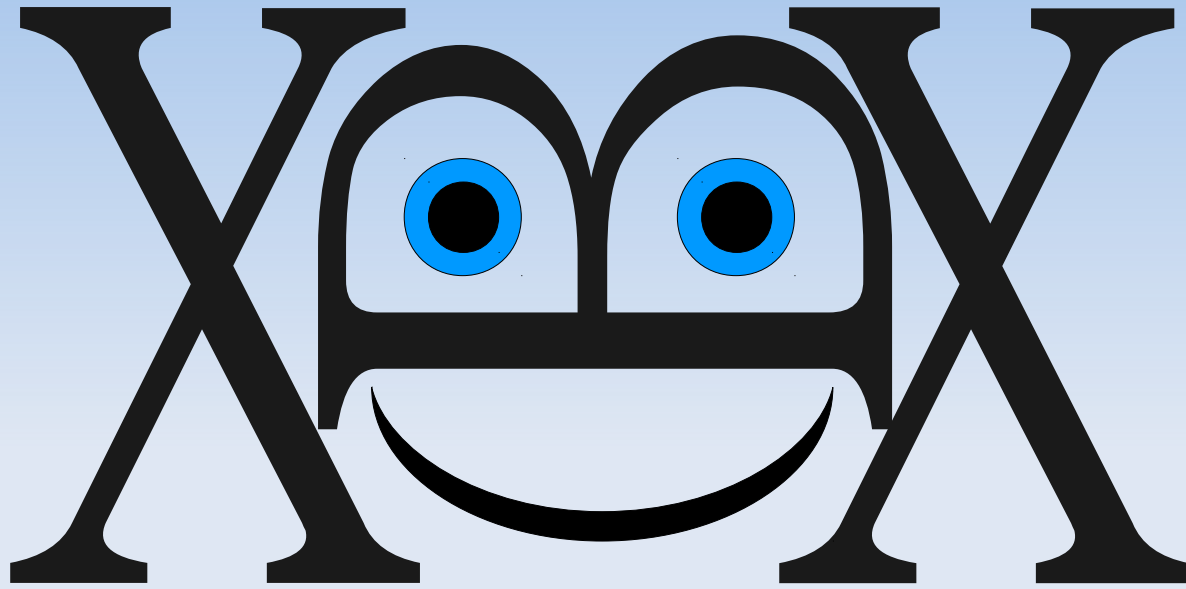
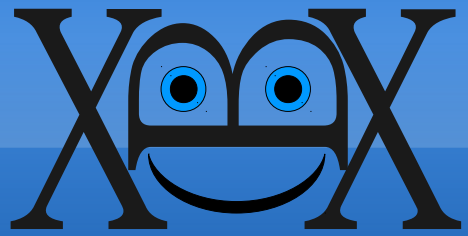


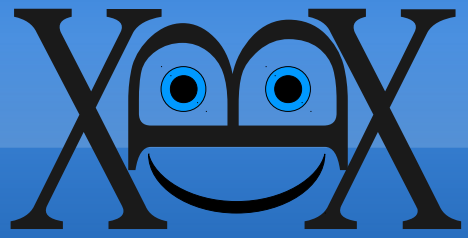
eXternal Benchmarking eXtension for the SUPERCOP crypto benchmarking framework

CHES 2010, August 17-20, Santa Barbara, UCSB
Christian Wenzel-Benner, ITK Engineering AG
Jens Gräf, LiNetCo GmbH

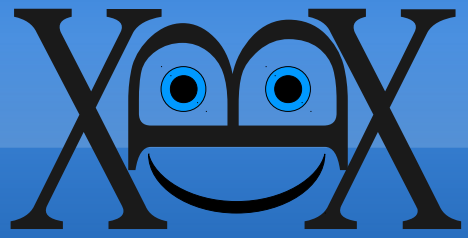


Slides Overview

- Introduction
  - Motivation
  - Design Goals
- System Overview
  - Hardware
  - Software
- Reviewer Comments
- Example Benchmark Results
- Present & Future

