DEVELOPMENT AND BENCHMARKING OF CRYPTOGRAPHIC IMPLEMENTATIONS ON EMBEDDED PLATFORMS

by

John Pham

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

_____	Dr. Jens-Peter Kaps, Thesis Director
_____	Dr. Kris Gaj, Committee Member
_____	Dr. Craig Lorie, Committee Member
_____	Dr. Monson H. Hayes, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: _____	Summer Semester 2015 George Mason University Fairfax, VA

Development and Benchmarking of Cryptographic Implementations on Embedded Platforms

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science at George Mason University

By

John Pham
Bachelor of Science
George Mason University, 2009

Director: Dr. Jens-Peter Kaps, Professor
Department of Electrical and Computer Engineering

Summer Semester 2015
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my parents and friends.

Acknowledgments

I would like to thank Dr. Jens-Peter Kaps and Dr. Kris Gaj for their teaching, guidance, support, and patience during my time at GMU, as well as giving me the opportunity to complete my Master's here. I would like to thank Dr. Kaps additionally for his guidance and patience regarding this thesis.

I would also like to thank my friends at CERG, especially Marcin, Rajesh, Ice, Ahmad, Panasayya, Umar, and Rabia for making my time at CERG fun and interesting, and their help on various things.

Finally I would like to thank my friends Allie and Kat for their support during tough times.

Table of Contents

List of Tables

List of Figures

Abstract

DEVELOPMENT AND BENCHMARKING OF CRYPTOGRAPHIC IMPLEMENTATIONS ON EMBEDDED PLATFORMS

John Pham, M.S.

George Mason University, 2015

Thesis Director: Dr. Jens-Peter Kaps

In 2007, the National Institute of Standards and Technology (NIST) announced the Secure Hash Function-3 (SHA-3) competition to select a successor of the SHA-2 standard after vulnerabilities were discovered in the related SHA-1 algorithm. The Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was announced in 2013 to select algorithms for Authenticated Encryption and Associated Data (AEAD) that exceeded the performance of Advanced Encryption Standard-Galois Counter Mode (AES-GCM). As part of both competitions, algorithm implementations in hardware and software were compared in order to evaluate the performance and other characteristics. An area of interest is performance on embedded devices, where resources are more constrained.

Embedded devices are becoming increasingly interconnected. Formerly "dumb" appliances, such as thermostats, light bulbs, door locks, coffeemakers, and insulin pumps now have the ability to connect to the Internet. This is done in order to allow remote control and monitoring. Therefore, their communications need to be secured via encryption, in order to maintain privacy, or prevent malicious control. This may come at the cost of increased complexity, decreased throughput, increased energy consumption, and increased storage or die area. In order to minimize these costs, standards should take embedded performance into consideration.

In this thesis, we describe our support of the SHA-3 and CAESAR competitions. We made a lightweight implementation of the SHA-3 finalist JH [72] in hardware on FPGAs as part of a comparison between other lightweight implementations of SHA-3 finalists. For fair, comprehensive, and automated evaluation of hardware, the Computer Engineering Research Group (CERG) at George Mason University (GMU) developed the ATHENa tool. We supported this effort by creating a searchable online database to store and present the results to the research community. For software, the eXternal Benchmarking eXtension (XBX) [71] evaluates SHA-3 candidates on several microcontrollers. We ported XBX to the MSP430 platform and reported results.

As part of the CAESAR competition, we overhauled XBX to cover Authenticated Encryption and Associated Data (AEAD) ciphers, ported the test harness to a more capable platform, and proposed a means to measure power. We also extended the ATHENa database to support authenticated ciphers.

Chapter 1: Introduction

In 2007, the National Institute of Standards and Technology (NIST) announced the Secure Hash Function-3 (SHA-3) competition to select a successor of the SHA-2 standard after vulnerabilities were discovered in the related SHA-1 algorithm [70]. The Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was announced in 2013 to select algorithms for Authenticated Encryption and Associated Data (AEAD) that exceeded the performance of Advanced Encryption Standard-Galois Counter Mode (AES-GCM). As part of both competitions, algorithm implementations in hardware and software were compared in order to evaluate the performance and other characteristics. An area of interest is performance on embedded devices, where resources are more constrained.

Embedded devices are becoming increasingly interconnected. Formerly "dumb" appliances, such as thermostats, light bulbs, door locks, coffeemakers [38], and insulin pumps [41] now have the ability to connect to the Internet. This is done in order to allow remote control and monitoring. These embedded devices communicating need a means to secure this communication via encryption [41], in order to maintain privacy, or prevent malicious control. This may come at the cost of increased complexity, decreased throughput, increased energy consumption, and increased storage or die area, as encryption adds a step that requires additional processor time, or dedicated hardware. In order to minimize these costs, standards should take embedded performance into consideration.

In this thesis, we describe our support of the SHA-3 and CAESAR competitions. We made a lightweight implementation of the SHA-3 finalist JH in hardware on FPGAs as part of a comparison between other lightweight implementations of SHA-3 finalists. For fair, comprehensive, and automated evaluation of hardware, the Computer Engineering Research Group (CERG) at George Mason University (GMU) developed the ATHENa tool [28]. We supported this effort by

creating a searchable online database to store and present the results to the research community. For software, the eXternal Benchmarking eXtension(XBX) evaluates SHA-3 candidates on several microcontrollers. We ported XBX to the MSP430 platform and reported results.

As part of the CAESAR competition, we overhauled XBX to cover Authenticated Encryption and Associated Data (AEAD) ciphers, ported the test harness to a more capable platform, and proposed a means to measure power. We also extended the ATHENa database to support authenticated ciphers.

1.1 Cryptographic Standards and Competitions

Cryptographic standards are important in order to maintain interoperability, and to identify which algorithms are suitable for various purposes. Cryptographic standards are defined by organizations such as NIST, who may bless an existing algorithm or sponsor a competition, in the case of AES. Competitions have the benefit of allowing algorithms to be scrutinized in the open. For SHA-3, NIST decided to use a competition to pick the successor to SHA-2, which was designed by the NSA and blessed by NIST as the standard.

Extensive hardware and software performance metrics are desirable as they are part of the criteria for deciding on an algorithm, with the other criteria being security. Standards need to perform well and remain secure on large variety of platforms for the lifetime of the standard. Once an standard is promulgated, it is extremely difficult to make revisions. Metrics are also useful as feedback to algorithm designers during parts of the competition where minor revisions are allowable.

Results are usually obtained via simulation in the case of hardware, or by actually running the algorithm in the case of software. There is some difficulty in this, for multiple reasons. Implementation author skill variance, restrictions on source codes impeding verifiability, and difficulties in simulating power usage impact both software and hardware [26]. For FPGAs, there is no single convenient cost metric, as FPGAs have multiple available resources that need to be measured, such as DSP units, in addition to registers and logic units.

In addition, in the early stages of a competition, there are many candidates that need to be evaluated. Optimized versions for every algorithm on every platform, ideally written with the same skill level are needed, but is difficult to obtain.

Lightweight implementations are important, as an increasing number of embedded devices are now connected into the "Internet of Things." Embedded devices typically have fewer available resources. A lightweight implementation could allow securing these devices without compromising functionality or switching to more expensive hardware.

An algorithm's ranking in a high speed implementation may differ than that for a lightweight implementation, as some algorithms have features that do not scale down well, or the reverse, not scaling up well. An extreme example would Blue Midnight Wish (BMW) [32], which has an irregular internal structure not conducive to de-duplication (folding), and thus does not scale down well.

Chapter 2: Low Area Cryptographic Hardware Designs

2.1 Introduction

Hardware-based designs are typically used where throughput requirements cannot be matched by software. Hardware is also used when freeing up the central processing unit (CPU) for other tasks is considered valuable. Purpose designed hardware can also be more resistant to various side channel attacks, e.g. cache timing attacks [4] or power analysis [64], which is useful for smartcards and trusted platform modules. Resistance against cache timing attacks is one of the motivations for the AES-NI instruction set in Intel CPUs [36].

In embedded applications, hardware resources are limited and a smaller design will allow for a smaller Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC), which lowers unit costs. Even in non-embedded applications, low area designs are important, as devices will seldom be solely performing cryptographic applications and will have to share resources with other functionality. A smaller design could allow for area to be reassigned elsewhere. In this chapter we will discuss important metrics for hardware design, and discuss optimization techniques for low-area.

2.2 Metrics

For hardware cryptography implementations, the properties of interest are area, throughput, latency, and power/energy usage.

2.2.1 Area

Area is measured differently depending on the type of implementation. The major types of hardware implementations are ASICs and FPGAs. In this paper, we will concentrate on optimizing

for area on FPGAs.

ASICs

An ASIC is what one would think of as a "computer chip", created at a fab (short for semiconductor fabrication plant). They have the advantage of being efficient in terms of energy, have higher throughput, and have a low unit cost compared to FPGAs. Area is either given as a process-specific physical area, or as a process-independent gate equivalent by dividing the area of the design by the area of a NAND gate [6].

FPGAs

An FPGA is a chip that can have the gate layout programmed in the field, as opposed to having the gate layout hard wired. They have the advantage of having a much lower nonrecurring costs than ASICs, at the cost of higher recurring costs and lower performance, in terms of both power and clock speed . On an FPGA, the area is usually given as the number of various resources on the FPGA that are being used, which are dependent on the device family and vendor. Typically there is one type of resource unit consisting of lookup tables used to emulate logic gates (slices on Xilinx, logic elements on Altera), and other more specialized units to do arithmetic, memory, I/O, etc.

2.2.2 Throughput

Throughput is the amount of data that can be processed in a fixed amount of time. It is typically stated as bits/sec or bytes/sec. This depends on how much the architecture can process in a given cycle, and the clock rate which is dependent on the underlying process or FPGA family as well as the architecture's critical path in terms of gates.

2.2.3 Latency

A related property is latency, which is the time in number of cycles between data input and seeing the output of that data. This depends on the architecture. Designs that are pipelined to

obtain a higher clock rate will typically have greater latency in terms of cycles (although the time taken by cycles may be shorter, as pipelining allows higher clock rates).

2.2.4 Power/Energy

Power, measured in Watts, is the rate at which energy, measured in Joules, is consumed. Some devices are power-limited, and other are limited by the total energy that is available due to, for example, being battery powered. Power usage will also affect the total energy that can be extracted from non-ideal batteries [47]. Power is divided into two types: static (quiescent) power and dynamic power. Static power is the power consumed from leakage when idle, while dynamic power is the power consumed when the logic gates are switched, and is dependent on the activity on the chip as well as the clock frequency.

2.3 Low Area Design

In this section we will discuss low-area optimization via folding, component reuse, and Read-only Memory (ROM) based control units. Details specific to our JH design will be discussed under SHA-3.

