Certified Translation in TXS Core

Richard B. Kreckel
Framatome GmbH
BU Instrumentation & Control
Erlangen, Germany
richard.kreckel@framatome.com

Alexandre Bérard
Université Grenoble Alpes
CNRS – Verimag
Grenoble INP, France
alexandre.berard2@univ-grenoble-alpes.fr

***Abstract*—In the civil nuclear industry, the lingua franca for the specification, design, and maintenance of digital instrumentation and control (I&C) systems are function block diagrams (FBDs), interconnecting instances of pre-qualified function block types. In addition to thorough verification & validation (V&V), the industry's standards and regulatory bodies emphasize the semantic preservation of the translation from FBDs to executable machine code. We outline the steps taken in Framatome's TELEPERM XS (TXS Core) digital I&C product platform to achieve rigorous and automated verification of semantic preservation in a fully automated toolchain.**

***Index Terms*—software verification, compilation, function block diagram, safety I&C, PLC toolchain**

I. Introduction

At the user level, Framatome's TXS Core toolchain begins with a graphical editor: I&C engineers drag graphical FBs from a menu of over 100 different pre-defined and qualified FB types, drop them onto function block diagrams (FBDs), and interconnect them with signals representing the dataflow between the FBs and I/O.

The subsequent process of translation to machine code consist of (i) generation of linearly structured ISO C code and (ii) compilation to ARMv7 machine code (Fig. 1). No user intervention whatsoever is required in this fully automated process (in fact, it is not even permitted). Below, we will unfold them and the methods to ensure semantic preservation across the entire toolchain.

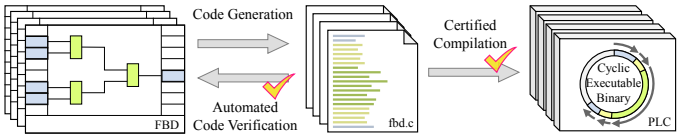

Fig. 1. The TXS Core toolchain translates FBDs into certified binaries.

The three main steps taken over the last decade towards a high-performance toolchain with a robust case for semantic preservation are: (i) the introduction of automated code verification in TXS Core 4.0 (Kreckel, Framatome), (ii) the switch to the CompCert [1] certified compiler for the toolchain's second stage in TXS Core 4.5, and (iii) our specific improvement of CompCert for specific code patterns as they appear in the C code generated by the TXS code generator (Bérard, ongoing work).

Assuring semantic preservation is a highly valued asset of safety-related toolchains for use in nuclear power plants [2]. We assert its complete and automated implementation in the TXS toolchain without any burden on users.

II. Translation to ISO C Code

Using the persisted graphical FBDs as input, a code generator first produces ISO C code following a defined grammar. That code consists almost entirely of function calls to the standardized, qualified, and pre-compiled function blocks and of code representing the dataflow between the blocks.

Diverse Back Translation of Generated C Code

Framatome has developed a fully automated verifier operating on the persisted graphical FBDs and on the generated code as input. It exploits the fact that an FBD is a directed acyclic graph connecting FB ports (the 'netlist'). Every signal line connecting two FBs is equivalent to the constraint of causality: In every PLC execution cycle, the source FB_{src} must be invoked before the destination FB_{dst}. Within the generated code, causality is equivalent to the constraint that the source FB_{src} must be called before the destination FB_{dst}. This induces a strict weak ordering of FBs. The generated ISO C code can be verified element-by-element:

- From the input FBDs and the generated code, construct the sets of all FB instances, properly attributed. Verify that the two sets are equal.
- From the FBDs and the generated code, construct the sets of all signals, properly attributed by source and destination(s). Verify that they are equal.
- For each signal, verify that its FB_{src} is computed before its FB_{dst} in the generated C code, i.e. that their weak ordering is preserved.

Successfully performing the above steps proves that the semantics of the FBD is indeed preserved in the generated application software: The structures are the same and the generated code respects all the FBD's causal relations.

Since version 4.0, the TXS Core toolchain includes an automated verifier covering the entirety of the generated C code. Its implementation is fully diverse to the code generator so to exclude common faults.[1]

[1]Full diversity also implies a different implementation language.

III. Certified Compilation to Machine Code

Compilation to optimized machine code is a complex task which, taking place *after* static analyses, is safety-critical. Nuclear codes and standards pose very strict requirements on verification & validation (V&V) of the final products to rule out the absence of any translation error [2]. Increasing the trust in the compiler is also important. Amongst others, it can significantly support the introduction of local functional changes to FBDs at later stages of the product lifecycle, for instance during a plant revision outage. For this reason, TXS Core uses the CompCert certified compiler for producing machine code since version 4.5. (Studies subjecting compilers to synthetically generated C code have independently confirmed that the rate of bugs in CompCert is orders of magnitude below that of usual compilers [3], [4].)