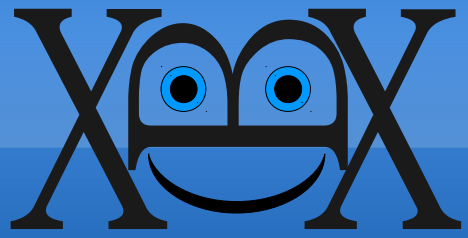


Introduction



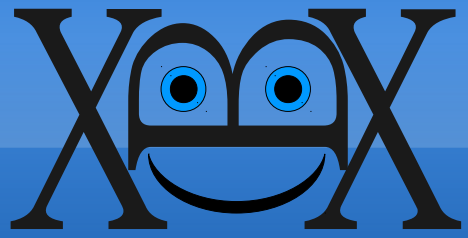
Motivation

- Big demand for benchmarking in crypto
  - For speed, and for embedded applications also size
- Reproducibility is very important
  - Compiler versions and flags must be logged
  - Benchmarking method must be well specified
  - Setup should be cheap so others can replicate it
- SUPERCOP addresses most of the above
  - XBX adds benchmarking for size
  - XBX allows to benchmark small devices



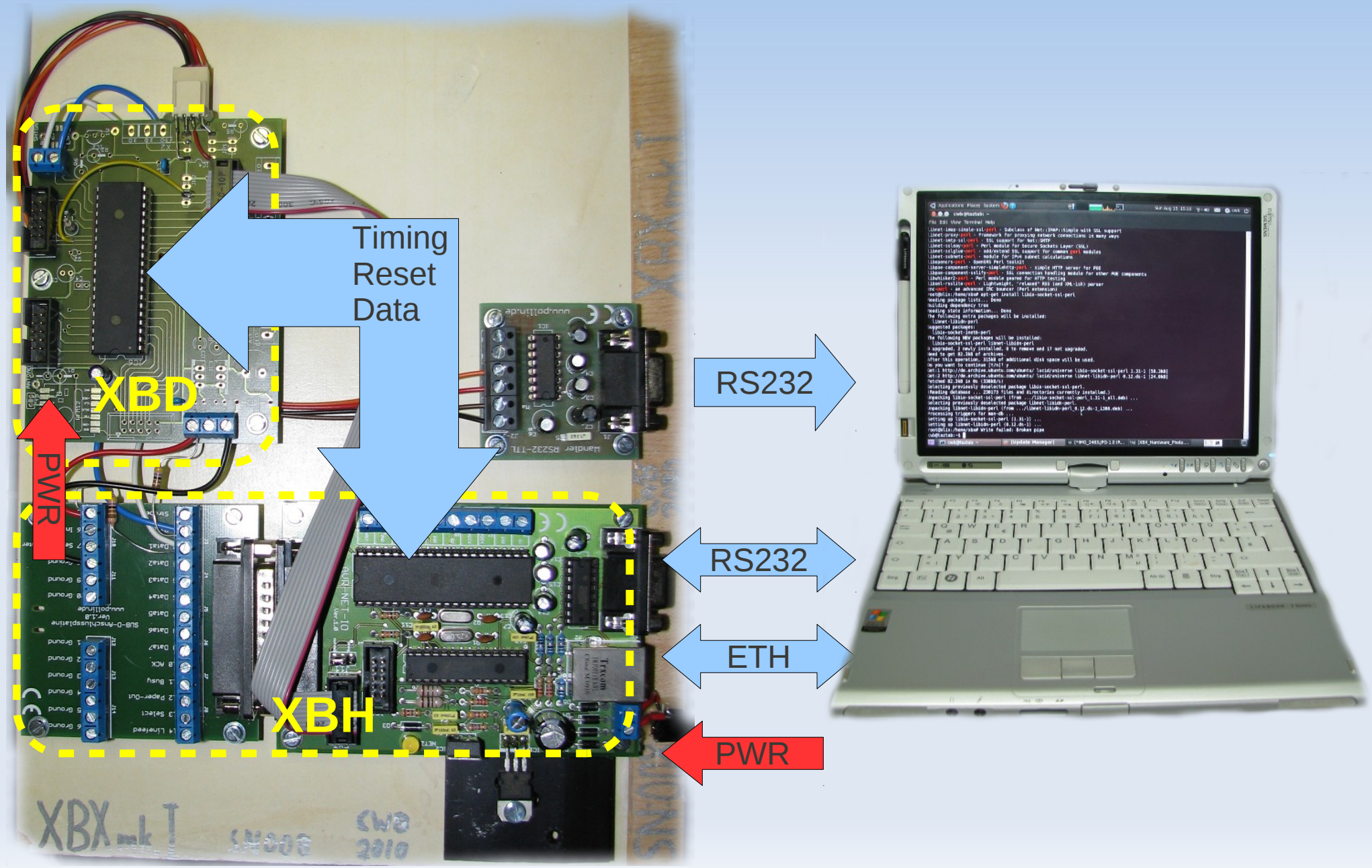
Design Goals

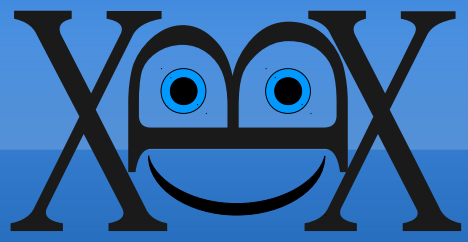
1. Automatic testing by scripts
2. Precise, real world performance numbers
3. Free source code for others to inspect
4. Cheap, easily available hardware
5. SUPERCOP input compatible
6. SUPERCOP output compatible
7. Development with pre-owned and/or free tools
8. Heavy component re-use
9. Focus on SUPERCOP-eBASH



