Towards HW-SW Co-Design for Secure Industrial Real-Time Ethernet Applications

Hans Dermot Doran, Sven Schneider, Judith Meisterhans,
Jöel Bronwasser, David Ganz
Institute of Embedded Systems
Zurich University of Applied Sciences
Winterthur, Switzerland
{doran, scdv, mesh, benl, ganz} @zhaw.ch

Stefan Eberli
Duagon AG
Dietikon, Switzerland
stefan.eberli@duagon.com

***Abstract*—Real Time Ethernet protocols are currently undergoing specification updates to account for security features. These features generally taking the form of a public key infrastructure and associated network node and message authentication and, occasionally, message encryption. The computational expense of authentication and encryption in hard real-time applications adds substantial expense in both initial implementation and post-commissioning maintenance. In this body of work, we seek to manage this cost and complexity by the use of high-level synthesis to generate field programmable gate array IP from an open-source security stack. We detail the motivation, first steps in the establishment of a process and first results.**

Keywords—Real Time Ethernet, Network Security, High Level Synthesis, FPGA, HW-offloading

I. Introduction

The application domain of embedded distributed control includes strategic infrastructure such as water purification, fuel refining, energy generation and conversion and factory automation. These application domains are increasingly subjected to malicious network attacks ([1], [2].) The Stuxnet experience ([3], [4]) – has left industry in little doubt that a distributed network is just as vulnerable as the individual nodes attached to it. The result is that real-time Ethernet (RTE) protocol promoters have finalised or are finalising appropriate specifications for securing distributed embedded network traffic. We situate our work in the context of an FPGA-based RTE communication controller [5].

The addition of security represents a substantial additional computational expense, especially for hard real time protocols. An expense, anecdotal industry experience suggests, end-users are not especially keen on underwriting. The device manufacturer must bear the burden of what is a two-pronged cost-structure, the (specific) application, that is the hard- and firmware of the device and secondly, the (partially generic) implementation and maintenance of a network security sub-system. Both expense and implementation time-frames can be considerable. Whilst the defect rate of an application can be expected to converge to zero after a number of maintenance iterations, that of a security system cannot, resulting in long-term maintenance effort - the primary motivation for this body of work.

We gratefully acknowledge the financial support of the Swiss Innovation Agency, Innosuisse, under project grant 42637.1 IP-ENG

To approach this issue we derive HW off-loadable cryptographic code from an established codebase such as Mbed TLS [6]. We synthesize the code into HW using high level synthesis (HLS) tools but develop a ruleset for iterative optimisation. Our conclusion is that a library of cryptographic functions derived from this codebase would be of substantial service to industry. Our novelty is the establishment of a ruleset and the use of an established baseline of code.

We explain the motivation and aims for the deployment of a hardware software co-design process in the following subsections. In Section II we outline work and results to date and in Section III we discuss the results and future work.

A. State-of-the-Art

There are numerous technologies available for offloading the computational expense of cryptographic algorithms for embedded systems. These include secure elements [7], priced on a per-piece basis and currently recognised as unsuitable for reaching hard real time deadlines [8]; processing elements tightly coupled with the instruction set architecture of a general purpose CPU [9] and intellectual property (IP) whose target is either application-specific integrated circuits or field programmable gate-arrays (FPGAs) [10], [11]. The final category are hand-coded implementations or ASIC designs, research articulations of which can be found in publications such as [12]. A particular disadvantage of commercial IP is the additional cost. IP is typically licensed per device family or FPGA-type and a company with multiple products sold in small numbers, as is typical in factory automation, will find this kind of cryptographic solution expensive. Hand-coding a solution represents a development risk. A simple, yet impractical, solution is to synthesise the entire TLS stack into HW. Salient parts of the stack, including cryptographic algorithms can however be offloaded via high level synthesis tools. There has been substantial activity in this area, [13] provides a useful overview. The corner points show some merely establish the general feasibility [14] others [13], [15] simply experiment with self-derived baseline code implementations. [16] come closest to tracing optimisation iterations like we do but no publications take higher level considerations like code complexity into account.

The challenge of cryptographic computational expense can clearly be addressed [17]. The challenge in implementing secured communications is clearly not in the domain of cryptography but in integrating that cryptography into an implementation of high integrity and maintaining it over an

expected lifespan of 20 years or more. This issue motivates the consideration of a co-design process to implement security in an embedded networked device. Our novelty is to propose a process using a well-known source code as a baseline.

