such a case the system's modules and components are continuously modified and refined until all constraints are met and the system design is in a stable state. A transaction level model can be used for such a design space exploration in this phase. It is essential that security requirements are considered from the very first beginning. In this work we[1] demonstrate a novel approach how to use meta-information in transaction level models to verify the consistent application of security requirements in embedded systems.

## 1 Introduction

A lot of modern embedded systems need to provide security functionality. Faults in design and implementation of a system can cause serious security issues. Thus, careful security verification is needed. This is a considerable cost factor. External Common Criteria security evaluation (described later) can cost $100k and more and usually takes months. Finally discovered vulnerabilities in such a late development phase cause serious project delay and cost.

Security has to be considered from the beginning of a development process and in all abstraction levels [6,10]. To support this we propose a methodology to use meta-information in transaction level models (TLMs) for early and continuous security verification in a hardware/software codesign flow. TLMs are abstract functional models of the system. Iterative refinement is used for design space exploration until all system constraints are met. In each iteration our methodology supports security verification appropriate to the model's abstraction level.

The contribution of this work is a novel design and development approach that allows continuous security verification. System designers and developers will gain a better security understanding of the system which reduces the risk for a costly failed Common Criteria security evaluation.

[1] This paper is a result of a project which is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology (contract FFG 816464).

Ú. Erlingsson, R. Wieringa, and N. Zannone (Eds.): ESSoS 2011, LNCS 6542, pp. 264–271, 2011.

## 2 Related Work

Our proposed methodology is based on the Common Criteria (CC) security verification process. It applies the basic CC practices to verify TLMs. Thus, to explain our approach, we first summarize the needed CC and TLM basics. After that we shortly list advantages and disadvantages of formal verification methodologies to motivate our contrary simulation based approach.

### 2.1 The Common Criteria Process

The Common Criteria [3] is *the* de-facto standard for security evaluations of IT products. The entire process is too extensive to be described here. However, some basics should be clarified to understand our proposed approach.

CC is a rather documentation centric approach. Threats against the IT product are analyzed. Security Objectives are defined to counter these threats. Security Functions provide the functionality for them. Notice that Security Functions are rather a concept and not a concrete implementation. Each Security Function is composed of Security Mechanisms implemented in hardware or in software. Security Functional Requirements (SFRs) describe their functional requirements. These SFRs must be provided by the system to achieve the security objectives. The CC Security Target (ST) describes the relations of security threats, objectives, mechanisms, and functions. It is a design document for the security architecture of one certain IT product. In combination with design documentation an external CC evaluator uses the ST to judge if the security architecture is able to counter threats. This is a manual and extensive process.

The authors of [8] mention a lack of a clear relationship between the CC process and a system development approach. However, security engineering should be integrated in the system development process. [8] describes a CC conform UML based approach for security requirement engineering in a software engineering process. In comparison to that, our approach covers system development (hardware *and* software) and is not restricted to UML. Instead, we support implicit security modeling in a TLM of the system.

### 2.2 Transaction Level Modeling

Transaction level modeling allows development and analysis of system models [2]. The main concept is the abstraction and separation of the computational part of the system and the communication part of the system. In our approach we extend this by the security part of the system. Meta-information in TLMs allows an early estimation of e.g., the system's performance and power/energy consumption [11]. At the time of writing we were not able to find related work considering security requirements in TLMs.

### 2.3 Formal System Verification

There is no doubt that a formal verification (FV) would be preferable in comparison to our proposed simulation based verification. However, we think that

today one main aspect still acts against FV: FV is very costly [6] if applied on systems with realistic complexity.

As a consequence, FV can be applied on selected parts of a system only or it can be applied on very abstract meta-models of the system. This has been shown for software [4], for hardware [7], and on system level [1]. FV of selected parts only is unacceptable as system security can not be achieved by single separate modules in a system [10]. A meta-model based approach can be difficult to apply in a dynamic design process. The developer would need to work on both, the functional model and the meta-model.

Meta-models are typically written in special modeling languages and can be created manually as a first design step [4,1] or translated from other models by a verification tool [5]. Manual adaptation of the meta-model would be costly and error-prone. FV of a translated model means, that in a strict sense, not the system's model is verified but a model that was generated by the tool. The quality of the verification strongly depends on the translation tool and the capabilities of the modeling language of the meta-model.

The authors of [9] created formal templates for CC SFRs. These templates can be used for FV of the system's security specification. Again, not the system model itself is verified but its specification. In contrast to that our proposed simulation based methodology allows verification of the relationships of Security Mechanisms and SFRs against the functional system model under development.