2.3.1 Folding

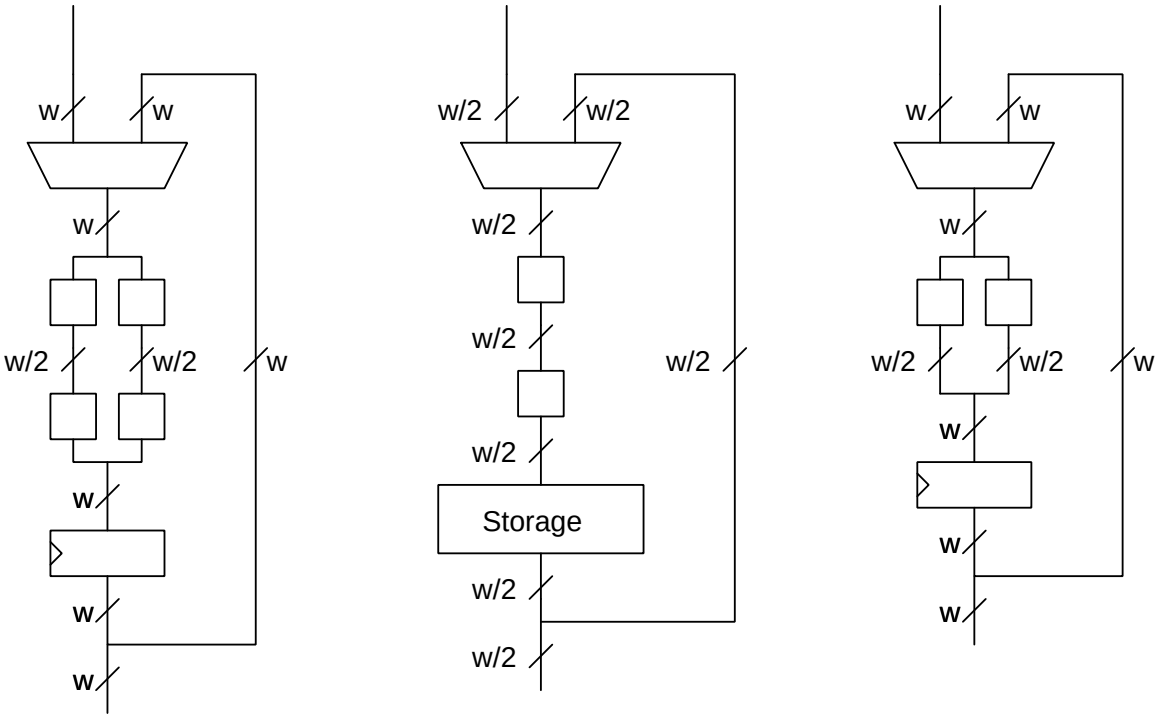

Figure 2.1: Folding. From left to right: Unfolded, 2x Vertically Folded, 2x Horizontally Folded

Horizontal folding is the reduction of datapath length while keeping the width constant. Length in this case refers to the amount of logic between input and output. Vertical folding reduces the datapath width [40], where width refers to how much data is processed at a time. Horizontal folding is akin to using a loop instead of repeating code in software, while vertical rolling is analogous to processing an array of elements fewer at a time, or reducing from a 32 bit processor to an 8-bit one. Horizontal folding negatively impacts performance by requiring more clock cycles to process the same amount of data. However, this is offset by allowing shorter clock cycles, as the critical path is shortened. This is not true of vertical folding, where the critical path is often constant.

2.3.2 Component Reuse

To reduce size, larger components are often multiplexed to become part of multiple operations. This must be performed with care, as multiplexers have their own significant size penalty.

2.3.3 ROM-based Control Unit

Due to the folding and component reuse, addressing can be particularly complex, resulting in a large control unit. In order to save logic units, a ROM-based control unit can be used [**rom**]. An FPGA uses a small memory storing a lookup-table (LUT) with a single output, and n inputs. The LUT stores 2^n bits. Using the LUTs directly as memory to store outputs instead of as logic to generate the outputs is more efficient in terms of area, power, and performance in an FPGA. However, increased numbers of inputs increases the size of the ROM exponentially. Therefore, a small amount logic is used in order to compress input signals into fewer signals [29] in order to keep area small.

Chapter 3: Cryptographic Software Benchmarking

3.1 Introduction

Software based designs are used where performance and security requirements do not justify the expense of a hardware solution.

Implementations are more quickly put into the field, as software can be deployed on existing hardware. It takes a considerable amount of time and expense to develop, validate, and manufacture hardware. This is particularly important for adoption of a new standard.

Software (along with FPGA implementations) with source code has the advantage of also being more easily verified by the end user to be free of tampering, as the source can be compiled and programmed by the end user.

There are many existing tools for benchmarking cryptographic software. The System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives (SUPERCOP) is a cryptographic benchmarking tool that benchmarks a large variety of cryptographic primitives on general purpose computers[**supercop**]. However, it is not capable of being run on embedded platforms, and lacks important metrics for embedded such as ROM and RAM usage.

Two software benchmarking tools for embedded are XBX[71] and FELICS [12][18]. XBX is an tool for evaluating hash functions packaged for SUPERCOP, and primarily runs implementations on actual hardware.

FELICS is a newer tool, focused on lightweight primitives, and current lightweight block ciphers, and adds cycle-accurate simulation for a variety of platforms as well as running implementations on actual hardware.

In this paper, we further extend XBX to cover AEAD primitives in addition to XBX, and allow easy extensibility to support the rest of the operations SUPERCOP supports. In addition, we intend to add the ability to measure power and energy consumption.

3.2 Metrics

For embedded software cryptography implementations, the areas of interest are ROM usage, Random Access Memory (RAM) usage, throughput, and power. This: differs from non-embedded environments, where typically only throughput is considered when evaluating performance, as disk storage (analogous to ROM), RAM, and power are plentiful.

3.2.1 Throughput

Throughput for cryptographic software is given in cycles per byte, in an effort to be independent of different clock speeds within the same family of CPU, as well as variable clock speeds due to power saving [13]. This only helps with variations in clock speeds, or comparing between otherwise identical architectures with varying features such as a Single Instruction Multiple Data (SIMD) unit like NEON or SSE. Results from different instructions per clock and/or instruction formats still cannot be compared with each other.

3.2.2 ROM

ROM usage is typically determined by the program code as well as any constants, and preinitialized variable data. Microcontrollers have anywhere between a few kibibytes to a mebibyte of ROM. This is typically consists of the `.text` (code and constants) and `.data` (initialization data for static buffers in RAM) sections in a compiled binary.

3.2.3 RAM

RAM is organized into statically allocated RAM set aside at the beginning of program execution, stack memory that grows as function call chains get deeper, and dynamic allocations via `malloc` and similar functions. The statically allocated RAM is divided into `.bss` and `.data` sections. `.data` is for preinitialied modifiable buffers in RAM that also take up space in ROM for the initialization data as previously mentioned, while `.bss` is for uninitialized buffers that only occupy RAM.

3.2.4 Power

As software ultimately runs on hardware containing a processor, the same constraints on power that affect hardware affects software. The power and energy used is influenced by which processing units the software uses, the microprocessor the software runs on, the clock speed and the total running time.

3.3 Code for embedded

Typically, memory usage (ROM and RAM) is a bigger constraining factor in embedded applications than speed [71]. In order to optimize for the limited program space in ROM, code should be rolled into a loop and broken up into functions for reuse. This is the opposite of what is often done for cryptographic code on platforms without memory constraints, which are often highly unrolled and inlined for performance reasons.

Due to limited memory, it is often desirable to compute large tables of constants on the fly instead of using lookup tables, which also have security implications if the indices are data dependent, if processors contain a cache, as timing differences due to varying access times can be used as a side channel [4].

In embedded applications, `malloc` is often avoided as it is nondeterministic and can fail due to memory fragmentation in memory constrained environments [48]. Often, functions such as `printf` will invoke `malloc` and must also be avoided.

Integer type lengths should be explicitly stated, using the types defined in `inttypes.h`, such as `int32_t` instead of int, `int8_t` instead of char, etc. `size_t` should be used for values holding memory sizes or memory indexes where the exact size of the value isn't important. The reason is microcontrollers can have different sizes assigned to `long`, `int`, `short`, and `char`, and explicitly stating the length allows consistent behavior. Davis also [33] recommends using `[u]int[8,16,32]_fast_t` for local variables likely to be stored only in registers to avoid sign extension/truncation instructions.

Care should be taken to account for endianness of different platforms.

Attention should be spent on C struct ordering. Due to alignment requirements, ordering struct members in arbitrary orders can cause padding to be inserted, wasting precious memory [20]. For the example below, the struct `foo` wastes 3 more bytes of space compared to the struct `bar`, on an architecture such as ARM which requires 32-bit (4 byte) alignment:

```
struct foo{
    uint8_t a;
    // 3 bytes of padding to be 32-bit aligned
    uint32_t b;
    uint8_t c;
    // 3 more bytes of padding to be 32-bit aligned
}
struct bar{
    uint32_t b;
    uint8_t a;
    uint8_t c;
    // 2 bytes of padding to be 32-bit aligned
}
```

The simplest way to pack structures is to order the members in decreasing order.

3.4 SUPERCOP

SUPERCOP "[...] is a toolkit developed by the VAMPIRE lab for measuring the performance of cryptographic software [**supercop**]." The project collects and benchmarks many implementations across many primitives on multiple platforms. It consists of a series of shell scripts and test harnesses that compile implementations and their dependencies across different compilers and flags. It then verifies outputs and correctness, and measures performance characteristics. Power measurements, code size, and memory usage data measurements are not supported.

3.4.1 Detailed description of SUPERCOP operation (as of 2015-07-02)

The latest version of SUPERCOP can be obtained from `http://bench.cr.yp.to/supercop.html` After unpacking the tarball, the user executes the `do` script.

This script runs subscripts that compile test programs that determines the valid {compiler, flags} tuples, application binary interfaces (ABIs), CPU information, and type sizes. It also compiles the `killafter` binary which is used to run the compilation of implementations and the implementations themselves with a specified time limit.

For each ABI and compiler, flags combination it builds `gmp` [63], which is a library for handling arbitrary precision numbers and is a dependency of some primitive implementations included with SUPERCOP, and `cryptopp` [10], which some primitive "implementations" included with SUPERCOP wrap.

The script loops through each operation, primitive, ABI, implementation, and compiler, flags. In the context of SUPERCOP, operation is defined as the class of algorithms, e.g. symmetric encryption, authenticated encryption, hash functions, etc. Primitive is a specific algorithm belonging to an operation, for example, Keccak [34] or AES. An implementation is code that performs the primitive.

For each implementation, it generates header files and C macros to interface the implementation to a generic test harness specific to the operation. Then for each {compiler, flags}, the script compiles and executes the implementation with the test harness, which verifies that the implementations produce the expected results, do not overflow buffers, resist attacks, etc.

Some primitives will have a "used" flag indicated by a file of that name in the primitives directory. This indicates that the primitive is a dependency. The best implementation of a primitive in terms of throughput is added to a library ("libsupercop") for each ABI, which is linked in during compilation and testing of subsequent primitives.

SUPERCOP results are output as gzipped space separated values in plain text to `bench/<hostname>/data.`

3.5 eXternal Benchmarking eXtension

The eXternal Benchmarking eXtension (XBX)[71] is an extension to SUPERCOP that covers platforms that are not capable of hosting a compiler and require cross-compilation, such as microcontrollers or small embedded Linux systems. It reuses the primitive implementations collected by the SUPERCOP project (which it refers to as an "algopack"), as well as others, and attempts to retain the same output data format. XBX is able to test multiple implementations across multiple primitives with multiple compilers and compiler flags without intervention, which is important with the large number of combinations to be tested. Due to the importance of small code and memory size for embedded platforms, these are also measured and recorded.

