Ti50: What we want from Tock

TL;DR

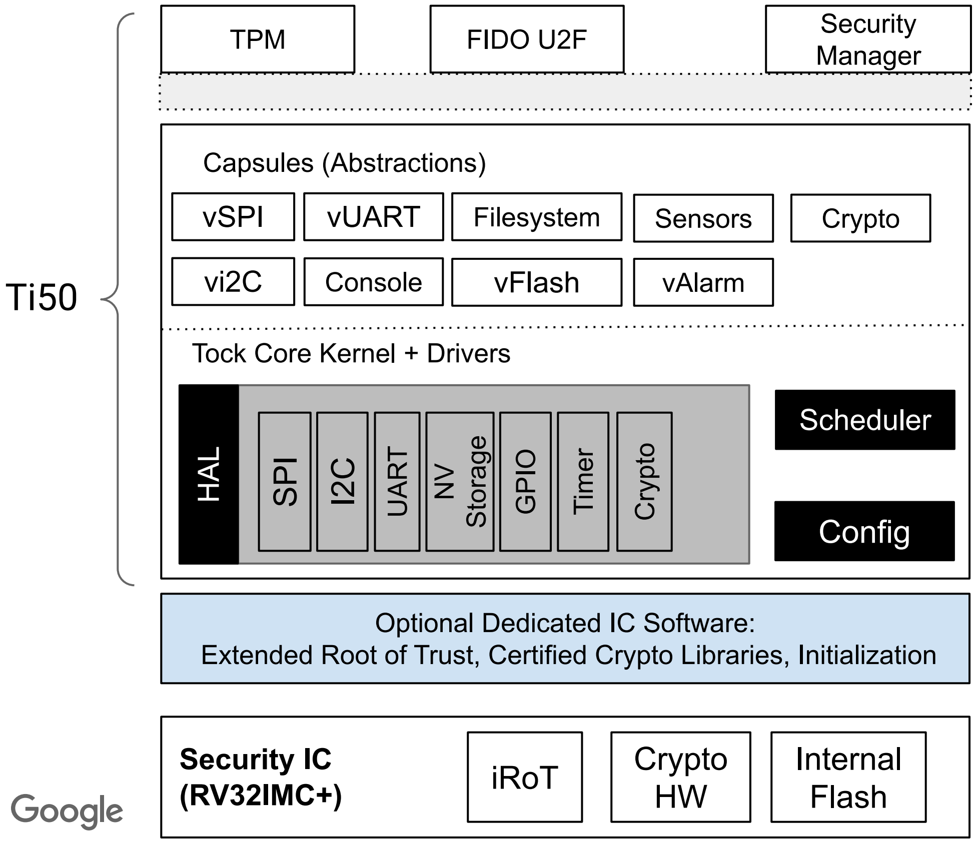
This is a broadcast of ideas

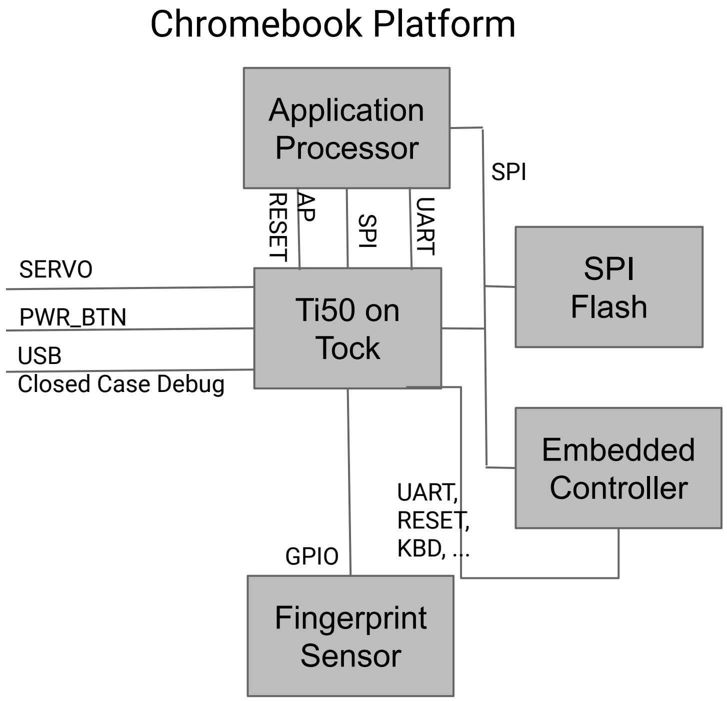
We are looking for collaboration / deduplication / inspiration of work

and early feedback

Sharing “why” we do some changes, what we want to accomplish

ChromeOS Use Case





ChromeOS Use Case

- System Manager / Platform Root Of Trust
    - enhanced, security hardened RISC-V chip
    - detection, mitigation and recovery of security issues
    - always ON, even when Application Processor is powered off -> power management
- Multiple applications
    - execution from flash (due to the code size)
    - max code reuse between several chip variants
- Secure & robust firmware upgrades
    - 2 flash banks - active(golden) copy and updateable + active data
- Management of platform secrets (TPM, U2F, OS login, etc)
    - hardened crypto API
    - confidentiality & integrity of system & apps persistent data

