SECWORKS

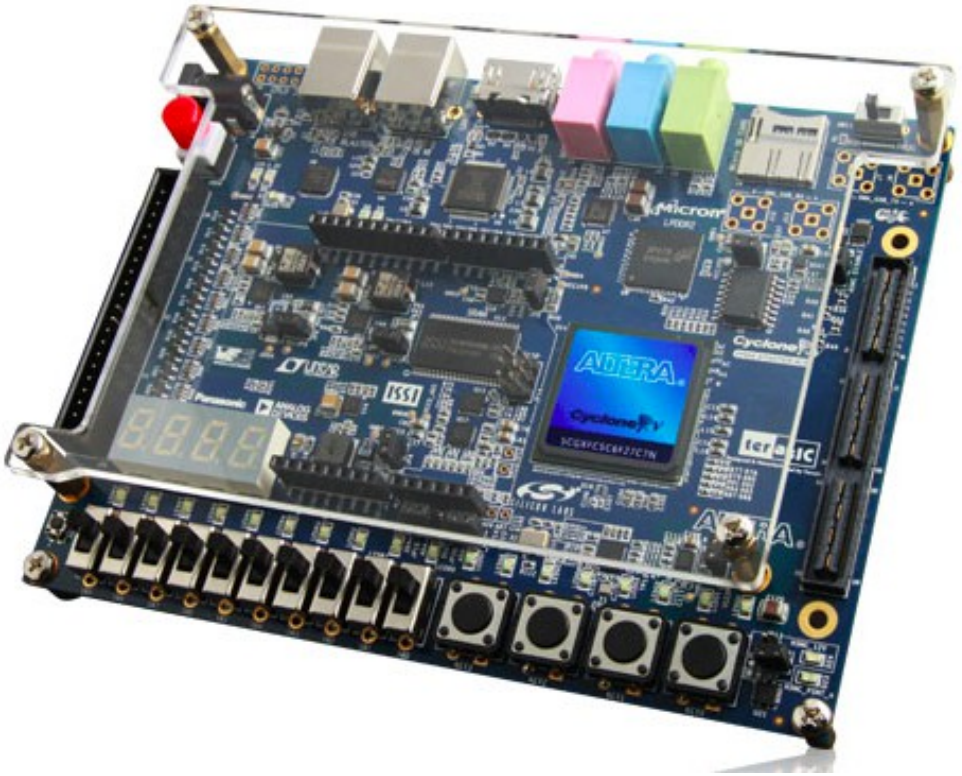
CRYPTECH

Cryptech
HW development status 2014-03-10

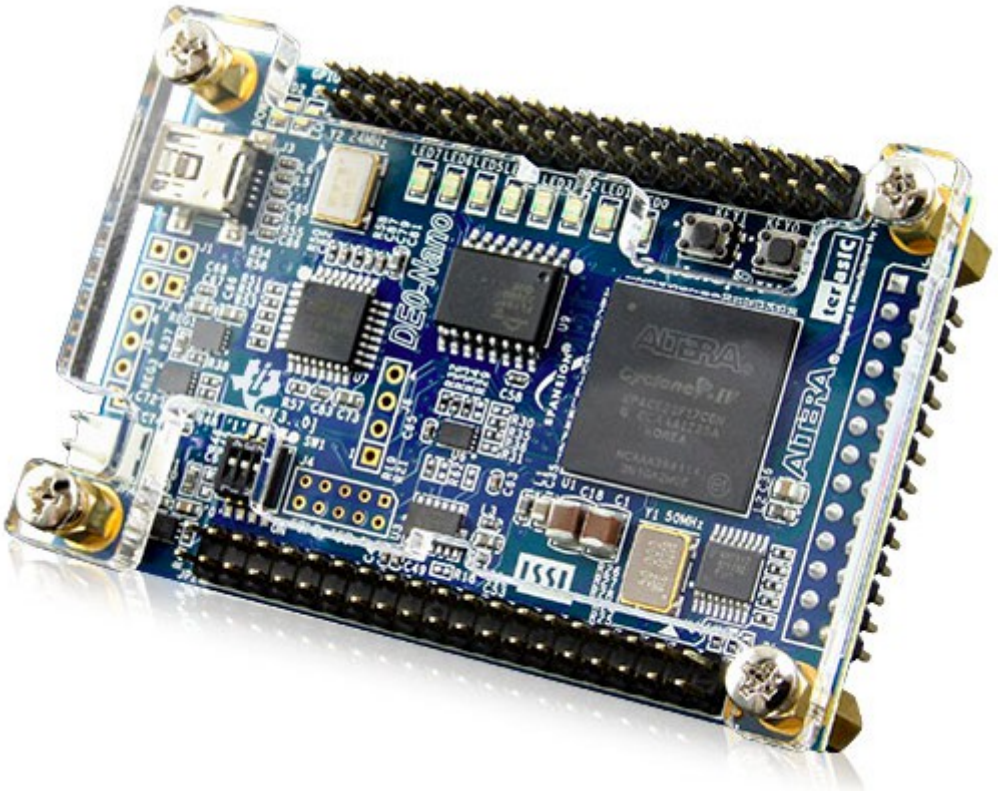
Joachim Strömbersgon – Secworks AB

SECWORKS

CRYPTECH



Full system capacity
Arduino headers
Integrated RS232-USB
High speed I/O



Core, subsystem capacity