3.5.1 Goals

The goals of XBX, as stated by the XBX project, are [71]:

- Automated testing
- Performance metrics for different message lengths typically encountered in the real world
- Free and open source code
- Low cost hardware
- Compatibility with SUPERCOP algopacks
- Compatbility with SUPERCOP result format
- Use of free or low-cost development tools
- Component reuse for rapid development
- Focusing on the SUPERCOP subset which benchmarked hash functions, which was of interest at the time, due to the SHA-3 competition

3.5.2 Components

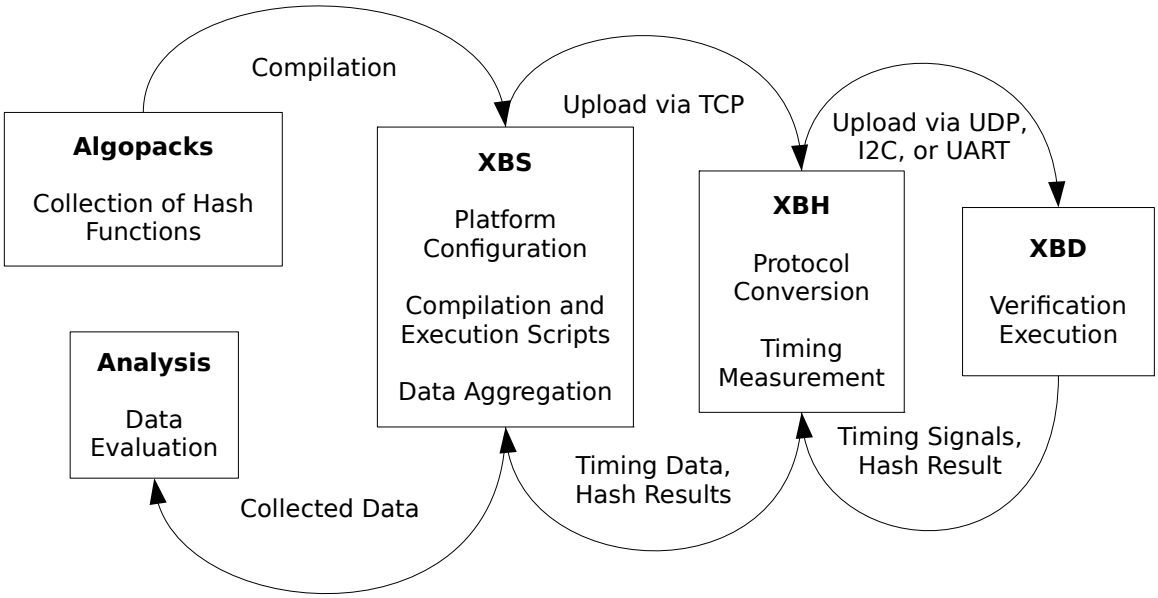

Figure 3.1: Block Diagram of XBX components [71]

XBS

XBX is comprised of the eXternal Benchmarking Software (XBS) running on a desktop computer, the eXternal Benchmarking Harness (XBH), and the eXternal Benchmarking Device (XBD), as shown in figure 3.1.

The XBS is software running on a PC that consists of scripts that handle compilation of algopacks and firmware upload to the XBH, as well as data aggregation and logging. Static allocation of RAM and ROM usage is also calculated from the compiler output. Multiple compilers and compiler flags are used to compile an implementation.

XBH

The XBH is a board that translates high level commands from the XBS sent via TCP/UDP to series of commands over a protocol understood by the XBD (I^2C , RS232, or Ethernet). Future references to the XBS communicating to the XBD refers to this. It also measures execution time via the timer capture pins, which precisely record the time that an edge event occurs. Because not all XBD platforms support cycle counters, the execution time is measured externally. The current XBH consists of a AVR-NET-IO ATmega32 board with ethernet connectivity, running code on bare-metal[3]. The ATmega32 is a 8-bit microcontroller with 2kiB of SRAM and 32kiB of ROM.

XBD

The XBD is the platform hardware that the primitive implementations are tested on. It performs the execution task and notifies the XBH of start and completion via a general-purpose input/output (GPIO) pin connected a timer capture pin on the XBH. It also returns the maximum stack RAM used to the XBH over the XBD↔XBH communication channel. The supported platforms are ARM, AVR, MSP430, and MIPS.

The XBD code consists of two components. One is a small bootloader that is flashed once to XBD that performs basic calibration and self programming of the primitive implementations over the communications channel.

The second component is a test harness that is linked with the primitive implementation that is downloaded over the communication channel. This code is responsible for signaling start and stop, supplying the implementation with input data, verifying the correctness of the algorithm implementation and returning the output to the XBH.

Both of these are linked to code which provides device specific drivers for communication and execution signaling, as well as stack measurement.

3.5.3 Typical Execution

Prior to running any scripts, the XBD is initialized by compiling the platform-specific bootloader and flashing it normally (i.e. using a programming cable). This bootloader has the responsibility of downloading code from the XBH. This allows board-specific programming tools to be avoided, besides the initial setup, and thus all subsequent operations can be performed by the XBS over a network, potentially as part of a farm.

Scripts from the XBS compile all implementations of primitives, linking with the hardware abstraction layer (HAL) and the test harness, with the possible {compiler, flags} defined in the platform support files. The number of generated application binaries should be {compilers, flags}× primitiveimplementations. The UNIX `size` command is run on the generated application binaries to get the sizes of `.bss`, `.data`, and `.text`. Implementations listed in a platform-specific blacklist and a global blacklist are omitted from compilation. Binaries are compiled into Executable and Linkable Format (ELF) which is then converted into Intel Hex (IHEX) for transfer to the XBH. XBX checks to see if a binary will fit, otherwise it will not be loaded later.

The XBS then orders the XBH to calibrate cycle calculations on the XBD. The XBD's bootloader executes a busy loop for a fixed number of cycles determined by the HAL. The XBH measures the physical time executed. The cycles run is then calculated by dividing the measured time by the clock rate.

The XBS then sends the application binary to the XBD. This is sent one flash block at a time, as existing data on the flash must be erased, and this can only be done one block at a time. After each block is completely received, the XBD writes the block from the RAM buffer to flash memory.

When all blocks are received, the XBD will switch to the application by calling the address of the application's entry point, which is set to a fixed address by linker scripts or compiler flags. This is the code that copies `.data` from ROM to RAM, zeroes out `.bss`, and initializes the stack pointer, then calls main. This is typically defined by the C runtime, and is often known as "crt0". However the ARM Stellaris platform has this defined as part of the HAL, as it is based off example code provided by the chip vendor which also does this. As the initialization is run, the

previous call stack and static memory allocations of the bootloader is obliterated and replaced with that of the application. A similar procedure is applied to switch back to the bootloader after application execution.

Once in application mode, the XBD is ordered to verify correct execution of the primitive implementation. This is done by hashing repeatedly (XBX only supports hashing), mixing in the results of the previous hash, and comparing the results to that generated by the reference implementation on a PC.

The XBD is then ordered to perform the actual benchmarks. Data randomly generated by the XBS of varying lengths (0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 576, 1024, 1536, and 2048 bytes) is sent to the XBD, and hashed. The XBD signals to the XBH the start and stop times by setting a signal line low upon start and raising it high again upon stop. Any delays in detecting timing should be symmetrical across start and stop and should not affect the result [71]. This is repeated multiple times (defaulting to 5).

Stack usage is measured by painting memory with a canary value from the top of the statically allocated memory sections to the current location of the stack before executing the implementation. After execution, the number of addresses still containing the canary value and not overwritten by the call stack growing when the implementation returns is subtracted from the number of addresses painted to determine the amount of stack memory used.

Upon execution completion, the XBS will request the measured stack value from the XBD and the timing results from the XBH. These values are logged and the process repeats starting with loading the next application binary, until all of them are tested.

Results are output as a gzipped space separated file named `xbxdata-[hostname]-[XBD IP]-[timestamp]`.

Our previous contributions to XBX

We contributed the MSP430FG4618 port to XBX, and provided results for this platform for the SHA-3 competition [9]. Prior to this, the XBX project did not support a 16-bit microcontrollere of any kind. Much of the understanding of XBX gained from this was of great utility for extending

XBX.

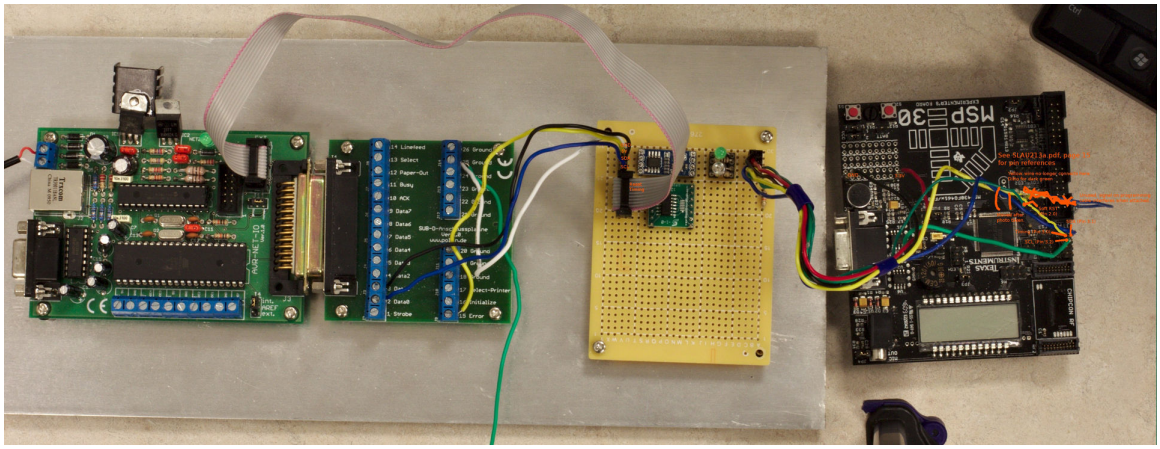

Figure 3.2: Our XBX setup with the MSP430FG4618. The XBH is on the left, and the XBD is on the far right, with a level shifter in the middle

XBX Limitations

XBX currently does not handle operations supported by SUPERCOP other than hash functions. This was due to the operations of interest at the time being hash functions for the SHA-3 competition.

Power measurements are also out of scope. Due to performance reasons, the checks for correctness that XBX does on the algorithms is not as complete and is limited to verifying that the output is what was expected, as opposed to SUPERCOP which does bounds checking of input and output buffers.

Cycle counters aren't used on architectures where they are available, as the intention was to use physical measurements of start and stop signals universally, as cycle counters aren't available on all platforms. This can be problematic for architectures that have a variable clock rate, or where the clock rate isn't stable, i.e. the processor uses some kind of internal oscillator instead of a crystal.

Chapter 4: Power and AEAD additions to XBX

4.1 Goals

Our primary goal in extending XBX is to generalize it in order to support operations other than hashing, and enable power benchmarking, in addition to the previously mentioned goals of original XBX. We initially intended to keep as much backwards compatibility with original XBX as possible, especially with the HAL code and the algopacks. However, we decided to break backwards compatibility with the SUPERCOP results interface, as XBX's compatibility was not exploited by exporting the data to the SUPERCOP website. A binary database was considered to be better suited for querying and logging large amounts of data, and storing and resuming partial runs. The output format cannot be identical in any case because of the addition of power measurements, which we intend to perform. If compatibility with SUPERCOP results is desired, it should be trivial to write a script to extract results from the binary database and reformat it.

We intend to measure power directly, as opposed to simulation. Accurate simulations of power and energy performance for software are possible citecyclesim[7], but requires the availability of models and simulators for each platform, which may not be always available or which may be proprietary.

Our code covers AEAD ciphers, as the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [14] is the focus of current interest, now that the SHA-3 competition is complete.

In order to meet the goals of being low-cost and automated, we have to integrate power measurement hardware, as opposed to simply performing power measurements by hand on an oscilloscope. Assuming 8 different {compiler, flag} tuples, 174 AEAD primitives in SUPERCOP, 1 implementation per algorithm (assuming only portable reference implementations is available), and 52 different permutations of authenticated data/encrypted data message lengths, there are

72384 measurements that have to be performed. These have to be repeated a few times to get a good average. Due to the large number of measurements required, performing measurements by hand is not practical. In addition, oscilloscopes are not inexpensive and there is no standardized way to automate obtaining results from them. Digital multimeters are not suitable for measuring transient spikes.

Currently XBX supports AVR, MIPS, ARM (Cortex M0 and M3), and MSP430 boards. We intend to initially focus on MSP430 and ARM, specifically Cortex M4, and backporting to AVR and possibly MIPS later, as the hardware for these two platforms is what we initially have on hand.

Resuming interrupted runs was something we felt was desirable, as failure could interrupt a test run, and we did not want to have to rerun the entire test run from scratch. This was implemented but necessitated large rewrites in order to save state and prevent duplicate runs.

We support Link time Optimization (LTO), which looks at the entire generated binary when optimizing during the linking phase as opposed to a single file, This allows for more aggressive inlining and code-size reduction. However the aggressive optimizations may often break functionality and make debugging extremely difficult due to the aggressive inlining- stepping through the generated assembly is required.