System Overview

Hardware

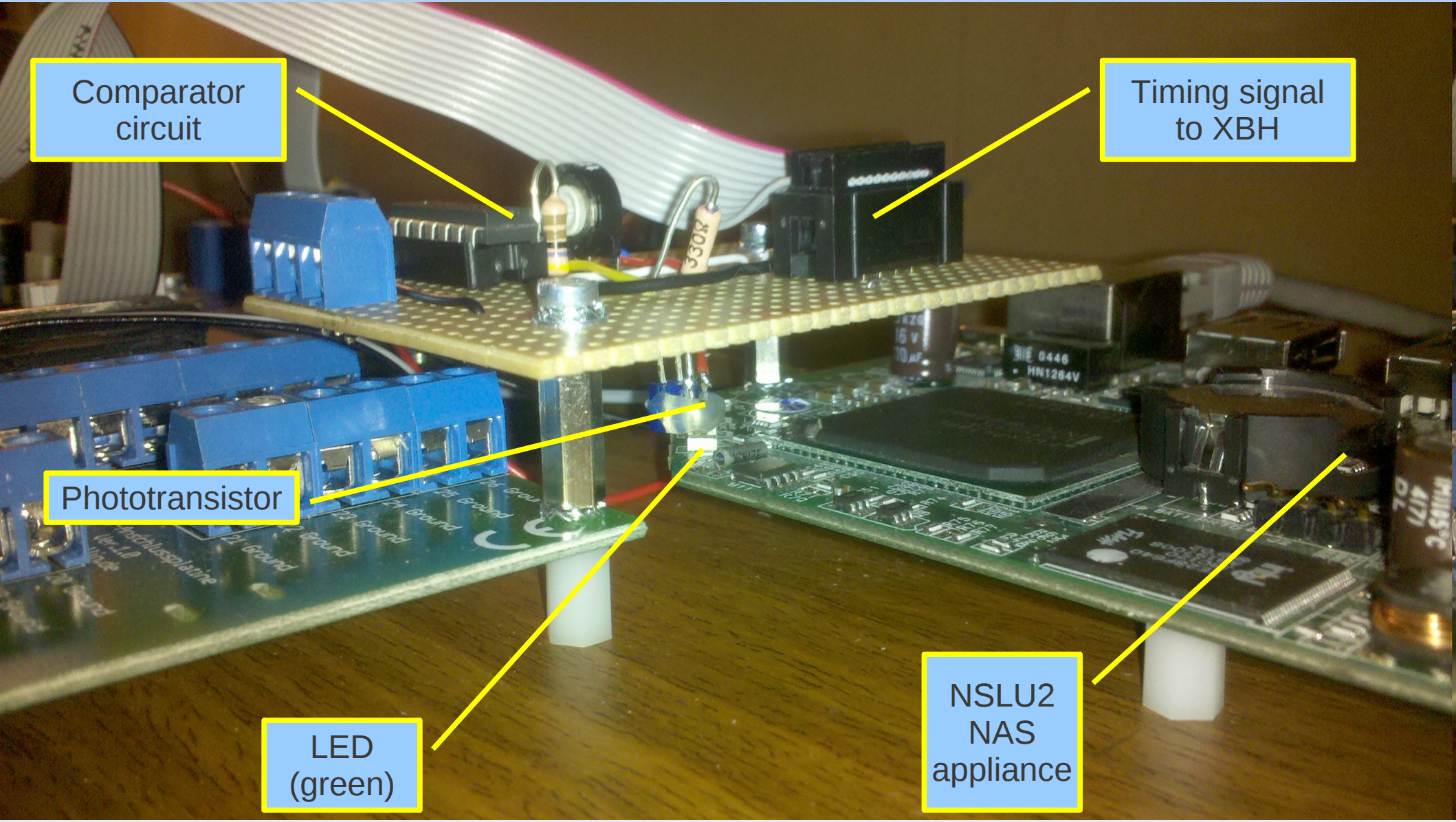


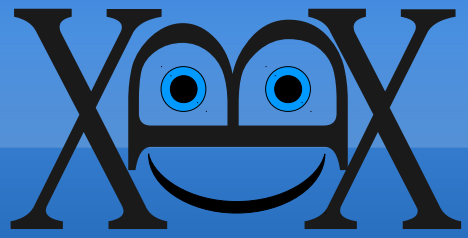


Hardware

- Personal Computer
  - Needs Ethernet, RS232 recommended
- XBH, eXternal Benchmarking Harness
  - Ethernet (TCP) to PC
  - RS232 to PC for configuration and debug
  - Digital I/O Pins to XBD (Timing, Reset)
  - Data to/from XBD: I²C, UART or Ethernet (UDP)
- XBD, eXternal Benchmarking Device
  - Digital I/O Pins to XBH (Timing, Reset)
  - Data to/from XBH: I²C, UART or Ethernet (UDP)

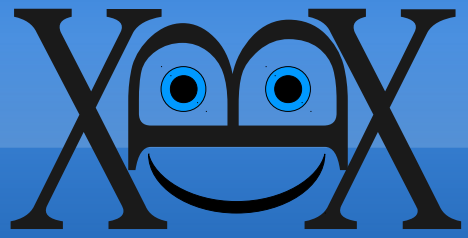
