CHERIoT: Rethinking Security for Low-Cost Embedded Systems

Saar Amar, Tony Chen, David Chisnall, Felix Domke, Nathaniel Filardo, Kunyan Liu, Robert Norton-Wright, Yucong Tao, Robert N. M. Watson, Hongyan Xia – Microsoft
Presented by Jonathan Woodruff – University of Cambridge

Abstract

Small embedded cores have little area, power, or performance budget to spare for security features and yet must often run code written in unsafe languages and, increasingly, are exposed to the hostile Internet. CHERIoT (Capability Hardware Extension to RISC-V for Internet of Things) builds on top of CHERI and RISC-V[1] to provide an ISA and software model that lets software depend on object-granularity spatial memory safety, deterministic use-after-free protection, and lightweight compartmentalization exposed directly to the C/C++ language model[2,3]. This can run existing embedded software components on a clean-slate RTOS that scales up to large numbers of isolated (yet securely communicating) compartments, even on systems with under 256 KiB of SRAM.

CHERIoT-Ibex Implementation

CHERIoT-Ibex is an open-source RTL implementation of CHERI ISA based on LowRISC's Ibex core[4](*https://github.com/microsoft/cheriot-ibex*). CHERIoT-Ibex is a 32-bit RISC-V microcontroller which implements the CHERIoT ISA extension in addition to RV32IMCB. As with the original ibex core, the design can be configured with either a 2-stage or a 3-stage pipeline.

CHERI-Ibex supports all 30 instructions listed in the CHERIoT ISA specification, including those to query and test capabilities, load/store capabilities from memory, control program flow and access special capability registers. Certain compressed instructions are also extended for capabilities, including c.incaddr, c.jal, c.jalr. The RV64 c.ld and c.sd instructions are repurposed for c.clc and c.csc instructions.

The CHERIoT-Ibex implementation extends a configurable number of the general purpose registers into CHERI capabilities. CHERIoT-Ibex extends its data bus to 33-bit, where the MSB 1-bit is used as a valid tag to differentiate between capabilities and data. The load-store unit also supports atomic capability load and store transactions. CHERIoT-Ibex also adds Special Capability Registers (SCRs) including MTCC (replacing mtvec) and MEPCC (replacing mepc).

CHERIoT-Ibex performs capability-based memory access rule checking including:

- data load/store accesses
- capability load/store accesses
- Instruction fetch (PCC-based)
- jump target calculation (cjal and cjalr)

New capability exceptions are generated in the case of access rule violations.

Temporal memory safety support

The CHERIoT load-capability instruction provides a sophisticated filter to accelerate temporal memory safety. When loading a valid pointer to heap memory, the instruction checks a shadow memory with a bit for each heap granule (8 bytes currently). The tag bit of the loaded capability is cleared if the shadow bit of memory it points to is 1. These bits are set by the RTOS's heap allocator on deallocation to ensure that stale capabilities are not propagated in the system before a memory sweep can remove them from memory in preparation for reuse.

Backward compatibility

CHERIoT-Ibex provides a backward-compatibility mode that can be set on startup. In this mode, the CHERIoT instructions can still execute, however access rules are disabled and any binary code generated by non-CHERI RV32 compilers can run unmodified in CHERIoT-Ibex.

Timing and area

CHERI-ibex (with 3-stage pipeline) has been synthesized at 330MHz using TSMC 28nm HPC+ libraries (HVT only) and > 1GHz using TSMC n5 libraries (SVT only). The design size is ~70k gate equivalents. A detailed PPA analysis is under way at Microsoft.

CHERIoT Sail: Formal Specification

Our team has adapted the CHERI-RISC-V Sail executable formal model to describe CHERIoT (*https://github.com/microsoft/cheriot-sail*). This executable description of the CHERIoT instruction set is the gold standard for the ISA. It simultaneously provides

human-readable documentation in our technical report, compiles into our reference simulator, and facilitates proof of ISA properties using formal analysis.

The Sail compiler can translate Sail code into a Satisfiability Modulo Theories (SMT) problem that can be given to a solver such as CVC4 or Z3 to check whether a given function returns true for all input values. We have used this feature to check important properties of the capability encoding as implemented in Sail, including the following representative examples:

`prop_andperms` checks that any capability and permissions mask CAndPerm will result in a capability whose permissions are a subset of the original permissions and the mask. This is non-trivial on CHERIoT due to its novel compression of the permissions field.

`prop_setbounds_monotonic` checks monotonicity preservation of setCapBounds, despite rounding.

`prop_repbounds` checks that the limits of representability as the address strays out of bounds match expectations from the encoding.

CHERIoT RTOS and software stack

The CHERIoT RTOS research platform has also been open-sourced to enable wider collaboration (*https://github.com/microsoft/cheriot-rtos*). The CHERIoT platform is designed around the principles of least privilege and intentionality. As such, the structure is somewhat different from a conventional OS or RTOS. Particularly central to the architecture are the notions of memory safety, compartments, and threads.

A system is said to be **memory safe** if its references to memory are:

- Unforgeable: A reference to memory (in particular, the authority to access memory) can be constructed only from other references.
- Monotonic: A constructed reference will have no more authority than its progenitor reference(s) (and may have less).
- Spatially Safe: References to memory authorize access to a set of memory locations determined when the reference is constructed.
- Temporally Safe: References to a region of memory will not remain usable across reuse of memory for a different allocation.

Compartments define spatial ownership. A compartment defines a set of code, globals, exported entry points, imported entry points from other compartments, and imported memory-mapped I/O (MMIO) regions. A compartment is treated as a single protection domain. While a compartment is running, it is assumed to be able to execute any of its code, modify any of its globals, access any of the MMIO regions that it has access to, or transfer execution to any of the compartments whose entry points it has imported.

Threads define temporal ownership. A thread defines a register state (either in the register file while running, or in a register save area when preempted), a stack, and a trusted stack. Threads are always executing in one compartment and that compartment has access to a subset of the stack The trusted stack maintains a record of the compartments that a thread has suspended by performing a cross-compartment call and which will be resumed when those calls return. Each cross-compartment call shrinks the stack so that all state created by existing compartments is inaccessible to the new callers.

Conclusion

The CHERIoT project would not have been possible without the existing “big CHERI” research, exploration of green-field CHERI-aware operating systems, and work to adapt CHERI software models to embedded systems. This ISA attempts to scale CHERI down yet further. For an optimal result, we have simultaneously designed our ISA, compartment model, programmer model, compiler, and RTOS. The capability encoding and instruction set have also been tuned to this use-case and validated by running existing embedded software in compartments. We hope that the CHERIoT project will motivate and inform the standardization of a CHERI extension for RISC-V in embedded systems.

---

[1] Watson, Robert NM, et al. Capability hardware enhanced RISC instructions: CHERI instruction-set architecture (version 7). No. UCAM-CL-TR-927. University of Cambridge, Computer Laboratory, 2019.

[2] Davis, Brooks, et al. "CheriABI: Enforcing valid pointer provenance and minimizing pointer privilege in the POSIX C run-time environment." ASPLOS. 2019.

[3] Joly, Nicolas, Saif ElSherei, and Saar Amar. "Security analysis of CHERI ISA." 2021.

[4] Lowrisc, IBEX documentation, Oct 2020, https://ibexcore.readthedocs.io/en/latest