4.2 Hardware

4.2.1 XBH Replacements

We plan on integrating power measurement hardware into the XBH. In order to keep up with polling power measurement hardware at a decent rate, running the TCP/IP stack, as well as having enough SRAM to buffer data, we decided the XBH needed to be replaced with a more capable platform. A built-in analog-digital converter (ADC) was deemed desirable in order to simplify hardware design and avoid I/O performance issues. The XBH replacement should also have an ethernet port, like the current XBH, as writing a network application is easier than writing to USB directly, and would allow code reuse. The XBH platform would also need both

hardware I^2C and UART connectivity in order to connect to XBDs. At the time that we initially evaluated the XBH replacements, the potential platforms we looked at were Raspberry Pi (1st generation) [53], the BeagleBone (original, not the Black), and the Spartan 3E starter board. We later also considered the EK-TM4C1294XL board from Texas Instruments (TI) [61].

Raspberry Pi

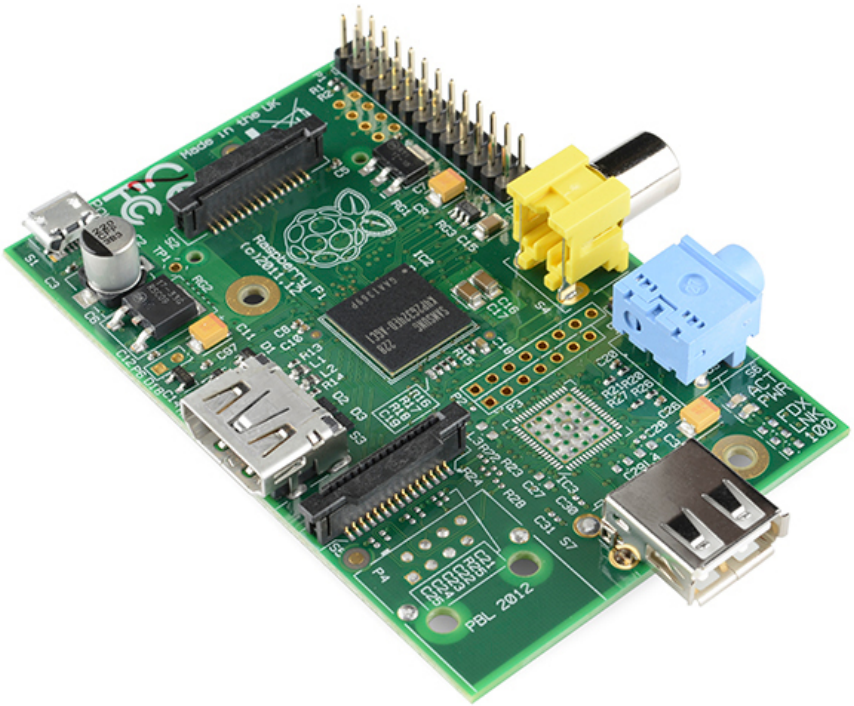

Figure 4.1: Raspberry Pi Model A [69]

At the time of initial evaluation, the Raspberry Pi was the cheapest platform at about 35 US Dollars (USD), and running Linux was considered a benefit due to ease of programming. The processor is a 700MHz Broadcom ARM11 chip, with 256MB of RAM, roughly equivalent to a 480MHz Cortex A8. However, it lacks ADCs and an external ADC would have to be interfaced

via Serial Peripheral Interface (SPI) or I^2C . A built-in ADC, however, would still require an analog frontend to have the proper impedence and biasing.

BeagleBone

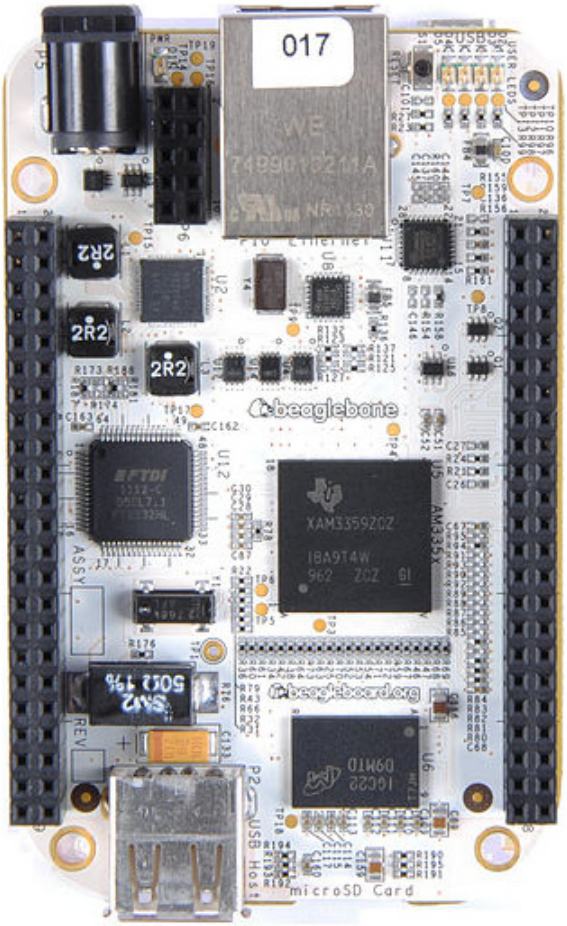

Figure 4.2: BeagleBone [23]

The original BeagleBone is significantly faster, running a 700MHz OMAP3 with 512MiB of RAM, which is TI's implementation of the Cortex A8. It has built-in ADC capable of 100k samples per

second, and also runs Linux. However it cost significantly more at around 90 USD. There is currently a newer version called the BeagleBone Black which is significantly cheaper at 45USD with a 1GHz processor.

FPGA Boards

FPGA boards were briefly considered, such as the Spartan 3E starter board, which has the benefit of being able to interface with extremely high speed ADCs, necessary for cycle-accurate power measurements. However, there were significant disadvantages, including cost and the difficulty of integrating IP cores to make a functional XBH. High speed power measurement was ultimately considered out of scope.

Realtime considerations

Due to cost reasons, the Raspberry Pi was favored over the BeagleBone, with power measurements handled by an external I^2C ADC. However, upon further evaluation a Linux based platform such as the Raspberry Pi and or the BeagleBone was considered unsuitable. The Linux distributions typically available for the Pi and the BeagleBone are not suitable for realtime usage. Logging of the XBD start and stop signals needs to be consistent with minimal jitter which is not possible with regular Linux.

Realtime Linux variants were then considered. There are two realtime extensions (PREEMPT_RT and Xenomai) available that provide realtime capabilities. PREEMPT_RT is a set of kernel patches that make various kernel functions preemptible by realtime tasks, amongst other changes [25]. Xenomai runs the Linux kernel alongside a realtime co-kernel [57]. The latest version has the option of merely being a wrapper for PREEMPT_RT to use traditional realtime operating system APIs, like VxWorks [37]. At the time of evaluation, PREEMPT_RT had issues with breaking the MMC driver needed to read the SD card holding the operating system, and the cokernel model of Xenomai required reimplementing drivers [52], which is a significant amount of work.

Even with the realtime extensions, jitter is in the order of at least tens of microseconds [5],

which is not acceptable when relying on timing measurements to count clock cycles in the XBH. Xenomai with the co-kernel option as in Xenomai 2.x does appear to perform significantly better than PREEMPT_RT.

Using the boards bare-metal appeared to be non-trivial, and alternative realtime OSes such as ChibiOS appeared to lack drivers for the ethernet hardware, due to it requiring a working USB stack.

Tiva C Connected Launchpad

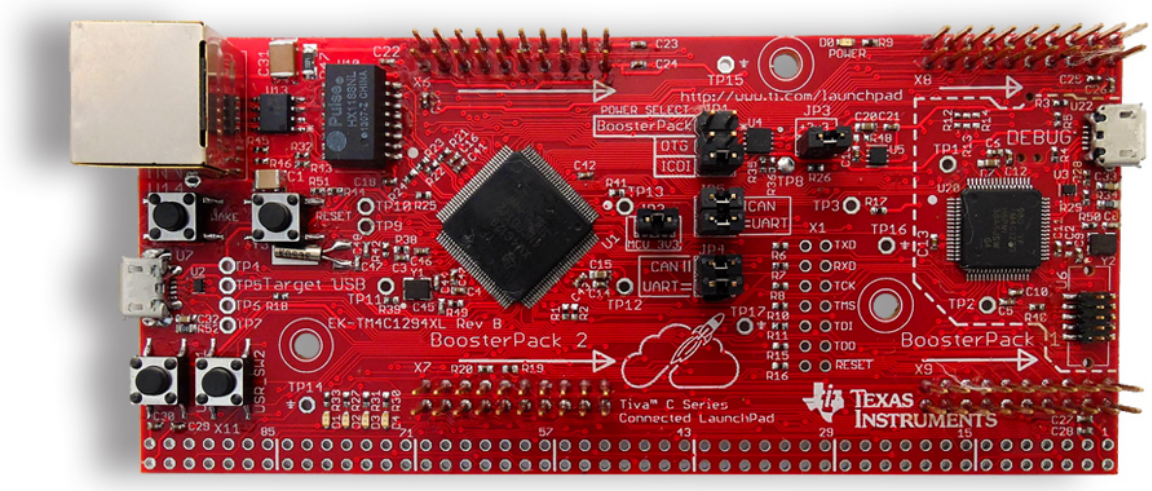

Figure 4.3: Tiva C Connected Launchpad[58]

At the same time, the EK-TM4C1294XL (Connected Launchpad) platform became available [61]. This is a 120MHz board using the TM4C1294NCPDT (hereforth referred to as TM4C1294), which is TI'S version of the 32-bit ARM Cortex-M4F, with ethernet connectivity. There is 256kiB of SRAM and 1MiB of ROM, which is far more than is available on the previous XBH. This board typically runs software on bare metal (without an operating system), and is suitable for realtime

usage. It also has two built-in 12-bit ADCs capable of 2 million samples per second.

Ultimately the EK-TM4C1294XL was chosen due to the realtime capabilities and low monetary cost, with the built-in ADC being a bonus.

4.2.2 Current Sensing

We are only concerned with energy/power usage as it affects battery power. The power used by the device is determined using the equation 4.1

$$P = IR \tag{4.1}$$

where P is power, V is the supply voltage, and I is the current. Energy is the integral of P over the run time. For our purposes, the supply voltage is fixed, usually at 5V or 3.3V. Therefore, only current needs to be measured.

Measurements will only need to be concerned with min, max, and average. Typically real devices will have capacitors and other power-related circuitry that smooths out short transients. Cycle-accurate power measurements are out of scope, and does not appear possible on the low-cost hardware that we desire.

High vs low-side

We considered multiple approaches for current measurement. Typically current is measured by sensing the voltage drop across a small shunt resistor. There are two possible configurations for this. One is to put the shunt resistor between the voltage source and the device. This configuration is called high-side current sensing, as shown in figure 4.4. The second is to put the shunt between ground and the device. This is called low-side current sensing [51], as shown in figure 4.5. The advantage of high-side current sensing is that it directly measures current delivered by the voltage source- multiple ground paths do not need to be taken into account, while low-side sensing does not have to deal with the higher common-mode voltage, which is the average voltage before and after the shunt. A low-side sensor can be simpler as it does not

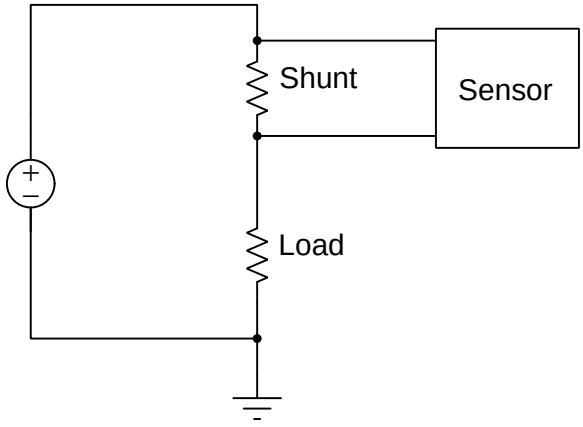

Figure 4.4: High-side current sensing

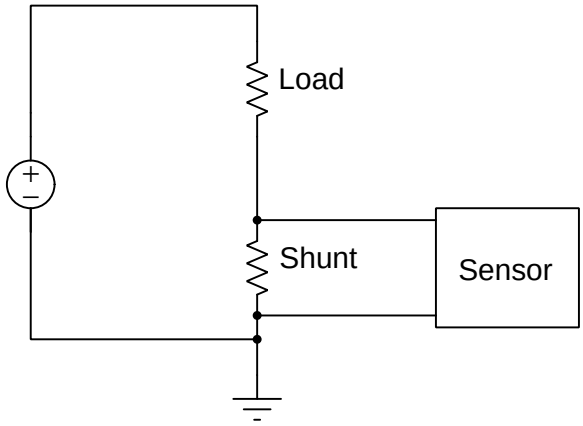

Figure 4.5: Low-side current sensing

have to subtract the supply voltage and can simply read the voltage between the device and the shunt.