USB or battery powered

Cores and status (1)

- SHA-256 HW-verification in progress
- SHA-1 HW-verification in progress
- SHA-512/x RTL nearly completed
- AES-128 Started
- Ed25519 Very early phase RTL
- TRNG Implementation Proposal started

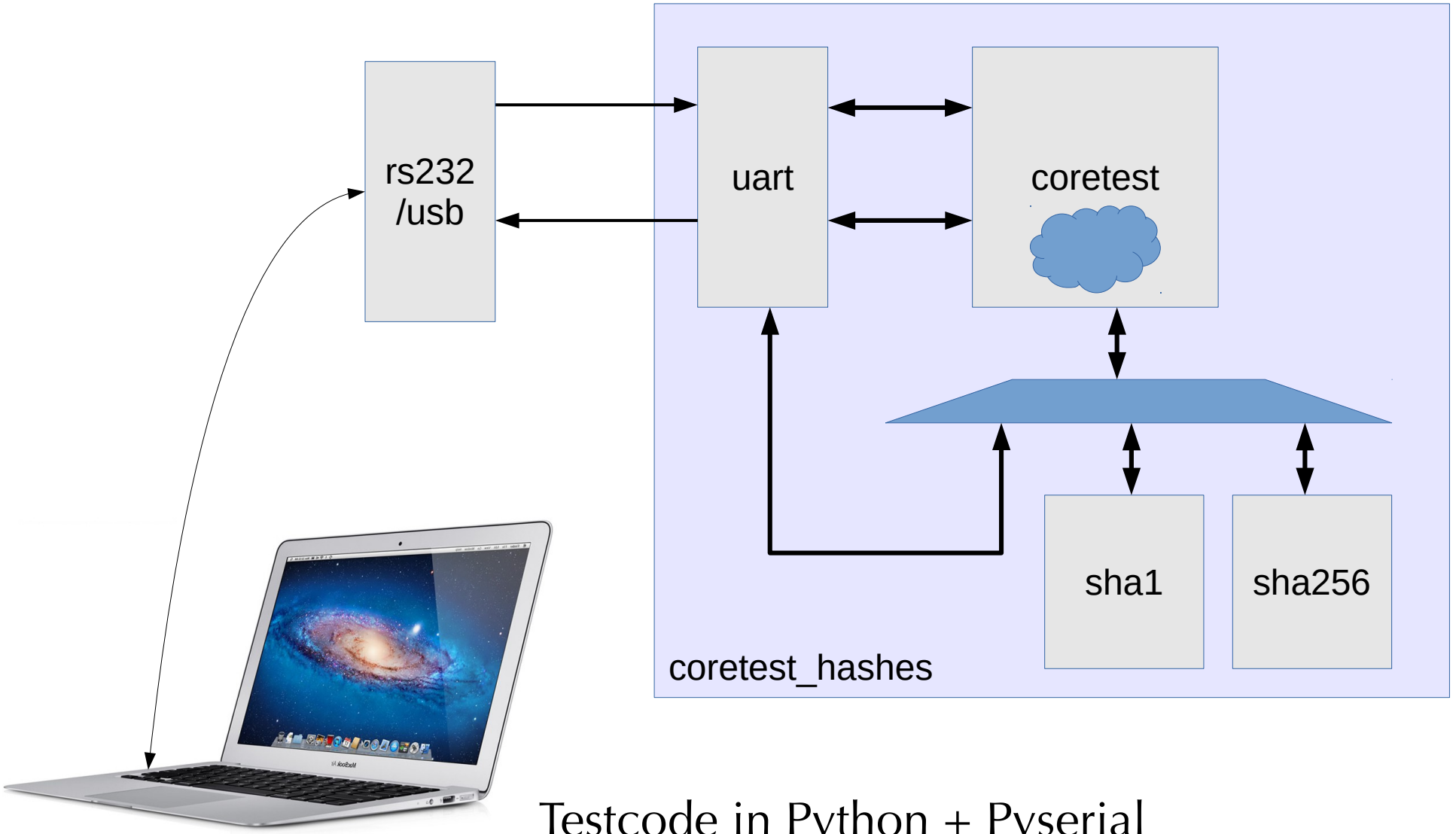
Cores and status (2)