II. Co-Design

A. Implementation Architecture

Obfuscation is not conducive to security. The "many-eyes" principle facilitates rigorous inspection; the discovery of vulnerabilities introduced by implementation; assuming regular maintenance updates and fixes the longevity of the implementation. It follows that a well-regarded open-source implementation of a security protocol offers these benefits.

1) Motivation for HW-SW Co-Design

A number of offloading options have been looked at in a previous subsection, they share the common denominator that they are disembodied from the control-flow of the security stack. This disembodiment may introduce additional vulnerabilities. For instance, in a secure elements design, the communication to the secure element must be secured and the base security-stack modified. Over multiple maintenance cycles the potential for implementation-specific vulnerabilities is introduced as the implementation code departs from the open-source codebase. This in turn may introduce anomalies into the validation process across a product range. These considerations warrant an attempt to apply a more linear approach which we find in the application of specification driven HW-SW co-design.

2) Interpretation of HW-SW Co-Design

HW-SW co-design was once taken to mean that HW and SW engineers were co-located in offices and the project would benefit from continuous cross-disciplinary contact [18]. The meaning of the concept has mutated in different directions including specification-driven HW-SW co-design which is taken to mean synthesis of a solution from one specification [18]. Despite many alternatives, there is no established representation for this specification so it is legitimate to begin with a specification expressed in C-code and synthesise HW and SW from this specification [19]. We propose using the Mbed TLS stack as such a specification. The attractiveness is that one starts from a code-base that exhibits a history of being well-maintained and is opensource albeit, due to its history, the code is optimised for general purpose processors.

B. HW-SW Co-Design Context Implementation Process

1) Architecture Constraints

Synthesising code from a single specification written in C-code in the context of involves allocating components, binding functions to these components, performing the synthesis and then measuring performance [20]. The target hardware is an Intel FPGA board (5CEFA4U19A7 [21]) fitted with an embedded NIOS processor, operating eCOS and running a PROFINET stack. The HW synthesis tool is chosen as the HLS tool from Intel, the SW synthesis tool is the GNU compiler in an Intel/NIOS II version [22].

2) HW-SW Co-Design Optimisation Parameters

Typical HW-SW co-design optimisation parameters are (monetary) cost, latency and power. The architecture constraints determine the monetary cost. The target latency can be derived from the operating parameters of the example RTE protocol, PROFINET. Standard PROFINET IRT upholds cycle times down to 250 µs supporting one asynchronous and one IRT frame. It follows that authentication must be completed within 125 µs. In an FPGA the power consumption is largely determined by the circuitry within the FPGA and the monetary cost of the FPGA is commensurate with the number of logic elements (LEs) within the FPGA. The currently used FPGA features 47k LEs and an occupancy of 30%. Whilst a commercial footprint-optimised cryptographic functional unit such as one from Xiphera [11] will fit comfortably, so will a substantially larger IP. In terms of optimisation target, the HLS generated authentication functions may occupy up to 32k LEs.

3) Allocation and Binding

The allocation (hardware component FPGA) has been pre-determined by the necessity of meeting hard real-time deadlines.

We must determine which functions are to be bound to the FPGA. In order to bind functions to FPGA we need to partition the C-code which acts as specification. It follows that the partition points must be determined and to proceed in a structured fashion, key performance indicators should be determined. Examining McCabe's measure of complexity is instructive [23]. McCabe reduces a function to a control flow graph which is used to determine the number of unit tests, and in a second reduction step, the interface tests a function requires. We use the measure of complexity to decide whether a function should be implemented in HW or SW. In the Mbed TLS stack the computationally expensive code, that is the cryptographic code, is packed in one or two functions called by a wrapper function. This wrapper function is called in three (control) paths of the Mbed TLS stack. The block of code of these paths features a McCabe complexity of >500, 5 is commonly considered human inspectable code. It is clearly impractical to attempt to synthesise this entire block. We therefore focus first on the on the feasibility of synthesis of the cryptographic functions and note that this wrapper function has a defined interface with a small number of input parameters. We then begin synthesising the SHA256 cryptographic code from the TLS stack, applying refactoring in each iteration. We also track the process. The results are shown in Figure 1 which illustrates the progression of the initial HLS conversion (bottom left) through successive code re-factorings to the final version (top right.) This graph represents something of a random-walk through various code refactorings to determine a ruleset providing cost-efficient and deterministic iterations. Having established, and documented, lessons learned, we apply these to a number of other algorithms, refining the ruleset as we go along. In some cases, we can reduce the number of iterations to four (Figure 2). Two points require further examination - the first concerns the optimisation parameters. The second concerning the ground truth of the baseline code. Both are performance related.