Due to multiple paths to ground because of I/O pins, as well as the low cost of integrated circuits which easily deal with the common-mode voltage, we decided to go with high-side current sensing.

4.2.3 Sensors

Initially, since the Raspberry Pi was considered as the XBH candidate and did not have a built-in ADC, we looked at I^2C and SPI ADCs and current sensors with digital output. We picked the

Texas Instruments INA219 [62], as it performs both current and voltage sensing without needing much circuitry beyond filtering capacitors and a shunt resistor. Unfortunately the INA219 has a rather low sampling rate, at 1.7 kilosamples per second. The I^2C bus also adds some latency to measurements, and additional software complexity.

Because we switched the XBH board over to the EK-TM4C1294XL, the use of analog outputs became an option. Using the ADC to sense the voltage across the shunt resistor directly was considered, however this will not be an option for measuring devices that require a 5V supply, as the ADC will only do 0 to 3.3V , thus requiring a resistor network to divide voltage. More importantly, the built-in ADCs have a very low input resistance at only 2kΩ and need a buffer. In addition, the voltage drop across a 1Ω resistor is likely to be very small and needs amplification to use the full range.

An op-amp in differential configuration was subsequently considered for use in the high-side configuration, however integrated circuits including the resistor network purpose built for the purpose of current sensing are available and inexpensive, called current shunt monitors, or CSM. We decided to use these instead. These ICs come in current output and voltage output variants. We chose voltage output, as current output requires using a precision resistor to convert the output to a voltage signal recognizable by the ADC. In addition, the low input resistance of the ADC would negatively affect the accuracy of the result.

We then considered the Analog Devices AD8210 [1]. This CSM has a gain of 20, and importantly has a buffer to deal with the TM4C1294's low ADC impedance, as well as a 100kHz bandwidth. Using a shunt resistor of 1Ω , each milliamp of current will cause a 1 millivolt drop in voltage across the shunt, and a 20 millivolt signal to the ADC.

$$\frac{V_{\text{max ADC}}}{\text{Gain} \times R_{\text{shunt}}} = I_{\text{max}} \tag{4.2}$$

Using equation 4.2 where max ADC input ($V_{\text{max ADC}}$) is 3.3V, gain is 20, and shunt resistance

(R_{shunt}) is 1Ω, the maximum current that can be measured by the ADC (I_{max}) is 164mA.

$$\max\left(\frac{I_{\text{max}}}{2^{n_{\text{ADC bits}}}}, I_{\text{max}} \times \delta_{\text{CSM gain}}\right) = I_{\text{resolution}} \tag{4.3}$$

Using equation 4.3 where the ADC resolution ($n_{\text{ADC bits}}$) is 12-bit for the TM4C1294, and the CSM gain error (δ_{CSMgain} being 1/200, the minimum current that can be resolved by the ADC is 840 μA , which is insufficient. Gain errors may be tolerable, as we are mostly interested in fluctuations during execution, in which case we can use equation 4.4.

$$\frac{I_{\text{max}}}{2^{n_{\text{ADC bits}}}} \tag{4.4}$$

This gives a resolution of about 40.03μA . We initially thought this was sufficient, however upon closer examination of the MSP430F5529 datasheet [60], which is a device we are examining as an XBD, the resolution is insufficient, as typical execution uses only 290μA . Having a higher gain will not solve resolution problems for all boards, as there is potential for clipping off measurements on the high end with boards that use more current.

Subsequently, we looked at the INA225 [59] from TI, which has a programmable gain setting via GPIOs between 25 and 200, buffered output, and a 250kHz bandwidth at a gain setting of 25. The listed gain error is 1/1000. This allows for a current resolution of 16.5μA at 200 gain, topping out at 32mA, and 132μA resolution at a gain of 25 topping out at 132mA, when using equation 4.3, which appears to be sufficient, without taking into account equation 4.3. However, at 25 gain, biasing will be required to keep the minimum CSM output voltage above 10mV, which the device requires. Otherwise, the INA225 appears to meet our requirements.

4.2.4 XBD Targets

The original XBX focused on ATMega chips, some MIPS routers running Linux, as well as the Luminary Micro LM3S chips. We contributed support for the MSP-EXP430FG4618 board which

uses the MSP430FG4618 chip.

The two boards we initially are focusing on are the MSP-EXP430F5529LP [49], which uses an MSP430F5529 chip, and the Tiva-C Launchpad (EK-TM4C123GXL)[65] which uses a TM4C123GH6PM (hereforth referred to as TM4C123).

The MSP430x5xx series is the successor to the MS430x4xx, and the TM4C series is a re-branding of the LM4F, which is the successor to the LM3S chips, after Luminary Micro was purchased by TI. The TM4C123 markedly differs from the TM4C1294 used in the XBH in that it is slower and does not have ethernet connectivity.

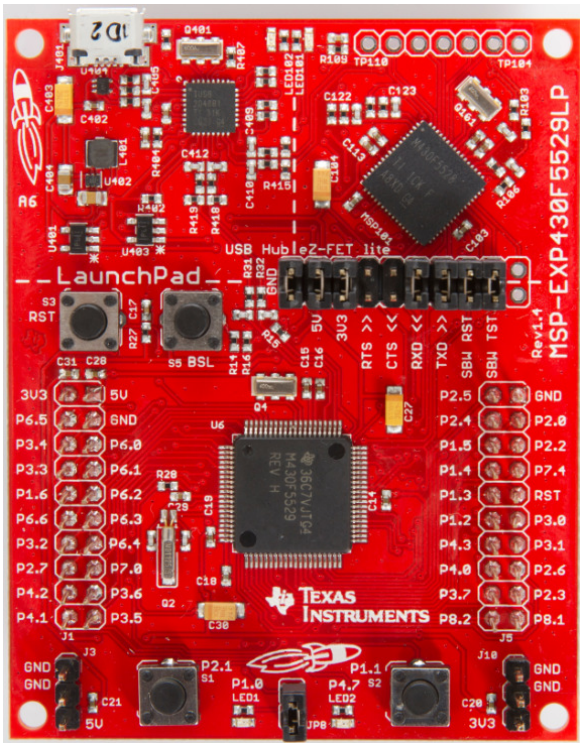

Figure 4.6: MSP-EXP430F5529LP[22]

The MSP430F5529 is a 16-bit chip capable of being clocked at up to 25MHz, and has 10kiB of SRAM and 128kiB of flash. The TM4C123 is a 32-bit chip which can be clocked up to 80MHz, and has 32kiB of SRAM and 128kiB of flash.

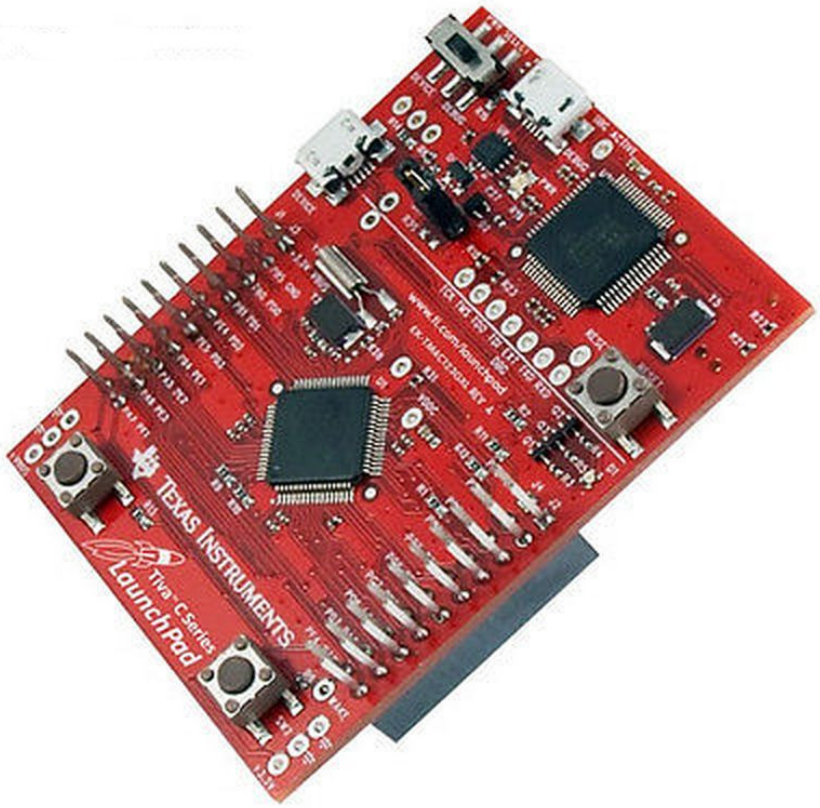

Figure 4.7: EK-TM4C123GXL [69]

We decided to focus on these two boards, as they use chips that are successors to the ones supported by original XBX and were easily available to us. We intend to examine the ATMega devices next.

4.3 Software

4.3.1 XBH Software

The original XBH ran on bare metal using a TCP/IP stack modified from Ulrich Radig's Webserver UVM [68]. We decided to not attempt to port this to the TM4C1294 chip. Instead, the TivaWare [66] software package intended for use with the EK-TM4C1294XL includes Lightweight IP (lwIP) [46] and MicroIP (uIP), along with drivers. We decided to use lwIP as it seemed to be under

more active development, and came with the necessary drivers for the TM4C1294.

Toolchain

We decided to use the GCC (GNU Compiler Collection) compiler as it was free and easily available, and supported the ARM chip on the EK-TM4C1294XL. We decided to use crosstool-ng in order to build GCC, along with newlib, which provides an implementation of the standard C library.

RTOS vs Bare Metal

Initially we intended to use lwIP by itself on bare metal via the event based API, However, this appeared to be more difficult to use compared to the UNIX socket API which lwIP also supports. The socket API requires an RTOS in order to handle TCP/IP in the background while the application executes. We decided to go with the socket API with an RTOS, since an RTOS also allowed benefits such as handling networking, power measurements, and timing measurements without explicit task switching.

During testing, we encountered events where the XBH would be nonresponsive to the XBS, while waiting for the XBD. Due to the use of an RTOS, we were able to easily separate the XBD handling into it's own task, allowing us to handle reset requests in the network task.

RTOS selection

The Tivaware package contains a version of FreeRTOS [24], which is an RTOS licensed under the GNU Public License (GPL), with a linking exception for proprietary drivers). Alternative operating systems with support for the TM4C and/or Cortex M4 are TI-RTOS and NuttX. FreeRTOS was chosen as it came with the board support package with examples and was freely licensed, as well as having support in the lwIP port for the TM4C1294.

The versions of both FreeRTOS and lwIP included in the Tivaware package was a few versions behind. The version of FreeRTOS in Tivaware was 7.1, and the version of lwIP was 1.4.1. We upgraded FreeRTOS to 8.2 and lwIP to the latest from git. Upgrading lwIP required no

changes besides copying the lwIP port to the TM4C1294 provided by TI over, while upgrading FreeRTOS required modifying the lwIP port provided by Tivaware. The benefit of updating to the latest version is minimizing the delta of required changes in the future if a feature is desired. The git version of lwIP provides dual-stack IPv6 and IPv4 which lwIP 1.4.1 does not support.