ChromeOS Use Case (2)

- Closed-case debug support
  - low latency UART multiplexing
- Shared interface to apps (SPI or I2C)
  - Dispatching of commands from AP among Ti50 apps based on command codes
- High availability - reboot cause platform reboot
  - Support for Watchdog timer, sleep & deep sleep modes
- Certifications (FIPS crypto, Common Criteria - TPM, U2F, etc)
  - Need to prove that isolation of applications is good enough. Example of requirements.
  - Testing, fuzzing, 100% branch-coverage, no dead code at source & binary
  - Traceability of security threats, security objectives, security functional requirements and functional tests -> tracking of requirements, artifacts
  - Reproducible builds
  - Independent testing, including vulnerability reward program

Multiple Applications

- extended threat model for our application - WIP
  - confidentiality & integrity of data, residual data leakage, covert channels, vulnerabilities, …
  - defense in depth (lite-ASLR, check for stack pivoting, no data execution, etc)
- code size & performance optimizations
  - shared libs among apps
  - static applications -> possible changes in Process::Create
  - efficient IPC, ideally close to zero-copy
  - syscall performance (next slide)
- isolation of resources
  - ACLs on syscalls / devices / capsules
  - encrypted files system with ACLs
  - crypto key management
  - application reset on panic

Performance optimizations

- low latency requirement for interrupt processing
  - transfer data from one UART to another while monitoring for control sequences
- syscall penalty reduction
  - expect many syscalls for crypto & I/O
  - home-grown OS has 50 cycles penalty, Tock ~5172* cycles
  - synchronous syscalls (don't subscribe just to always yield) -> remove 2 syscalls
  - enable direct use of constants from .rodata -> remove some allow() syscalls
  - different syscall conventions (a0 -> t6 to minimize register shuffling), pass more regs, stay with just 'command' syscall, make 'allow' as part of 'command' syscall
- IPC using shared-memory
  - dispatch commands/responses up to 4K among apps
  - AppID as u32 or u8
- 64-bit timers, avoid long division in timer by changing timer frequency

* - OpenTitan team reduced it to ~450

Enhanced RISC-V core (Google internal)

- Integrated Root of Trust
  - code signing required
- Certified crypto libraries
  - Use API to perform operations vs. direct HW access
- 16 PMP regions
- Power management, Deep sleep support
- Security alerts
- Additional protection mechanism extending PMPs
- New CSRs, and instructions (subset of bitmanip)
  - Modified toolchain to support

Crypto libraries

- API for key generation and management
  - use key handles for apps
  - export keys only using key-wrapping, blob for apps
  - access control to keys on per application basis
  - side-loading of keys for hardware-bound keys
  - zeroization of keys
  - board-specific flash region with restricted access
- symmetric ciphers, different modes (-OFB, -GMAC, -KWP, -CTR, etc)
- public-key crypto (RSA 4K, ECDSA P-256/P-384, ECDAA(?), etc)
- parallel context support, sharing hardware resources
- FIPS 140-3 compliance (health tests, known-answer tests, etc)
- post-quantum crypto, firmware signature verification
- HW-accelerators (AES, HMAC, DRBG, Big num)
  - via certified crypto lib primarily for ChromeOS

Filesystem

- efficient use of shared flash space
- device-bound, application-bound encryption
- integrity protection (AEAD, etc)
- flash brown-out resistance (incomplete writes/erase due to power-off)
- ACLs for objects
- transaction support (detect incomplete transactions)
- flash wear minimization
- performance considerations:
  - minimize erase count
  - flash bank aware (avoid updating the one with active firmware)

Host emulation

- multiple targets for same code
    - target security IC
    - verilator
    - QEMU (with device emulation at register level)
    - host (device emulation at register level)
- device emulation at register level
    - hooks to tock::register
    - use mostly same driver code as on target for coverage
- host execution model
    - maximize code reuse from target, not just emulated syscalls
    - emulate context switching
    - interrupts from devices
    - syscall handling

Addressing:

- **Unit testing for drivers**
- **CQ testing, including full product ChromeOS + TI50**
- **Development velocity**
- **100% branch coverage**

Testing

- automated unit tests
- automated integration tests (single & multi-app)
    - reuse same test framework among apps and core
    - all levels and targets
- branch coverage (on target and host emulation)
    - no dead code requirement for source and compiled code
    - some stuff require emulation (security alerts, I/O errors, etc)
- fuzzing
- HWASAN (software memory tagging)
    - apps can be in C, unsafe code in crypto libs, etc
    - need toolchain enabling for RISC-V support
    - OS-specific libraries for Tock

Toolchain enhancements

- support new instructions (wip)
- build / link multiple Tock apps
- code size optimization
    - support for linker relaxation
    - support for tp- relative addressing for apps vs. gp- used for kernel (?)
    - -Oz general improvements
- on-target code coverage for embedded
    - replace 64-bit counters with 32-bit or 8-bit flags to save data & code
    - download coverage data from target
- HWASAN for RISC-V
- toolchain stabilization