Hardware


Comparator circuit
Timing signal to XBH
Phototransistor
LED (green)
NSLU2 NAS appliance



Software

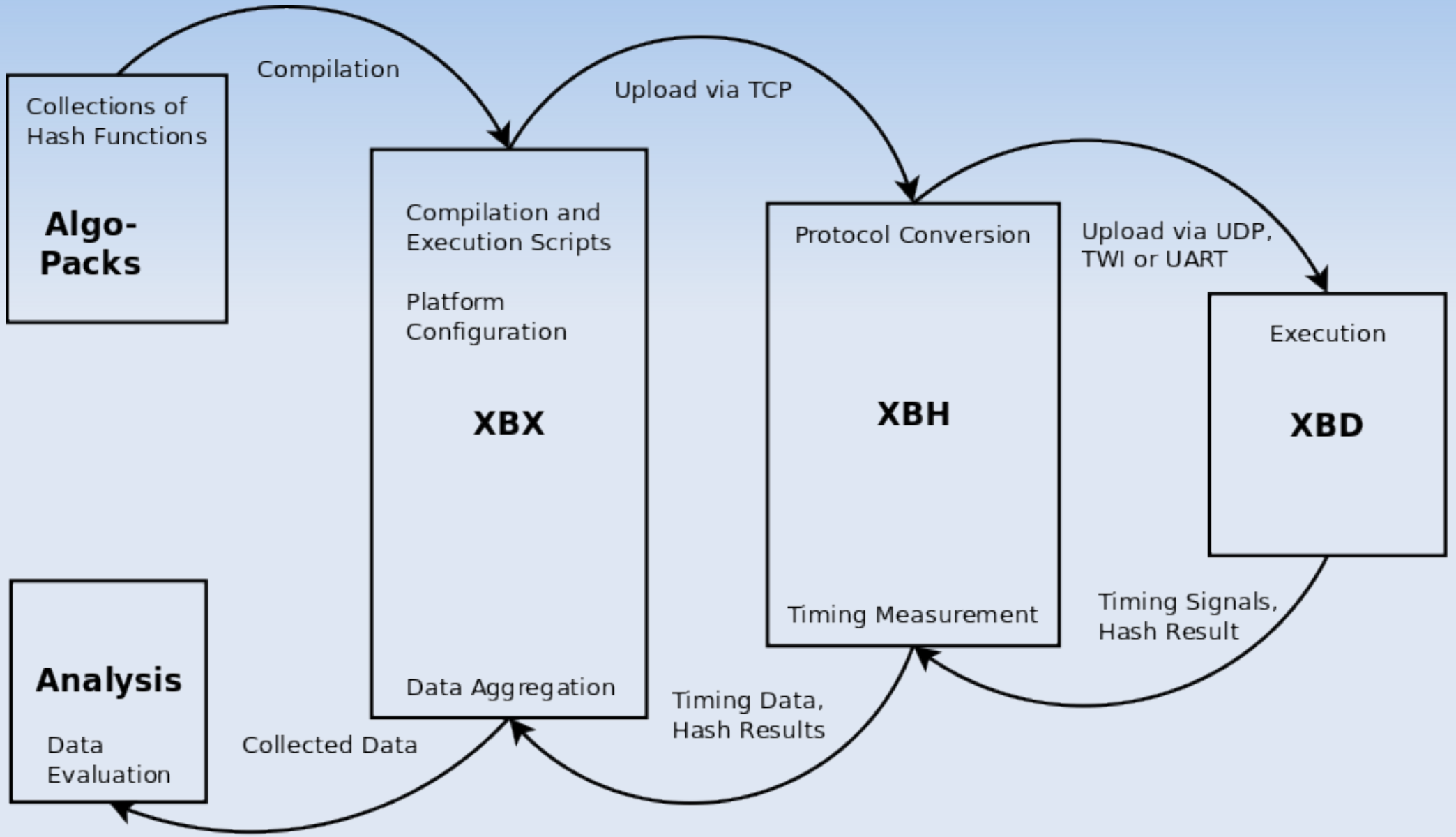
- Designed to closely emulate SUPERCOP
- Builds binaries from algorithms under test
  - Using different implementations
  - Using different compiler options
  - Using different compilers if available
- Tests binaries (try phase)
  - Execute a known-answer checksum test
  - Verify the result, flag broken binaries
  - Measure and log the time the operation needs
  - Measure and log stack usage