Licensing

A problem we encountered was that the licensing for much of the Tivaware example code was restrictive, prohibiting use with GPL'ed code and potentially causing problems for redistribution [66]. Some of this example code was needed to operate lwIP with FreeRTOS using the socket API, as it handled initializing the ethernet hardware and GPIOs, notifying the lwIP stack of incoming and transmitted ethernet frames, and setting up the IP address. This code had to be replaced in order to not violate the license terms. We restricted our use of Tivaware code to the driver library and the lwIP port which are freely licensed.

Code Structure

Much of the core XBH code dealing with the XBH $\longleftrightarrow$ XBD protocol and packet formats was reused, with some rewriting to conform to preferred coding conventions, such as using C99 `inttypes.h` instead of custom typedefs.

Some major difference to original XBH are:

- We only support communication to the XBD via I^2C - the original XBH supported I^2C , RS232, and raw Ethernet. The XBD platforms we are interested in all support I^2C
- We only support TCP/IP, as opposed to TCP/IP and UDP/IP with the original XBH. There is no reason to support two different protocols, and TCP handles reliability and in-order delivery.
- Messages are length prefixed. We are using TCP which is stream based and does not have a concept of messages, so we have to handle that ourselves. The previous XBX got away with not adding length prefixes because it assumed the application would only be notified

of a message when the entire message was received, which is probably a fair assumption if the message is under the ethernet frame size and won't get fragmented. However, we opted to be strict and not allow for this assumption. Message payloads are also in binary format instead of text, to remove the need to decode hex on the XBH.

- Power measurements are to be sampled and streamed to the XBS during measurement in realtime, after packing multiple measurements into a message, as large amounts of data cannot be stored on the XBH. This is however tentative, and we may only send a summary to the XBS from the XBH, i.e. min, max, and average current.

We did not foresee a need to replace the RTOS in the future, so we did not create an abstraction layer for it. However, we did write an abstraction layer to talk to the hardware, in order to replace the XBH easily if need be. Our HAL layer handled hardware initialization, I^2C communication with the XBD, writing to the UART for debugging, ethernet, interrupt prioritization, power measurement hardware, and watchdog reset.

Decoupling the hardware handling from the rest of the XBH software turned out to be useful even when the underlying hardware platform wasn't changed, as I/O pins were reassigned and hardware was reconfigured multiple times.

Tasks

The XBH software runs multiple tasks, which are managed by FreeRTOS and assigned separate fixed-sized memory stacks. The tasks, in order of priority, from lowest to highest, are

1. lwIP TCP/IP
2. XBH Server
2. XBH command execution and XBD communication (same priority as XBH server)
3. Ethernet Receive/Transmit - sends transmit and receive descriptors to lwIP.
4. Power Measurement - woken up periodically by timer interrupt to perform measurements and enqueueing them to the XBH server task.

Timeslicing is enabled, which makes FreeRTOS alternate between tasks of equal priority without waiting for one to block.

Execution time measurement is solely handled by interrupt handlers with the timing values saved to global variables, and thus do not need a dedicated task.

The XBH Server task handles packets received from the XBS. It queues commands onto the XBH command/XBD communication task for execution, and listens for any reset command from the XBS while that thread executes.

The XBH command/XBD communication task issues commands to the XBD and blocks waiting for the XBD to complete the ordered operation and respond.

The power measurement was separated into another task as we initially planned to use the INA219 which used the I^2C bus, which does not immediately produce a result when queried and thus is not suitable for direct querying by an interrupt handler. The I^2C communication implementation polls the registers to determine if a message has been sent, for reasons of simplicity, instead of having the task sleep and woken up later by an interrupt indicating the I^2C task has been completed. Using the FreeRTOS facilities is not an option, as it only has a precision less than or equal to the tick period, which is 1ms by default.

Instead, the power measurement task is woken by a timer interrupt on a consistent interval, and as it is the highest priority task, will preempt other FreeRTOS tasks. This task will do the polling of I^2C and enqueue the data to the XBH server task for sending to the XBS.

This will not be necessary with the analog current shunt monitor, which can read the value from the ADC register immediately and enqueue the value to be sent directly into the XBH Server.

Interrupts

The TM4C1294 has 8 interrupt priority levels, 0-7. The meanings are inverted, with priority 0 being the highest priority and priority 7 being the lowest, as with other Cortex-M processors. The interrupt priorities are configured as follows, from highest to lowest priority.

0. Unused

1. Timer Wraparound
2. Timer Capture
3. Max FreeRTOS SysCall Priority
3. Power Sample Timer
4. Watchdog
5. Unused
6. Unused
7. FreeRTOS kernel

The timer wraparound interrupt is triggered when the 16-bit timer used for timer capture wraps. The TM4C1294 uses is limited to 16-bit when used in edge-time mode to capture the time that the start and stop signals are asserted. At 120MHz, a 16 bit timer is insufficient. Using equation 4.5, we get 546μs before timer wrap which is shorter than the likely runtime.

$$\frac{1}{f} \times 2^{n_{\text{bits}}} \tag{4.5}$$

f is 120MHz, the clock rate we are running the board at, and n_{bits} is the bits used to store the tick count, which is 16.

To work around this, a periodic timer that cycles at the same rate at higher priority is used to count the number of times the edge timer has wrapped around [61]. The interrupt handler for this will increment the count of wraps, or if it has preempted the handler for the edge timer, will defer that incrementing for after the completion of the edge timer interrupt, which records the computed time since timer start into a 64-bit value. 64-bit is necessary, since by using equation 4.5 with 32 bits, we get a maximum time of 35.8 seconds, which is also shorter than the potential longest time of execution.

The power sampling timer periodically issues an interrupt to record the current using a timer synchronized to the wrap and capture timers as previously mentioned in 4.3.1.

The maximum syscall priority determines the minimum priority of the interrupts that will be left enabled inside critical sections. Interrupt handlers that have a priority higher than this cannot be preempted by FreeRTOS and cannot call FreeRTOS API functions.

The FreeRTOS kernel interrupts performs scheduling of FreeRTOS tasks and initiates context switches when no other interrupts are pending and all interrupts are unmasked.

4.3.2 XBD Software

The XBD software remained largely identical to that in original XBX. Most of the changes were limited to refactoring or adding AEAD support.

The substantially more comprehensive self-test suite to verify correctness of the primitive implementations from SUPERCOP replaced the simplified version found in XBX, as we wanted to retain this aspect of SUPERCOP and did not want to write our own test for AEAD.

Unfortunately this leads to substantially longer execution times and precludes operation with smaller devices especially for AEAD, which was probably the original XBX team's motivation for writing a simplified check.

For AEAD, the test requires two large buffers, one for input ciphertext/auth data and one for output, and multiple smaller buffers for keys, secret number, as opposed to hash functions, which only need a large buffer for the input and a substantially smaller buffer for the hash, with the largest output being 512 bits (64 bytes). In addition AEAD primitives require multiple smaller buffers for various parameters defined in the CAESAR interface, each with padding in order to detect overruns. The buffers and limits we assigned for AEAD are:

- Message
- Associated Data (Associated Data+Message must be less than 2048 bytes)
- Ciphertext, which is the size of Message+Associated Data plus additional bytes for authentication. Additional bytes is limited to 128 bytes.

- Secret Bytes (64 bytes)
- Public Bytes (64 bytes)
- Key Bytes (64 bytes)

Some primitives had unreasonable sizes for parameters such as Trivia-128v1 which specified slightly over 512kiB of additional bytes to the output, which is well over the total memory of most microcontrollers We decided on these limits since we deemed them the maximum reasonable for an embedded platform, although we leaned towards larger sizes unlikely to be found in an embedded environment in order to include more primitives. Embedded devices might also use larger than ideal primitives for interoperability reasons.

Other aspects of the XBD code were also modified to make most of the XBD code as operation-agnostic as possible, except for the test code. References to hash functions were removed in code that was generic to all operations.

In order to support a new operation already supported by SUPERCOP, changes are limited to modifications to the script used to generate the tests in order to allocate memory in order to fit our more restricted framework and code to unpack message payloads to set the appropriate buffer pointers.

Primitive Implementation Testing

As with the original XBX, primitive implementations are tested for correctness. We reused the SUPERCOP primitive implementation verification code. This code has options for large and small tests. As we are operating in a resource constrained environment, we only use the small tests, that handle lengths up to 128 bytes.

For hash functions, correct results are verified by mixing in the hash function output for various sizes into an instance of Salsa20 [16], instead of chaining the outputs as with original XBD. The output of this is later compared with a known value by the XBS. Deterministic results are also checked for, as well as if the implementation can handle the output being placed in the same buffer as the input and buffers remain within bounds.

For AEAD functions, the ciphertext is fed into the Salsa20 function, and is decrypted with the plaintext and associated data also fed into the Salsa20 function for checking by the XBS. In addition, the decrypted message and associated data is compared to the original message and associated data. In addition to the correctness verifications, the implementation is checked to see if the output is deterministic, if buffers remain within bounds, overlap of input and output buffers is handled if specified by the implementation, and if forgeries are detected.

The XBD to XBH (and vice-versa) communication code for sending and receiving large data over multiple I^2C packets was refactored to reuse the same code over multiple commands.

4.3.3 MSP430F5529 Port

Much of our previous work on original XBX for the MSP430FG4618 was reused, and the work was mostly altering pinouts and adapting to the new MSP430-GCC [30] toolchain, which had substantial differences to the older MSPGCC toolchain [31].

We decided to clock the MSP430F5529 at 16MHz, matching the clock rate on the AVR platforms in original XBX. The clock initialization on the MSP430F5529 is more complicated, and we used code from the Energia [19] project to handle this. The digitally controlled oscillator (DCO) in the MSP430F5529 does not run at exactly 16MHz, but mixes two frequencies to get an average at the specified frequency. This causes jitter and has implications for accurate measurement when cycles are measured externally, however the impact is likely to be negligible for the purpose of comparing algorithms. This is a motivation for using cycle counters where available if more precise measurements are needed, however.

In order to support LTO, we had to mark the interrupt handler for resetting the XBD explicitly as used using GCC attributes. Attributes are a compiler-specific C extension used to apply a property to a C function or variable, that is not in standard C. Without this attribute, LTO would optimize out the interrupt handler.

4.3.4 TM4C123 (Tiva C) Port

Much of the work assembling the toolchain, sorting out platform quirks, and sorting through licensing on the TM4C1294 XBH applied to the TM4C123.

Open On-Chip Debugger (OpenOCD) is a tool used to program and debug various chips and boards, including the Tiva-C boards used as the XBH and an XBD [50]. Currently, it does not support multiple Texas Instruments In-Circuit Debug Interface (TI-ICDI) attached to the same PC. We wrote a patch [42] in order to distinguish different TI-ICDI devices by serial number.

We encountered some issues programming the primitive implementations into the XBD. The IHEX files containing the code to be downloaded to the XBD contained holes, which are unused memory areas that are not specified in the file, making the defined memory locations noncontiguous. The original XBS software did not properly take these into account, and would load nonadjacent blocks of code adjacently. This was fixed in the XBS rewrite.

4.3.5 XBS Software

The XBS was rewritten in Python 3, away from a mix of shell script and Perl that it was originally written in. The primary reason for this was because we wanted to add the ability to resume runs if a failure occurred, and we felt it was easier to work with a full object-oriented scripting language with libraries in order to add this functionality. We were also more familiar with Python then we were with Perl.

Some functionality was not re-implemented.

- Verification using known-answer test (KAT) files. We considered the tests in SUPERCOP sufficient.
- Loading of formats other than IHEX is not supported. This functionality did not appear to be used.

Querying results

Results can be examined by opening the generated SQLite database (`data.db`) in a SQLite browser application, such as DB Browser for SQLite [17]. For analysis, the SQLAlchemy objects can be manipulated directly in an IPython [39] notebook. IPython is an interactive python environment, with an interface similar to Maple or SageMath with "notebooks." The schema is listed in appendix A.