- uart HW-verification in progress
- Coretest HW-verification in progress
- test_core HW-verification in progress

Testbenches, Python models,
SW-testcode, RTL-generators,
some docs etc

Coretest_hashes running in FPGA

2856 ALMs (10% of FPGA in C5G), 3688 regs, 80 MHz fmax in C5G

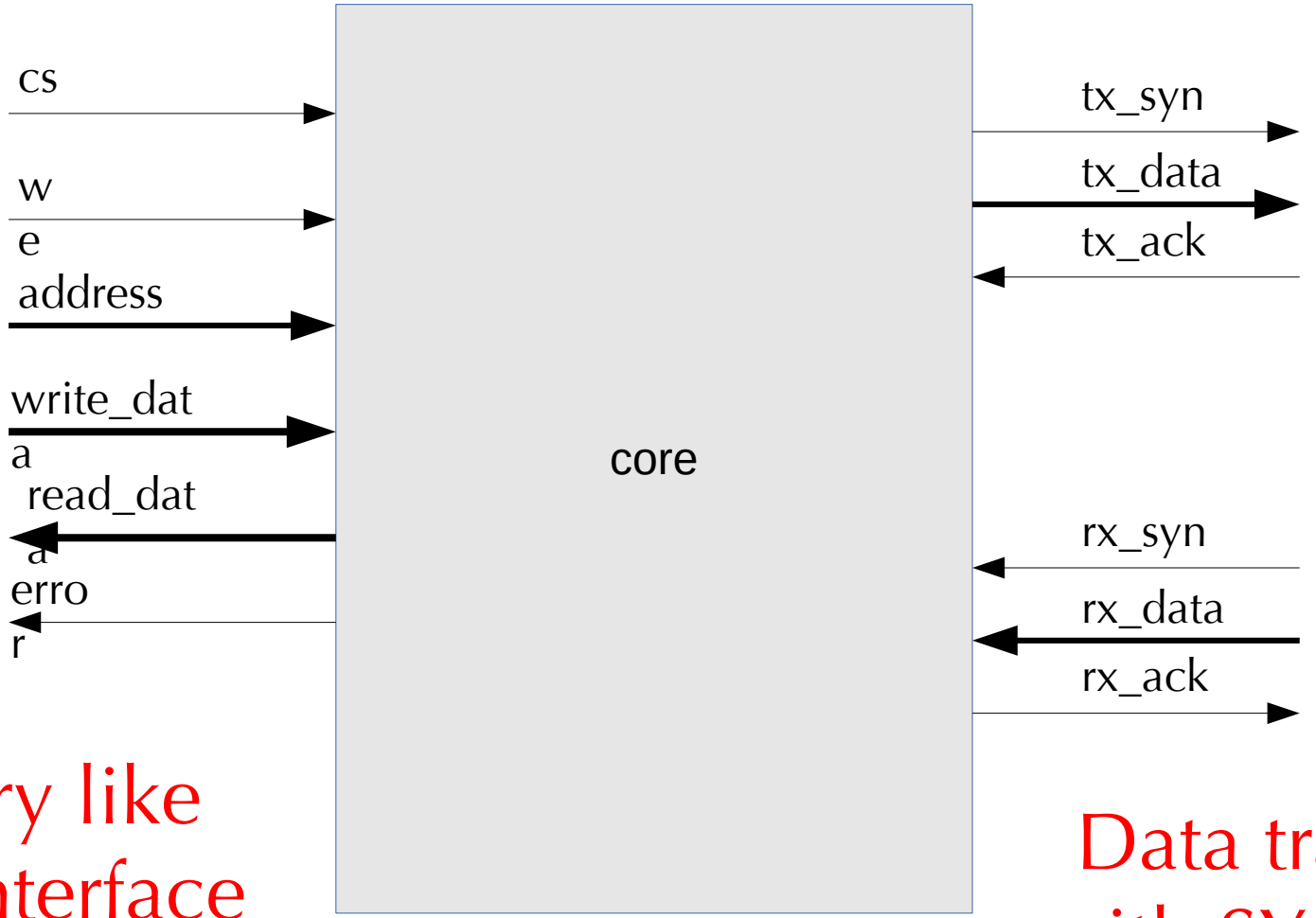


Testcode in Python + Pyserial

Coretest

- On-chip test runner
  - Connected to host via byte interface
  - Conncted to one or more cores to be tested
- Command parser and response generator
  - Read out 32-bit word from address
  - Write 32-bit word to address
  - Reset core under test
  - Sends respons to all commands

Can be a basis for a real on-chip controller/master



Memory like
32-bit interface

Data transfer