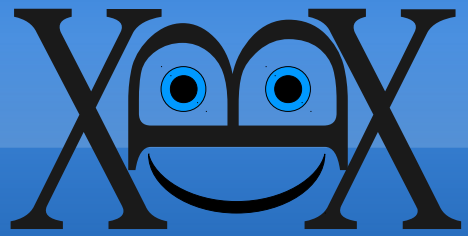


Software

- Tests binaries (measure phase)
  - Fastest implementation-compiler-options triple per algorithm is subjected to detailed benchmarking
  - Using different input/plaintext sizes
- Reports results
  - Detailed timings from measure phase
  - Static sizes from binary files (e.g. ELF,COFF)
  - Stack usage from the try phase
  - Generate best-of lists: Speed, RAM, ROM

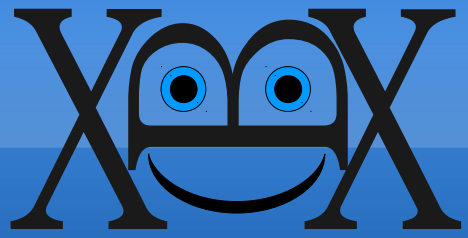
Software


Collections of Hash Functions
Algo-Packs
Compilation
Compilation and Execution Scripts
Platform Configuration
XBX
Data Aggregation
Upload via TCP
Protocol Conversion
XBH
Timing Measurement
Upload via UDP, TWI or UART
Execution
XBD
Timing Signals, Hash Result
Timing Data, Hash Results
Collected Data
Analysis
Data Evaluation

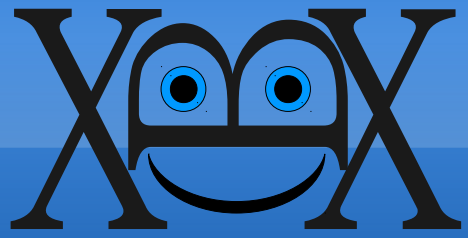


Software

- PC-based XBX software
  - Mostly Perl scripts
  - Some Bash scripts
  - SQLite for results analysis
- XBH software
  - C, some assembler
- XBD software
  - C, some assembler on small targets, some bash on embedded linux targets

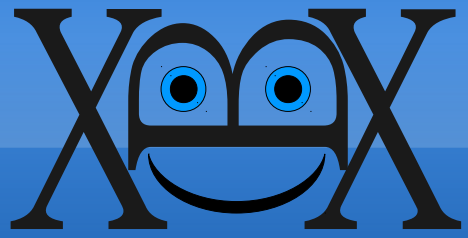


Reviewer Comments



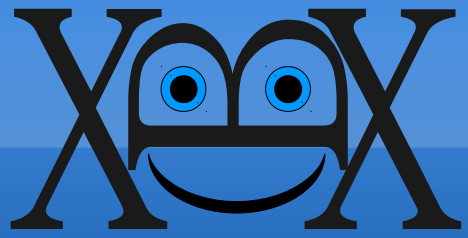
Reviewer Comments

- Why do you not use a XBH external clock pin to clock the XBD? This would give you the best timing accuracy.
- Yes, it would. Drawbacks would be:
  - None for self-designed AVR boards
  - Crystal removal and voltage level shifting for most (3.3V) microcontroller eval boards
  - Same for commandeered routers or NAS devices
  - Some on-chip oscillators and/or PLLs might not work with externally applied clock