## 3 Simulation Based Security Requirement Verification

In comparison to the work described in Section 2 our contribution is (1) the consideration and verification of security requirements in early design phases, (2) a verification on the basis on a TLM under development instead of a meta-model, and (3) a fast simulation based verification applicable for iterative design processes instead of a complete but extensive formal verification approach.

### 3.1 Iterative TLM Verification

Figure 1 shows the basic concept of simulation based verification in an iterative refinement process. A pure functional level is the basis for a TLM. In multiple iterations the TLM's modules become refined. During simulation of the TLM, when the test cases are applied, an automated verification generates a functional report and a security report. The reports are the basis for further design decisions. Another refinement cycle is necessary if not all constraints are met.

Different actions can be performed during TLM refinement: modules can be split in several smaller modules, combined to bigger modules, and mapped to software or hardware components of the system. All these actions require consideration of SFRs if the modules are related to Security Functions. To do so, the developer annotates the Security Mechanisms with meta-information about related SFRs. These annotations are evaluated during simulation and verification. If module modifications cause security violations this is reported during the simulation/verification and the developer can react immediately.

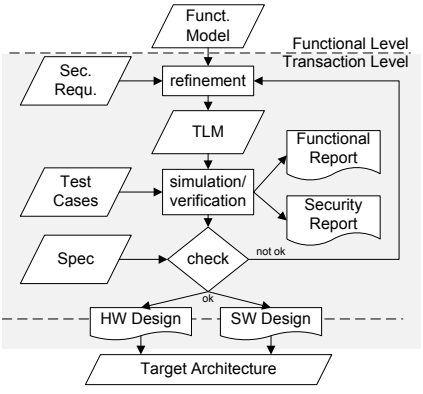


**Fig. 1.** Simulation based verification

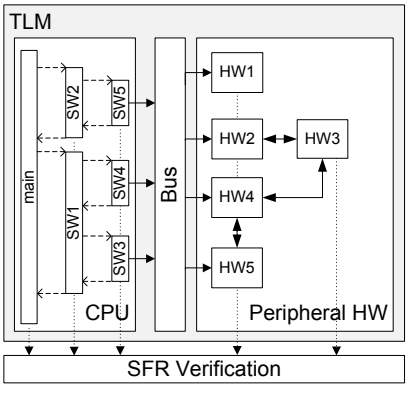


**Fig. 2.** TLM verification

### 3.2 HW/SW Verification Approaches

Figure 2 shows a very simple example of a TLM consisting of software modules (embedded in a CPU module) and peripheral hardware modules. Two things should be noted in this figure: the software is shown as a sequence of function calls whereas the hardware is modeled as a set of concurrent hardware blocks. Having sequential software and concurrent hardware is typical for TLMs. Thus, a verification approach has to be able to handle both concepts.

Requirement verification on sequential software is rather straight forward. A call graph clearly shows the dependencies of the security requirements of the software functions. This is not the case if hardware is modeled. Concurrent hardware processes do not provide a call graph as hardware functions usually never terminate. Additionally, hardware exceptions can interrupt the software at any time. This behavior is very difficult to describe in formal models. This is an essential reason why we chose a simulation based approach for our methodology.

Instead of a call graph, a data flow driven approach can be used for hardware verification. Data is passed from one process to another one. This can be done asynchronously - the data producer often has no influence if the receiver consumes the sent data or decides to ignore it. This might depend on the internal state of the receiving hardware module.

In a data flow driven approach data is annotated with SFRs instead of functions or modules. If this data is consumed, the annotated SFRs have to be evaluated by the simulation environment to verify the security capabilities of the module. As shown in the figure, software and hardware interact naturally (sketched as a bus connecting the CPU with the peripheral hardware blocks). As a consequence, SFRs have to be mapped from the call graph to the data flow graph and vice versa.

In our simulation based approach function calls and data generation respectively data consumption is reported to a security verification module. The reporting includes the annotated SFRs of the acting software or hardware modules.

The verification module evaluates the relationship of the reported SFRs and is able to report occurring security violations.

### 3.3 Verification Rules

Yet we have not discussed the rules our proposed verification module should apply to the reported SFRs. Different sets of rules according to the abstraction level of the TLM are imaginable.

A straight forward approach is a *requires*/*implements* scenario. Modules that require SFRs are annotated with the SFR's identifier (e.g., a unique number) and a *requires*-flag. Modules that provide the functionality described by the SFR are annotated by the SFR's identifier and an *implements*-flag. The verification module checks if every call graph's node with a *requires*-flag leads to nodes with the according *implements*-flag; respectively if data that has a *requires*-flag is handled in modules that provide the *implements*-flag.