Persisting state

In order to persist the state of the runs and the actual results, we decided to store this information into a SQLite [56] database. We used SQLAlchemy [55] to simplify the task of persisting the state of the XBS. SQLAlchemy is an object-relational mapper (ORM) that maps Python objects to SQL tables and rows, without having to directly write SQL. We used this library as it is significantly simpler than using SQL directly.

Protocol deviations

We broke compatibility with original XBX's XBH $\longleftrightarrow$ XBD protocol, as previously mentioned in 4.3.1. We prefix messages with the length in ASCII, as TCP is a stream based protocol and has no concept of messages. The remainder of the message after the command type is in binary to ease parsing. Python makes handling binary data very easy, and removing the need to convert to and from ASCII hex in multiple places simplifies the code.

Typical Run

First, the configuration in `config.ini` is edited. This specifies whitelists and blacklists of implementations, the target operation, the target platform, paths, parameters, dependencies, etc. We assume that resuming a failed run will not involve changing the target operation and platform. Logging levels can be configured in `logging.ini`, and uses Python's built-in logging framework.

Then `compile.py` is then run. This creates the file `data.db`, used to store results, configuration, and other detailed information. `config.ini`, a platform-specific config file, and various blacklist files are all parsed.

If a whitelist exists in `config.ini`, only the implementations in this list will be built (and later run), otherwise all implementations available in the specified operation and primitives are built, excluding those specified in the blacklist. There is a blacklist in config.ini, specific to the run, as well as a global blacklist for implementations that are broken, and platform specific blacklists for implementations that are broken on specific platforms. Hashes of all implementation and platform code are taken to ensure exact replication of results, if required.

Once the configuration is assembled, the compile script generates the makefiles and header files for each implementation and builds the code for the XBD for each implementation × {compiler, flag}. This proceeds quickly, as the script spawns as many subprocesses as there are cores to execute builds. We decided to run multiple makefiles simultaneously as opposed to running GNU make with the `-j` flag, which does parallelization at the makefile level, as even with `-j`, the makefiles would stall trying to build something that would not build, e.g. attempting to compile x86 assembly to ARM. Running separate makefiles in parallel builds more quickly, by allowing stalls to not block other tasks. The `size` command is run as in original XBX on the compiled binaries, and the values saved in the database. We decided to limit the maximum static memory allocations to 3/4ths the total available RAM, leaving the remainder for stack growth, for both the Tiva-C and the MSP430 XBDs in the platform configuration.

As the builds use GNU `make`, rebuilds do not require recompiling everything from scratch, only the code that changed or that was interrupted. Original XBX did not use makefiles. It copied all source files to a build directory, and compiled the source files together all at once, which meant all files are always rebuilt even if nothing changed.

Unlike XBX, and like SUPERCOP, primitive implementations can depend on other implementations. However, the relations have to be explicitly defined, as there are multiple metrics for what a good implementation is, unlike SUPERCOP.

Once a build succeeds, the `execute.py` script is run. The configuration used from the last

build is loaded from `data.db`, if `config.ini` has not changed between the last build and calling `execute.py`. Behavior is undefined if `config.ini` or blacklists or code are modified between calls of `compile.py` and `execute.py`. Calibration is run as in original XBX.

For each successfully compiled implementation, the binary is loaded into the XBD and tests are run, as mentioned in section 4.3.2. These tests can be run multiple times if configured for such, however they take a considerable amount of time. Afterwards, the XBS generates data to send to the XBD for performance testing, as specified in the config file. A python object defining the parameters, how to generate them, and how to handle what is returned by the XBD is required to support an operation, and as previously mentioned, we have hash functions and AEAD implemented.

In the case of hashing, we simply generate random data of the specified lengths and send them to the XBD, discarding the result, while timing and measuring execution, as we don't have a known answer for hashing random data, while for AEAD, we encrypt the data, get the ciphertext from the XBD, and send it back for decryption, as well as decryption of a tampered ciphertext. We log time (and later power) measurements for all these results. We do an additional verification to see if the decryption matches the unencrypted plaintext, as this is essentially free without additional computations as the XBS has this information available. All of this information is logged for later analysis.

4.3.6 Problems Encountered

We encountered several issues, some related to the incompleteness of the example code, and others:

- The example linker script provided by TivaWare did not contain the `.ARM.exidx` section used to handle exceptions with 64-bit divides. This had to be added to the linker script. 64-bit types are used in our timer code.
- The example startup code (replacing `crt0` on other platforms) did not properly align

the stack, causing breakage when attempting to use a self-written `printf` implementation to write to a serial console. ARM platforms require 8-byte stack alignment, as some instructions work directly on 8-byte types.

- Due to some protocol changes, some buffer sizes needed to be increased in order to fit the additional data.
- The I^2C hardware in the XBH did not contain internal pull-up resistors, so external ones had to be added in order for communications with the XBD to work.
- While debugging, ethernet hardware interrupts repeatedly disrupts stepping though code. Disabling interrupts via OpenOCD commands alleviates this somewhat, but breaks calls to FreeRTOS functions causing a crash.
- In order to support LTO on the XBD, we had to set GCC attributes to disable inlining of the stack painting function, as it depends on being in its own stack frame.
- The default linker script for the MSP430F5529, used in the XBD, leaves read-write sections in the same section as read-only data, causing the entire section to be marked read-write [43]. This caused incorrect size calculations, as the section would be considered as using RAM due to the read-write nature.
- Assertians and `printf` calls in some of the algopack implementations for AEAD broke compilation.
- The verification phase takes an extremely long time. To resolve this, either primitive implementations for embedded need to be found and/or the testing code needs to be simplified.

Chapter 5: Status and conclusion

Currently, we have reached feature parity with XBX, other than platform support. AEAD functions and hash functions are supported benchmarking targets. The remaining near-term work to be published within a month is:

- Integrate the power measurement hardware
- Perform a full benchmarking run on all AEAD and hash algorithms that have implementations that can run
- Extend platform support to AVR
- Documentation.

One of our longer term goals is to obtain implementations optimized for embedded, as many of the AEAD primitives in the SUPERCOP algopack lack an implementations other than the reference implementation, which often does not comply with section 3.3.

Chapter 6: SHA-3

In 2007, NIST announced the SHA-3 competition [54], in order to find a replacement for the SHA-2 hash function [67], after vulnerabilities were discovered in the related SHA-1 algorithm by Wang, et al.[70]. A cryptographic hash function irreversibly compresses down a message into a smaller fixed-length value. It should have the properties of

- Pre-image resistance - difficult to find a message m where $h = H(m)$
- Second pre-image resistance - difficult to find a message m_2 where $H(m_1) = H(m_2)$, where m_1 is fixed.
- It should be difficult to find any two m_1 and m_2 where $H(m_1) = H(m_2)$

The goal of the competition was to find a successor to SHA-3 that was more secure than SHA-2, yet was as performant or better on a large variety of platforms. In order to compare the performance of the SHA-3 candidates, implementations in hardware and software were compared.

6.1 Hardware

To support the performance evaluation on FPGA hardware, the CERG group at GMU developed the ATHENa tool for automated benchmarking of FPGA implementations, which was inspired by SUPERCOP[28]. This tool aims to batch process the synthesis, implementation, and timing analysis steps of compiling FPGA designs. This is performed across multiple devices from multiple vendors, with options searched to best meet the desired characteristics (low area, throughput, throughput/area).

As part of this effort, we developed a database to collect, store, and present the collected results from using this tool, as well as those manually entered from outside contributors [11].

The database also collected outside results for ASICs. This database consisted of a PHP/PostgreSQL application, which took result files generated by the ATHENa tool, or hand-entering of results.

CERG developed full speed FPGA implementations for all Round 2 [27] and Round 3 [35] candidates, all with a unified interface. This effort used designers of roughly equal skill in order to present a fair comparison between the algorithms and not the implementations, across multiple devices. These efforts concluded that Keccak had the best performance in terms of throughput to area ratio, and absolute throughput on most FPGAs.

6.2 Lightweight SHA-3

For embedded applications, and to provide a realistic scenario where designs would be given a set of constraints to obey, CERG developed lightweight implementations of the Round 2 and Round 3 candidates. We contributed an implementation of JH, which was a Round 2 and Round 3 candidate.

Each implementation was given a budget of 400-600 Spartan-3 slices and, unlike other low-area efforts [44][45], was given 1 block RAM to work with. The design was targeted for the Spartan-3, however results were generated for the Spartan-6, Virtex-5, Virtex-6, and Cyclone II FPGAs.

6.2.1 JH

The JH hash function's uses bijective function consists of a grouping step, 35.5 (Round 2) or 42 (Round 3) instances of the round function, and a degrouping step. Each round consists of a row of 4-bit S-boxes pairs, selectable by bits in the round constant, a linear transform (omitted from the last half-round for the Round 2 version), and a permutation (also omitted from the half-round). The compression function consists of message blocks xored into the state before and after the state is passed into bijective function. The value $H^{(0)}$ is used to initialize the state for every new message.

Our implementation

Our implementation of JH (Fig. 6.2) stores the state and constants in BRAM. Two independent 32x8 Distributed RAMs store the state of the round constant generator. The BRAM is used as two independent memories to simplify the control logic and ease synchronization with the round constant generator. During initialization, which takes 35 cycles, the location of the state in BRAM is initialized with the precomputed starting value of $H^{(0)}$, from another address in the same BRAM. Grouping and de-grouping take advantage of the dual port memory and read two addresses from the BRAM simultaneously, retaining 4 bits from each address in registers and discarding the rest. This is repeated 4 times to write a full 32-bit value back into the BRAM and takes 160 cycles. The core round function is 32 times vertically folded and contains a pipelined permutation function. It needs a total of 34 cycles per round. Difficulties in creating this implementation were the memory access delays and the nonconsecutive read and write addresses. The tweak for round-3 of the SHA-3 competition increases the number of rounds to 42. This version of JH is called JH42.

[40]

6.2.2 Results

Table 6.1 shows the implementation results, while table 6.2 shows the throughput results from our JH implementations, and that of the other algorithms in that were part of the lightweight comparison. Our JH implementation had a fairly low area, and had the smallest control unit to datapath ratio, but also had somewhat low performance. The best performer was BLAKE [2] in terms of absolute throughput and throughput/area ratio, however our JH implementation was the lowest area, by a small margin. BLAKE appears to have excellent scalability on lightweight implementations, despite falling behind Keccak on high-end implementations.