with SYN-ACK
(If possible)

FPGA tools and methodology (1)

- Discovery of available, research, open tools
  - Growing collection of projects
  - Few that looks very promising
- Interview with expert (Tryggve Mathiesen)
  - Support from vendors regarding openness
  - Methods for creating trustworthy FPGA HW
  - Workarounds

FPGA tools and methodology (2)

- All backend done using FPGA vendor tools
  - No third party bitstream generators
- Reverse engineering of bitstream breaks license
  - Can not use FPGA vendor tools (which we must)
- Strategies for checking the design and result
  - *Double compilation* – different FPGA vendors
  - Designs with full external access

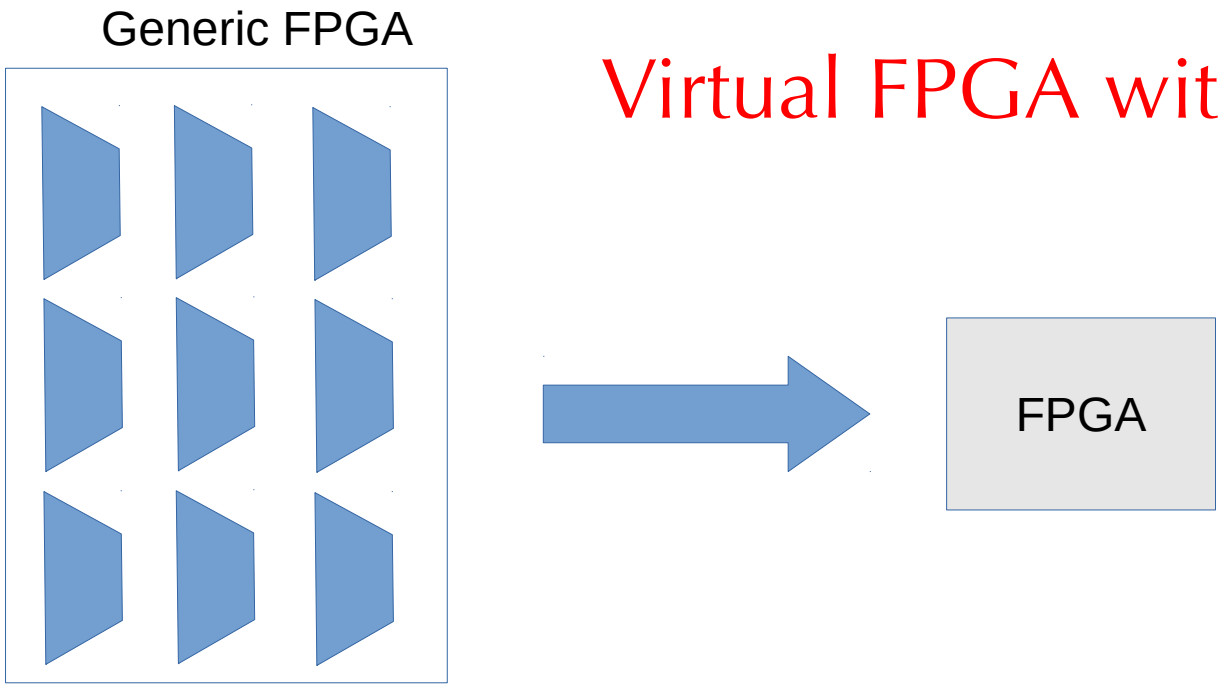
FPGA tools and methodology (3)