A more sophisticated approach is to annotate software and hardware modules that represent a Security Function and also modules implementing SFRs. The ST can be used to extract the relationship of (1) SFRs required by the Security Functions and (2) SFRs that have dependencies on other SFRs to create verification rules. The transaction level model consists of modules assigned to the software and the hardware domain. It represents the functional behavior and the security behavior of the embedded system. Certain modules might be implemented on different abstraction levels. Communication interfaces, for example, can be modeled on very high abstraction levels as they are usually not security relevant. The software model reports information to the verification module to generate call graphs for Security Functions. Similarly, the verification module generates data flow graphs for data in the hardware model. Call graphs and data flow graphs are verified against the rules extracted from the ST. This is shown in Figure 3.

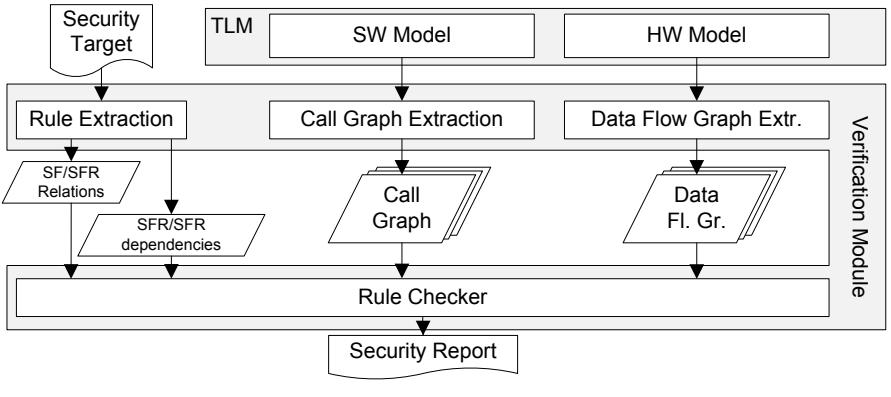


**Fig. 3.** Verification of rules extracted from a Security Target

## 4 Proof of Concept Implementation

We implemented a proof of concept verification library for SystemC[2] TLMs. The library provides (1) macros for annotating functions, data, and SystemC processes with SFRs, and (2) the verification module. The macros cause function calls of the static verification module instance. These function-calls cause the verification module to compute the call graphs and data flow graphs and to store the annotated SFRs. The rule set for verification is static and provides checks for *requires*, *implements*, and *supports* flags for SFRs. Modules that *require* SFRs have to utilize modules that *implement* according SFRs. *Supporting* SFRs can be used if interconnecting modules are needed.

So far, we have not evaluated our approach on a use case with realistic complexity. Instead, we first evaluated the general applicability by verification of different smart card STs from different vendors: STMicrosystems ST23YR80A, NXP P5Cx081V1A, Samsung S3CC91A, Infineon SLE66CX680PE, and Fujitsu MB94RS403. These STs are public[3].

In addition we implemented a very small use case example based on some few hardware modules: a CPU, a memory, a DES crypto co-processor, and a CRC co-processor. The software model reads DES key data from the memory and sends it to the DES co-processor. We chose the FDP_SDI.1.1 SFR from the Common Criteria standard [3] - it basically defines, that the DES key has to be integrity protected. The `setDESKey()` function was annotated to *require* FDP_SDI.1.1. Two different implementations to fulfill the security requirement were evaluated: (1) a software implementation calling `checkIntegrity()` within `setDESKey()` and (2) a hardware implementation connecting the DES co-processor with the CRC co-processor.

## 5 Results and Discussion

Table 1 shows the summarized evaluation results of the STs: the total number of defined Security Functions, the number of selected SFRs, the number of applied SFRs, and the SFRs we think that can be checked automatically by our proposed approach. Notice, that each SFR can be applied on several Security Functions. Thus, the number of applied SFRs can be higher than the number of SFRs. In a strict sense, this number should be even higher than the numbers depicted in Table 1 (third row) because each Security Function consists of several mechanisms. However, the number of Security Mechanisms is depending on the concrete implementation and not given in public STs. Therefore, we took the number of Security Functions as an estimator for our evaluation (we assume that each of them consists of one mechanism in minimum).

26 to 60 SFRs were applied in the selected smart card microprocessors. This should give a feeling about the verification complexity: 26 to 60 applied SFRs means that 26 to 60 times a module has to be verified if it provides the

[2] `www.systemc.org`
[3] `www.commoncriteriaportal.org/products/`

**Table 1.** Security Target evaluation results

| | STMicrosystems | NXP | Samsung | infineon | Fujitsu |
|---|---|---|---|---|---|
| Number of Security Functions | 10 | 9 | 5 | 9 | 10 |
| Number of SFRs | 15 | 16 | 18 | 21 | 19 |
| Number of applied SFRs | 40 | 60 | 26 | 35 | 26 |
| Potentially automated check | 90% | 85% | 73% | 77% | 73% |

appropriate SFR. Notice, that the selected STs describe smart card hardware only. If the software has to be verified as well even more Security Functions and SFRs will emerge and the verification effort will be even higher.