IV. CompCert Improvement

The C code generated by the TXS code generator from FBDs is linearly structured. It contains sequences of assignments from source signal to destination signal:

```
dstC1 = srcA1;
dstC2 = srcB2;
. . .
```

Each line assigns only a few bytes of data but it is not uncommon to see many thousands of such assignments in a row. For this code pattern, the baseline CompCert compiler generates machine code which is roughly twice as large and slow as that generated by the GNU compiler. Since all assignments are evaluated in each computing cycle (typically every 50 ms), they offer a significant optimization opportunity.

We used Chamois, a fork of CompCert's official releases maintained by Verimag in Grenoble [5]. Chamois includes more advanced code optimizations and a framework that facilitates their writing.

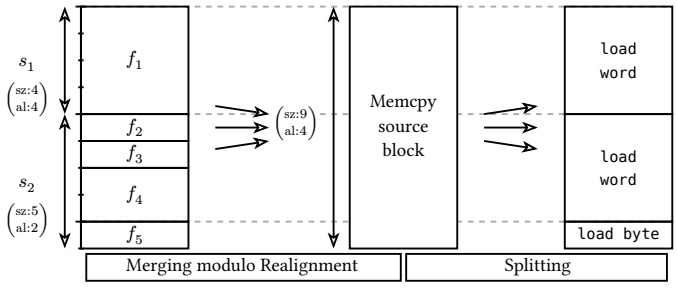

Fig. 2. Memcpy fusion and splitting (sz: size, al: alignment).

The new optimization is shown in Fig. 2. First, memory copies with adjacent addresses are merged into an internal bigger 'Memcpy' block, retaining the global alignment (a power of $2 \in \{1, 2, 4, 8\}$). Finally, this block is split into atomic and aligned parts which can be directly translated to hardware instructions [6]. Where they are unaligned, a short prologue and epilogue are added for establishing alignment. As all CompCert optimizations, these steps are formally proved correct in the Rocq prover.

We have benchmarked the result using original TXS generated code and hardware for a reactor protection system. On average, the resulting machine code of these sections is 1.9 times smaller and faster than the one generated by baseline Chamois and on par with the one generated by GCC.

Effect on CompCert's Trusted Computing Base

By normalizing all Memcpy instructions into atomic copies, we avoid the complex and unsafe translation of general case Memcpy, located in CompCert's trusted computing base[2] (TCB) [7]. We replaced this unsafe translation by a significantly simpler one with proof of correctness: following the same principle as assembly printing, it is just one-to-one translations of each case of atomic Memcpy instruction, safely created with fully proved methods.

V. Summary and Conclusion

In the nuclear industry, formally verifying that the translation from FBDs to the executable code preserves the semantics can not obsolete comprehensive validation of a final I&C system. Still, a complete certified toolchain is crucial – it justifies the trust placed in model checking and simulations. After all, these disciplines are almost always exercised directly on the FBD specifications.

For the most safety-critical toolchains, constructive formal methods increasingly complement established qualification methods based on tool experience and extensive testing.

On the other hand, one cannot ignore performance. In this context, the CompCert compiler is a mature tool, suitable for industrial use in highly dependable systems like TXS Core. It provides reasonably competitive compiler optimizations. Where it falls short of that, it can be improved.

We emphasize that while improving CompCert for specific cases, care should be taken not to increase its TCB.

References

[1] X. Leroy: "A Formally Verified Compiler Back-end" In: Journal of Automated Reasoning vol. 43.4, pp. 363–446 (2009) doi:10.1007/s10817-009-9155-4

[2] Regulator Task Force on Safety Critical Software (TF SCS): "Licensing of Safety Critical Software for Nuclear Reactors: Common Position of International Nuclear Regulators and Authorised Technical Support Organisations" (2026) https://www.onr.org.uk/media/i2anr3nd/software.pdf, last accessed 2026/07/20

[3] X. Yang, Y. Chen, E. Eide, J. Regehr: "Finding and Understanding Bugs in C Compilers" ACM SIGPLAN Notices, vol. 46.6, pp. 283–294 (2011) doi:10.1145/1993316.1993532

[4] V. Le, M. Afshari, Z. Su.: "Compiler Validation via Equivalence Modulo Inputs" ACM SIGPLAN PLDI'14, pp. 216–226 (2014) doi:10.1145/2594291.2594334

[5] D. Monniaux, S. Boulmé: "Chamois: agile development of CompCert extensions for optimization and security" In: 35es Journées Francophones des Langages Applicatifs (JFLA 2024) https://inria.hal.science/hal-04406465

[6] A. Bérard., B. Bonneau, S. Boulmé, D. Monniaux: "Formally Verified Compilation for Aligned Memory Copies" (2026), https://hal.science/hal-05493413

[7] D. Monniaux, S. Boulmé: "The Trusted Computing Base of the CompCert Verified Compiler" In: Programming Languages and Systems. Lecture Notes in Computer Science, vol. 13240, pp. 204–233. Springer, Cham (2022) doi:10.1007/978-3-030-99336-8_8

[2]The TCB of CompCert is the set of components whose correctness must be trusted without formal proof.