C. Performance Characterisation

1) Ground Truth

The Mbed TLS cryptographic code, is optimised for x86 architectures. The execution efficiency of this code on a HW platform is not clear.

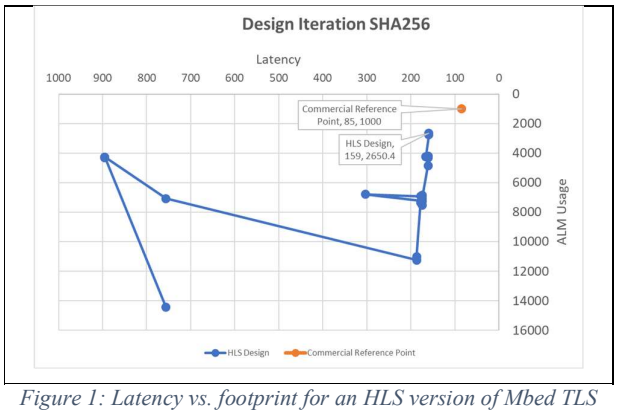

Figure 1: Latency vs. footprint for an HLS version of Mbed TLS SHA256 code for a number of refactoring iterations.

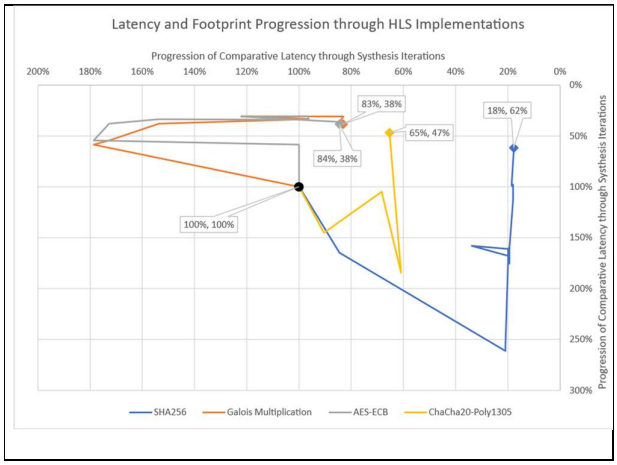

Figure 2: Latency vs. footprint for HLS versions of Mbed TLS cryptographic code for a number of refactoring iterations as percentages of the original synthesis.

To gain a better understanding we implement a naïve version of our target cryptographic algorithm, AES in discrete functions. We build a C-Unit test-set for regression-test purposes. Having verified that the software executes correctly we synthesise it as-is into HW using the HLS tool. The initial footprint was determined at ~100'000 logic elements, substantially worse than achieved by [14] for a non-optimised naïve interpretation of AES-GCM and [15] who progress through the optimisation of a naïve version of AES. The substantial number of LEs are largely attributable to the interface between individual AES functions being handled over registers. Progressing through optimisations of naïve code is already state-of-the-art and no new insights were anticipated, so we cease this line of investigation. What is clear is that this inter-function interface is not accounted for by McCabe's complexity measure. Clearly additional complexity performance indicators are called for. We tabularise the results of our Mbed TLS AES-encryption synthesis against a commercial version and [15] in Table 1 below. We provide two measures, one with the calculation of the expanded key (AES Full-IP) and one without (AES Function.)

2) Real-World Optimisation Parameters

The setting or determination of optimisation parameters is by the nature of this context, empirical. Many companies manufacturing embedded devices are faced with variant management, that is they develop, manufacture and sell variants of a product at low to medium volumes rather than a single product at high volumes. Integrating security features on a bespoke level across a number of variants of a single product line can become an expensive undertaking especially if the binding of the security function is different for each product variant. In low-medium product manufacturing there is a limited business case for extreme component-cost optimisation. The extra cost of a larger component purchased at low volumes measured weighted against the portability of a license-free security solution across the product-line, or indeed multiple product-lines, is generally low.