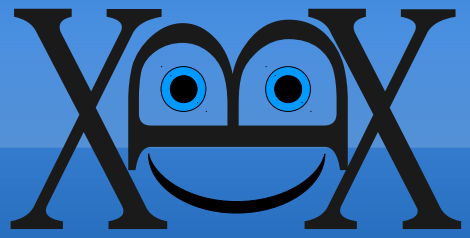


Reviewer Comments

- What about a multi XBD capable XBH, where you can switch between the desired target platform?
- → Feasible, but not with an ATmega644 due to limited RAM, I/O and timer resources
  - Could do it with a modern 32bit microcontroller
  - Current XBH has a €40 pricetag incl. accessories
- → Current solution is the ”XBX farm”
  - Severals XBH-XBD pairs, each with own IP
  - One Linux PC, XBX software plus ”farm” scripts

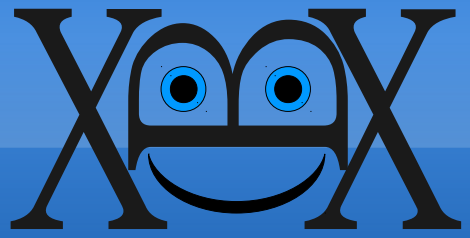


Example Benchmark Results

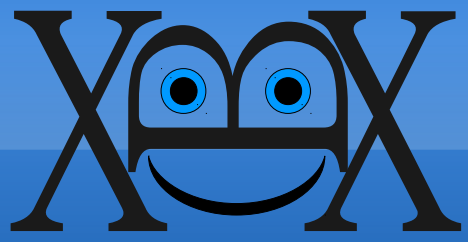


Hamsi, Speed

Platform	Try Cycles (1536)	Empty Message	8 Bytes	64 Bytes	512 Bytes	1024 Bytes	2048 Bytes	Long Messages	Compiler
lm3s811-evb	329483	3874	696	272	220	216	214	212	arm-elf-gcc -O2
fritzbox-7170	399196	52391	7009	1172	347	283	250	218	mips-uclib-gcc -O2
artila_m501	413696	19485	2734	585	304	278	272	266	arm-artila-gcc -O1
nslu2-openwrt	475957	26667	3780	798	387	334	297	261	armeb-uclibc-gcc -O1
atmega1281_16mhz	8312273	89949	16595	6758	5529	5441	5397	5353	avr-gcc 1281 -O3
atmega1284p_16mhz	8312293	89950	16595	6758	5529	5441	5397	5353	avr-gcc 1284p -O3

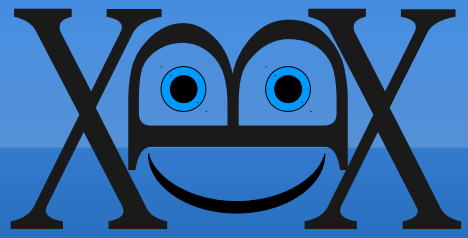


Present & Future



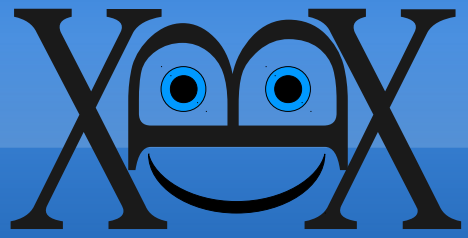
Present

- Fully automated benchmarks, speed and size
- XBD communication by I²C, UART or Ethernet
- Four XBD families currently supported
  - Atmel ATmega
  - Artila M501 (ARM 920T)
  - NSLU2 (Intel XScale, ARMv5te)
  - Fritz!Box 7170 (Texas Instruments AR7, MIPS)
- Website: https://xbx.das-labor.org/trac/wiki
- Peer-reviewed paper



Future

- Comprehensive hardware documentation
- User's Guide
- More XBDs
- XBD power benchmarking
  - Voltage & Current measurement
  - Will require power management on XBD
- More SUPERCOP: ciphers, signatures...
- FPGA boards as XBDs



Thank You!

Questions, Comments?