- Strategies for checking the design and result
  - Code and netlist review, checking
  - Custom tools. Some EC tools
    - Hard problem: Multiple mappings RTL--> FPGA reso

FPGA workaround strategies (1)

- Virtualization within the FPGA
  - Generic FPGA design implemented in FPGA
    - Load the actual functionality during runtime
    - Requires custom RTL-netlist tools
    - Bad optimization
    - Even more HW overhead
    - Risk of manipulation

Virtual FPGA within FPGA



FPGA workaround strategies (2)

- Virtualization
  - Constrained CPU, Crypto CPU implemented in
    - Load the SW during runtime
    - Bigger risk for manipulation – It is SW (again).
    - Side channels?

FPGA workaround strategies (3)

- Compartmentalization
  - Multiple FPGAs, discrete components
  - Easier to verify
  - Bigger, more complex and slower

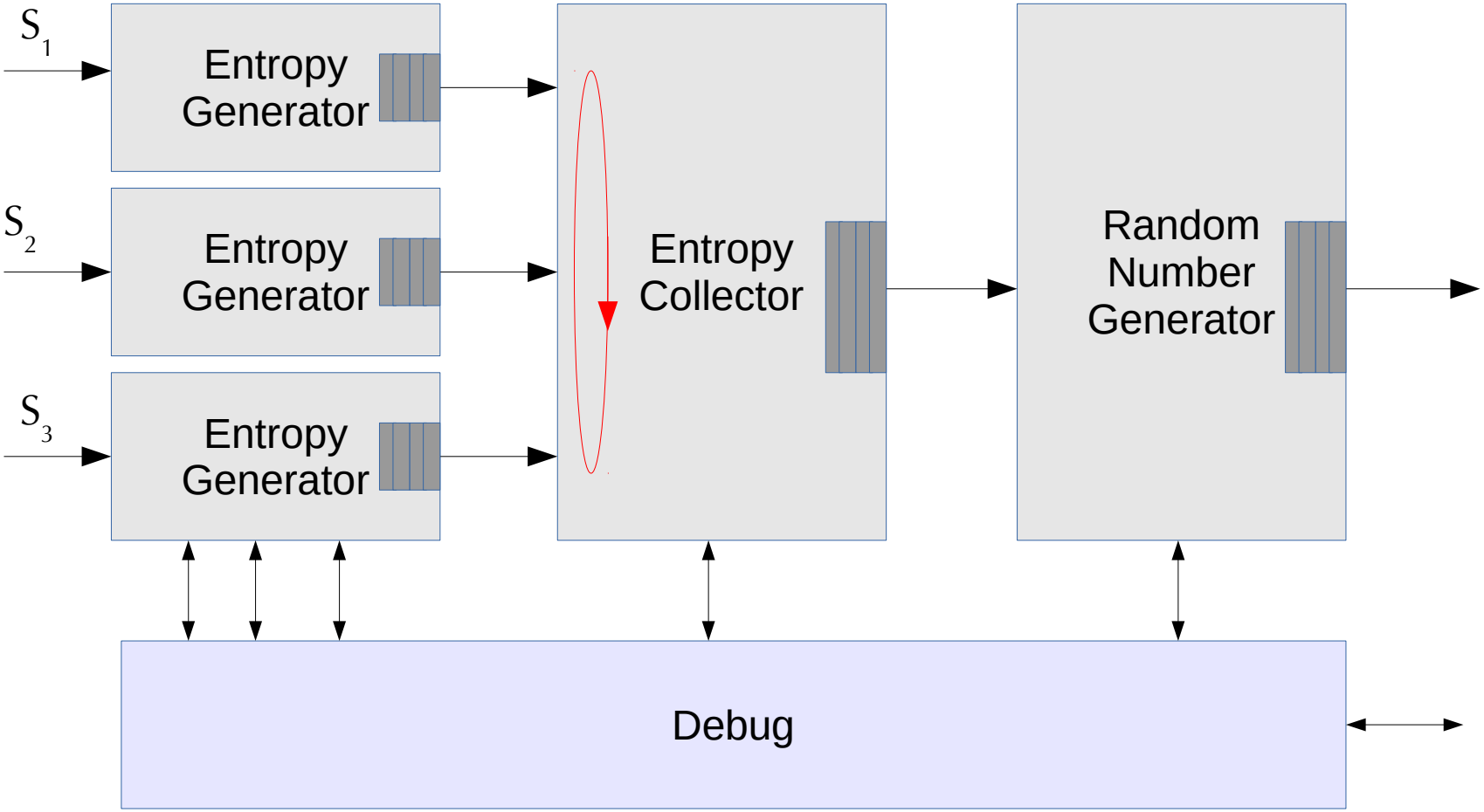
TRNG Ideas

This section moved to a separate presentation.
See Cryptech_TRNG_Ideas_2014-03-17.pdf

TRNG (1)

- Development drivers
  - Good performance
  - Conservative design
    - No wild, untested ideas and algorithms
- Support tranparency and testability
  - Debug access to raw entropy. Debug access to CSPRNG with injection
  - All normal access blocked when operating in debug mode
  - Full restart including warm-up when leaving debug mode
- Configurable
  - Security (number of rounds)
  - Number of entropy sources
  - Amount of entropy/seed
  - Amount of Random values/seed

TRNG


S1
S2
S3
Entropy Generator
Entropy Generator
Entropy Generator
Entropy Collector
Random Number Generator
Debug

Entropy generation (1)

- Two or more independent generators