Design	Size (ALM/LUT)	Latency	Size %	Latency %
Xiphera (XIP1101B)	1629 / 1615	100 ns	100 %	100 %
[15]	/ 2634	4.54 µs	163 %	4540 %
AES – Full IP	5478 /	5.36 µs	360 %	5360 %
AES-Function	3573.2 /	1.09 µs		

Table 1: Comparison of Commercial and HLS Variants of the Mbed TLS AES Algorithm Implementation

Set against these concerns, the predictability of synthesised results is clearly a concern. For instance, both the footprint and the latencies of the synthesised solution are larger than the commercial reference implementation [11]. Post-hoc discussions with the industrial partner determined that these are acceptable in the context of the immediate practical application. This may not be the case for other applications. More specifically, there is yet no catalogue of synthesised code that would allow the embedded-device manufacturer to estimate the footprint and achievable latencies. With such an ecosystem, the implementation risk for the device manufacturer, whether HW-SW Co-design is used or not, can be substantially reduced.

3) Validation

Security validation of a system is problematic. Absolute statements on security are by its nature, difficult to evidence. The depth of security has clearly been defined in the Real Time Ethernet industry. The chain from vendor electronic datasheet file down to the RT traffic is geared towards integrity of a RTE installation by securing the integrity of individual nodes.

We believe that by co-synthesising code from a well-recognised source like Mbed TLS, facilitates establishing the SW equivalency of the final solution. Selected C-functions are synthesised, the rest are compiled for execution on an attached CPU. The interfaces are replicated one-to-one on a HW layer. The resulting HW-SW implementation should be functionally identical and hopefully more performant than a pure SW implementation on a comparable CPU. We expect differential testing to show this to be the case [24].

III. Discussion

A. Results

We have presented an industrially viable HW-SW co-design process for secure communications implementation. We believed to have achieved reasonable results, when viewed in absolute terms and highly cost-efficient results for implementers in the low-mid volume manufacturing range. From a technical-tactical point of view this can support the manufacturer/implementer in providing a clear path from

recognised and well understood software solutions to strongly related and offloaded hard-real time solutions. This path ends at the security validation which we support using a differential test framework.

Whether the reported performance characteristics are useful for other implementers is a matter for those implementers. This body of work is also in an early stage and there are several future-work items that require further consideration.

B. Future Work

An original intention was that larger blocks of code would be HW synthesised and – should there be updates over time – these updates could be relatively simply re-synthesised. As evidenced by the number of LEs required to implement a seemingly trivial concatenation of 5 AES functions, this would appear to be impractical. Further work is required here, beginning with the determination of an HLS-relevant key performance parameter to augment McCabe's complexity measure.

We have shown that we can achieve a usefully optimized solution for a simple, yet commonplace, cryptographic algorithm in a small number of iterations. It would be of substantial help to industrial implementers to reference a catalogue of synthesized code to better assess the risk and investment costs in implementing an FPGA-bound cryptographic function.

We have focused on speeding-up the establishment of a secure connection between controller and device using TLS. In the operational lifetime it is expected that far more time is spent maintaining authenticated traffic. An implementation process where synthesized algorithms are inserted into a frame/data stream to speed up this task should be examined.

A common problem with early implementations of secure elements was that the communication between processor and secure element was in-itself not secure. This exposure of interfaces is present in FPGAs attached to external processors and requires detailed consideration.

REFERENCES

[1] M. Naedele, 'Addressing IT Security for Critical Control Systems', in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, Jan. 2007, pp. 115–115. doi: 10.1109/HICSS.2007.48.

[2] A. Paul, F. Schuster, and H. König, 'Towards the Protection of Industrial Control Systems - Conclusions of a Vulnerability Analysis of Profinet IO', in *DIMVA*, 2013. doi: 10.1007/978-3-642-39235-1_10.

[3] R. Langner, 'Stuxnet: Dissecting a Cyberwarfare Weapon', *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, May 2011, doi: 10.1109/MSP.2011.67.

[4] T. M. Chen and S. Abu-Nimeh, 'Lessons from Stuxnet', *Computer*, vol. 44, no. 4, pp. 91–93, Apr. 2011, doi: 10.1109/MC.2011.115.

[5] D. Gunzinger, C. Kuenzle, A. Schwarz, H. D. Doran, and K. Weber, 'Optimising PROFINET IRT for fast cycle times: A proof of concept', in *2010 IEEE International Workshop on Factory Communication Systems Proceedings*, May 2010, pp. 35–42. doi: 10.1109/WFCS.2010.5548637.