We do not expect that all applied SFRs can be verified with our proposed approach. Thus, we checked which used SFRs can be verified by our method. To do so, we evaluated the meaning of used SFRs in the selected STs and checked if we could define rules for our approach that are able to verify the SFRs automatically. Because of the limited space in this publication we can not explain this process in details here. The results are shown in Table 1 in the last row.

We think that 73% to 90% of the used SFRs could be checked with our proposed approach. Accordingly, the verification effort could be reduced by approximately 80%. Of course this is a very optimistic estimation as verification does not only mean to check if a module provides the right SFRs. However, we think that this clearly shows the potential of an automated verification approach.

From our very small case study we can summarize the following results. Without going into the implementation details we can note that the usage of our annotations is very intuitive and fits very well into the iterative design flow of an hw/sw codesign flow. However, we also have to mention that a puristic *requires*/ *implements* rule-set seems not to be sufficient. At least an additional *ignores*-flag was needed to avoid miss-leading incorrect security violations. However, we think that such an *ignores*-flag can come with a high risk of erroneous miss-usage.

In addition, we have to note that the implemented solutions are not identical from a security point of view. If the integrity check is performed in software, sending the sensitive key material to the DES block is unsecured. On the one hand we think that this could be automatically reflected in our verification result (e.g., as the 'distance' of a module that *requires* the SFR to the module that *implements* the SFR). This could help the developer during the design space exploration phase. On the other hand, if needed, there exist certain SFRs that reflect such requirements. FDP_ITT.1.1, for example, requires system module intercommunication to be protected as well.

As a summary we have to conclude that much more use case studies have to be done to extract a rule set that allows a sufficient verification of the SFRs.

## 6 Conclusion and Future Work

In this work we described a simulation based verification methodology for security requirements in transaction level models of embedded systems. The basic

idea has its roots in the Common Criteria process where Security Functions and according Security Functional Requirements are defined. We showed that our proposed approach is able to verify such requirements by evaluating meta-information in the model during the simulation of its functional behavior. Furthermore, we showed that gained cost savings can be significant.

However, we have to say that our work is in an early stage. More case studies are needed to evaluate the applicability of our methodology. Especially the verification rules have to be refined and their influences on the design have to be evaluated.

## References

1. Balarin, F., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, A.L.: A formal approach to system level design: metamodels and unified design environments. In: Third ACM and IEEE International Conference on Formal Methods and Models for Co-Design. IEEE, Los Alamitos (2005)
2. Cai, L., Gajski, D.: Transaction level modeling: an overview. In: Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. ACM, New York (2003)
3. Common Criteria. Common Criteria for Information Technology Security Evaluation - Part 1-3. Version 3.1 Revision 3 Final (July 2009)
4. Deng, Y., Wang, J., Tsai, J.J.P., Beznosov, K.: An approach for modeling and analysis of security system architectures. IEEE Transactions on Knowledge and Data Engineering 15(5), 1099–1119 (2003)
5. Garavel, H., Helmstetter, C., Ponsini, O., Serwe, W.: Verification of an industrial SystemC/TLM model using LOTOS and CADP. In: 7th IEEE/ACM International Conference on Formal Methods and Models for Co-Design. IEEE, Los Alamitos (2009)
6. Kocher, P., Lee, R., McGraw, G., Raghunathan, A.: Security as a new dimension in embedded system design. In: Proceedings of the 41st Annual Design Automation Conference. ACM, New York (2004)
7. Lotz, V., Kessler, V., Walter, G.H.: A formal security model for microprocessor hardware. IEEE Transactions on Software Engineering 26(8), 702–712 (2000)
8. Mellado, D., Fernández-Medina, E., Piattini, M.: A common criteria based security requirements engineering process for the development of secure information systems. Comput. Stand. Interfaces 29(2), 244–253 (2007)
9. Morimoto, S., Shigematsu, S., Goto, Y., Cheng, J.: Formal verification of security specifications with common criteria. In: Proceedings of the 2007 ACM Symposium on Applied Computing. ACM, New York (2007)
10. Schaumont, P., Verbauwhede, I.: Domain-specific codesign for embedded security. Computer 36(4), 68–74 (2003)
11. Trummer, C., Kirchsteiger, C.M., Steger, C., Weiss, R., Pistauer, M., Dalton, D.: Automated simulation-based verification of power requirements for systems-on-chips. In: 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems. IEEE, Los Alamitos (2010)