- Each generator connected to their own physical entropy source
  - P/N reverse bias avalanche noise
  - CCD black out noise
  - RSSI
  - Oscillator jitter

Entropy generation (2)

- Each generator collects entropy.
- Support for on-line tests.
  - Detect broken entropy source

- Provide raw entropy in debug mode
- Perform conditioning, whitening on entropy
- Provide entropy as 32-bit data to accumulator

Entropy accumulation (1)

- Based on SHA-512 (FIPS 180-4)
- Collects and combines words from generators
  - Round robin access
  - Overlapping XOR with 1/n stride where n is number of generators
- After 1024 bits collected the entropy is inserted into SHA-512 as a message block
- After at least 256 message blocks digest is extracted as seed to random number generator
  - If seed pool is full we simply continue to accumulate
- SHA-512 is then restarted on a new 'message'

Random number generation (1)

- Based on XChaCha with 256 bit key
  - Scalable security (number of rounds)
  - Scalable performance
  - Not AES, but pretty well proven. Good traction

AES-CTR would be the alternative

- 96 bit IV and 32 bit counter
  - (2**32 – 1) keystream blocks between reeseding
  - 255 Gbyte max
- At least 24 rounds. 32 possible with good performance

Seed used for key and IV

Random number generation (2)

- Debug support for testing
  - Inject user defined seed
  - Generate at most n blocks of random values then stops
  - Access to generated values via debug port only
  - Reading from normal port during debug yields is a read error.
  - Full restart including warm up when leaving debug mode
- Generates random values continiously
  - Overwrites blocks in output buffer unless being used