[6] 'Mbed TLS', *Linaro*. https://www.trustedfirmware.org/projects/mbed-tls/ (accessed Jul. 02, 2022).

[7] T. Schläpfer and A. Rüst, 'Security on IoT devices with secure elements', presented at the Embedded World Conference, Nuremberg, Germany, 26-28 Februar 2019, WEKA, 2019. doi: 10.21256/zhaw-3350.

[8] M. Noseda, L. Zimmerli, T. Schläpfer, and A. Rüst, 'Performance analysis of secure elements for IoT', *IoT*, vol. 3, no. 1, pp. 1–28, 2021, doi: 10.3390/iot3010001.

[9] ARM, 'CPU Architecture Security Features', *Arm | The Architecture for the Digital World*. https://www.arm.com/architecture/security-features (accessed May 11, 2023).

[10] 'Network Security', *Xilinx*. https://www.xilinx.com/applications/wired-wireless/network-security.html (accessed May 11, 2023).

[11] 'XIP3022B: SHA256 AND SHA224 | Xiphera'. https://xiphera.com/products/XIP3022B.php (accessed Jul. 02, 2022).

[12] S. Li, J. Torresen, and O. Soraasen, 'Exploiting reconfigurable hardware for network security', in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, Apr. 2003, pp. 292–293. doi: 10.1109/FPGA.2003.1227276.

[13] A. Silitonga, F. Schade, G. Jiang, and J. Becker, 'HLS-Based Performance and Resource Optimization of Cryptographic Modules', in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Dec. 2018, pp. 1009–1016. doi: 10.1109/BDCloud.2018.00147.

[14] T. Takaki, Y. Li, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Sugawara, 'An Optimized Implementation of AES-GCM for FPGA Acceleration Using High-Level Synthesis', in *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, Oct. 2020, pp. 176–180. doi: 10.1109/GCCE50665.2020.9291973.

[15] R. S. Meurer, T. R. Mück, and A. A. Fröhlich, 'An Implementation of the AES Cipher Using HLS', in *2013 III Brazilian Symposium on Computing Systems Engineering*, Dec. 2013, pp. 113–118. doi: 10.1109/SBESC.2013.36.

[16] E. Ozcan and A. Aysu, 'High-Level Synthesis of Number-Theoretic Transform: A Case Study for Future Cryptosystems', *IEEE Embedded Systems Letters*, vol. 12, no. 4, pp. 133–136, Dec. 2020, doi: 10.1109/LES.2019.2960457.

[17] M. Skuballa, A. Walz, H. Bühler, and A. Sikora, 'Cryptographic Protection of Cyclic Real-Time Communication in Ethernet-Based Fieldbuses: How Much Hardware is Required?', in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2021, pp. 1–7. doi: 10.1109/ETFA45728.2021.9613244.

[18] R. Ernst, 'Codesign of Embedded Systems: Status and Trends', in *Readings in Hardware/Software Co-Design*, G. De Micheli, R. Ernst, and W. Wolf, Eds., in Systems on Silicon. San Francisco: Morgan Kaufmann, 2002, pp. 45–54. doi: 10.1016/B978-155860702-6/50006-5.

[19] G. Hu, S. Ren, and X. Wang, 'A Comparison of C/C++-based Software/Hardware Co-design Description Languages', in *2008 The 9th International Conference for Young Computer Scientists*, Nov. 2008, pp. 1030–1034. doi: 10.1109/ICYCS.2008.204.

[20] M. Chiodo, P. Giusto, A. Jurecska, H. C. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno, 'Hardware-software codesign of embedded systems', *IEEE Micro*, vol. 14, no. 4, pp. 26–36, Aug. 1994, doi: 10.1109/40.296155.

[21] Intel, 'Cyclone V Device Overview'.

[22] 'High-Level Synthesis Compiler - Intel® HLS Compiler', *Intel*. https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html (accessed Jul. 02, 2022).

[23] T. J. McCabe, 'A Complexity Measure', *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.

[24] A. Walz and A. Sikora, 'Exploiting Dissent: Towards Fuzzing-Based Differential Black-Box Testing of TLS Implementations', *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 2, pp. 278–291, Mar. 2020, doi: 10.1109/TDSC.2017.2763947.