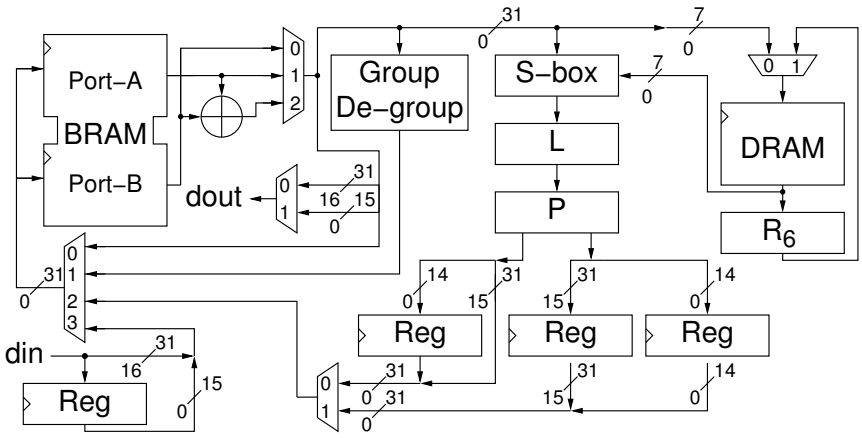

Figure 6.1: JH [40]

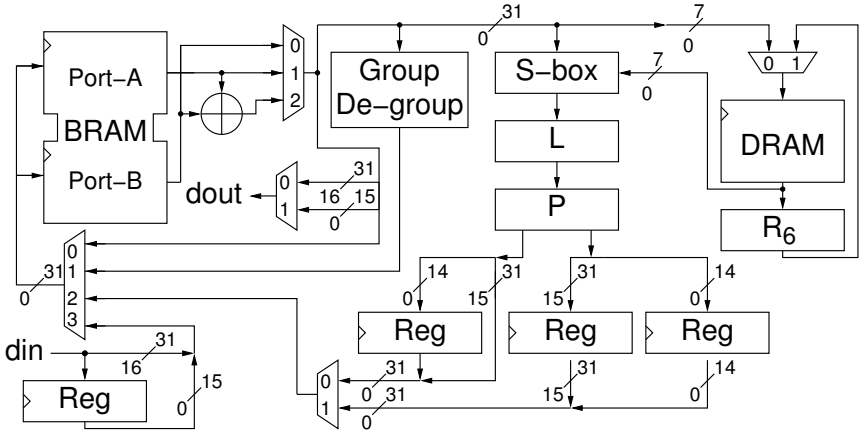

Figure 6.2: Block Diagram of JH [40]

Table 6.1: Implementation results of implementations of SHA-3 candidates [40]

Algorithm	Xilinx xc3s50-5			Xilinx xc6slx4csg-3			Xilinx xc5vlx20-2			Altera ep2c5f256c6		
	Area (slices)	Block RAMs	Maximum Delay (ns) T	Area (slices)	Block RAMs	Maximum Delay (ns) T	Area (slices)	Block RAMs	Maximum Delay (ns) T	Area (LEs)	Memory Bits	Maximum Delay (ns) T
BLAKE-256	545	1	8.42	139	1	6.47	212	1	4.30	1,365	2,048	8.70
Grøstl	537	1	6.95	163	1	5.44	234	1	3.77	1,026	2,560	5.86
JH42	428	1	9.74	142	1	8.16	176	1	5.28	702	8,704	8.59
Keccak	582	1	8.30	142	1	5.91	196	1	3.74	996	8,192	5.48
Skein	491	1	10.68	227	1	8.07	215	1	5.74	930	4,096	9.89
BLAKE-32	527	1	7.38	138	1	7.23	238	1	4.37	1,262	2,048	8.71
BMW	561	1	9.99	183	1	7.15	233	1	5.16	1,104	8,192	9.45
CubeHash	434	1	12.58	131	1	8.11	231	1	6.05	2,761	16,384	9.18
ECHO	508	1	11.33	155	1	9.72	232	1	5.34	1,069	16,512	10.14
Fugue	451	1	13.48	269	1	12.84	209	1	6.43	940	16,384	7.81
Grøstl-0	517	1	6.93	163	1	5.23	232	1	3.61	1,020	2,560	5.93
Hamsi	533	1	9.97	162	1	12.83	208	1	5.28	687	10,240	8.87
JH	482	1	9.99	180	1	8.35	161	1	4.97	702	8,704	8.59
Luffa	474	1	10.17	107	1	6.86	176	1	5.08	946	8,192	7.66
Shabal	502	1	10.17	165	1	7.66	231	1	4.86	2,093	1,760	9.16
Shavite-3	501	1	7.60	120	1	5.24	136	1	3.37	471	16,384	7.01

6.3 Software

The role of software benchmarking and comparison was taken up by SUPERCOP and XBX, which have been described in detail in sections 3.4 and 3.5. BLAKE and Skein both perform extremely well in software on high-end CPUs, with BLAKE also scaling down extremely well onto most microcontroller platforms, including on the MSP430 platform.

6.4 Competition Conclusion

Ultimately, NIST chose Keccak as the winner of the SHA-3 competition [8]. Despite the performance of BLAKE and Skein [21] in software, and BLAKE on low-area designs, NIST felt that Keccak better complemented the existing deployment of SHA-2. SHA-2 has remained unbroken contrary to expectations, and thus is likely to remain in deployment for quite some time. NIST

Table 6.2: Throughput results of lightweight implementations of SHA-3 candidates. First are Round-3 results followed by Round-2 results. [40]

Message / Algorithm	Xilinx xc3s50-5				Xilinx xc6slx4csg225-3			
	Long		Short		Long		Short	
	Throughput (Mbps)	TP/Area (Mbps/slice)	Throughput (Mbps)	TP/Area (Mbps/slice)	Throughput (Mbps)	TP/Area (Mbps/slice)	Throughput (Mbps)	TP/Area (Mbps/slice)
BLAKE-256	173.8	0.32	164.8	0.302	226.0	1.63	214.4	1.542
Grøstl	134.6	0.25	68.1	0.127	171.9	1.05	87.0	0.534
JH-42	28.5	0.07	28.2	0.066	34.0	0.24	33.6	0.237
Keccak	34.8	0.06	34.7	0.060	48.9	0.34	48.6	0.343
Skein	19.7	0.04	9.9	0.020	26.0	0.11	13.0	0.057
BLAKE-32	261.0	0.50	243.6	0.462	266.4	1.93	248.6	1.802
BMW	67.3	0.12	33.7	0.060	93.9	0.51	47.1	0.257
CubeHash	21.6	0.05	2.0	0.005	33.5	0.26	3.1	0.024
ECHO	53.3	0.10	52.5	0.103	62.1	0.40	61.2	0.395
Fugue	37.7	0.08	2.2	0.005	39.6	0.15	2.3	0.009
Grøstl-0	135.1	0.26	68.4	0.132	179.0	1.10	90.6	0.556
Hamsi	42.2	0.08	22.4	0.042	32.8	0.20	17.4	0.108
JH	31.2	0.06	30.9	0.064	37.4	0.21	37.0	0.205
Luffa	40.5	0.09	19.8	0.042	60.0	0.56	29.4	0.275
Shabal	524.7	1.05	149.9	0.299	687.5	4.17	196.4	1.190
Shavite-3	86.8	0.17	83.1	0.166	126.0	1.05	120.6	1.005

Message / Algorithm	Xilinx xc5vlx20-2				Altera ep2c5f256c6			
	Long		Short		Long		Short	
	TP (Mbps)	(Mbps /slice)	TP (Mbps)	(Mbps /slice)	TP (Mbps)	(Mbps /LE)	TP (Mbps)	(Mbps /LE)
BLAKE-256	340.4	1.61	322.8	1.523	168.1	0.12	159.4	0.117
Grøstl	248.5	1.06	125.7	0.537	159.7	0.16	80.8	0.079
JH-42	52.5	0.30	52.0	0.295	32.3	0.05	31.4	0.046
Keccak	77.2	0.39	76.8	0.392	52.7	0.05	52.5	0.053
Skein	36.6	0.17	18.3	0.085	21.2	0.02	10.6	0.011
BLAKE-32	440.8	1.85	411.4	1.728	220.9	0.18	206.2	0.163
BMW	130.3	0.56	65.3	0.280	71.1	0.06	35.6	0.032
CubeHash	44.8	0.19	4.1	0.018	29.6	0.01	2.7	0.001
ECHO	113.0	0.49	111.5	0.481	59.5	0.06	58.7	0.055
Fugue	79.0	0.38	4.6	0.022	65.0	0.07	3.8	0.004
Grøstl-0	259.6	1.12	131.4	0.566	157.9	0.15	79.9	0.078
Hamsi	79.7	0.38	42.4	0.204	47.5	0.07	25.2	0.037
JH	62.8	0.39	62.0	0.385	36.3	0.05	35.9	0.051
Luffa	81.1	0.46	39.7	0.225	53.8	0.06	26.3	0.028
Shabal	1097.8	4.75	313.7	1.358	582.5	0.28	166.4	0.080
Shavite-3	196.1	1.44	187.6	1.380	94.1	0.20	90.1	0.191

was concerned that the shared Add-Rotate-XOR (ARX) design of Skein and BLAKE could allow future breaks in SHA-2 to also apply to these algorithms. In addition, the hardware-oriented nature of Keccak with different performance strengths compared to SHA-2 was thought to be an advantage.

Chapter 7: CAESAR

The CAESAR competition seeks an authenticated cipher to replace the AES-GCM algorithm with one that is more performant yet still secure. An authenticated cipher is a cipher that also provides assurances of message integrity. Many authenticated ciphers also support associated data, which is unencrypted data which has integrity checks.

As with SHA-3, part of this competition involves performance evaluations and comparisons. For hardware evaluations, we have extended the ATHENa database to support the entry of AEAD results, while for software, SUPERCOP supports the benchmarking of AEAD algorithms. XBX originally did not support AEAD, however we have also added support for this as described in chapter 4.

7.1 Status

As of this writing, CAESAR Round 1 has just ended, with the following algorithms moving on to Round 2 [15]:

- ACORN
- AEGIS
- AES-COPA
- AES-JAMBU
- AES-OTR
- AEZ
- Ascon

- CLOC and SILC
- Deoxys
- ELmD
- HS1-SIV
- ICEPOLE
- Joltik
- Ketje
- Keyak
- Minalpher
- MORUS
- NORX
- OCB
- OMD
- PAEQ
- π-Cipher
- POET
- PRIMATES
- SCREAM without iSCREAM
- SHELL
- STRIBOB
- Tiaoxin

- TriviA-ck

Our overhaul of XBX to add AEAD support currently is capable of running AEAD algorithms, however, a public release will not be done until the outstanding issues listed in section 5 are complete.

Chapter 8: Conclusion

Cryptographic standards are now being decided by the results of competitions. These competitions require performance data on submitted candidates in addition to the security analysis in order to make a decision.

Embedded devices are becoming increasingly important, and thus so are lightweight implementations of candidates targeted towards embedded devices. Candidates must be evaluated separately for embedded and full-speed targets, as algorithms and implementations are not necessarily scalable.

We contributed to this effort to evaluate SHA-3 candidates on hardware by implementing the ATHENa database of results, and by implementing the algorithm JH as part of an evaluation of SHA-3 candidates on embedded. Our implementation of JH was the smallest of the five SHA-3 finalists, however the candidate BLAKE was the fastest.

For software, we ported XBX to the MSP430, and contributed results for that platform, which concluded that the candidate BLAKE was the best performing in terms of both performance and area metrics on MSP430.

As part of the CAESAR competition for hardware evaluations, we have extended the ATHENa database in order to support AEAD results.

For CAESAR embedded software benchmarking, we have overhauled XBX to support AEAD ciphers, and have concrete plans to add power measurement support. Prior to this point, comprehensive automated power benchmarks on software have been missing. We are at the point where we are one or two months away from being largely feature complete and ready to release results.

Appendix A: Extended XBX Database Schema

```
CREATE TABLE operation (
        name VARCHAR NOT NULL,
        operation_str VARCHAR,
        PRIMARY KEY (name)
);
CREATE TABLE platform (
        hash VARCHAR NOT NULL,
        name VARCHAR NOT NULL,
        clock_hz INTEGER,
        pagesize INTEGER,
        path VARCHAR,
        tmpl_path VARCHAR,
        PRIMARY KEY (hash)
);
CREATE TABLE primitive (
        name VARCHAR NOT NULL,
        operation_name VARCHAR NOT NULL,
        checksumsmall VARCHAR,
        checksumbig VARCHAR,
        path VARCHAR,
        PRIMARY KEY (operation_name, name),
        FOREIGN KEY(operation_name) REFERENCES operation (name)
);
```

```
CREATE TABLE compiler (
        platform_hash VARCHAR NOT NULL,
        idx INTEGER NOT NULL,
        cc_version VARCHAR,
        cxx_version VARCHAR,
        cc_version_full VARCHAR,
        cxx_version_full VARCHAR,
        cc VARCHAR,
        cxx VARCHAR,
        PRIMARY KEY (platform_hash, idx),
        FOREIGN KEY(platform_hash) REFERENCES platform (hash)
);
CREATE TABLE config (
        hash VARCHAR NOT NULL,
        config_path VARCHAR NOT NULL,
        platforms_path VARCHAR NOT NULL,
        algopack_path VARCHAR NOT NULL,
        embedded_path VARCHAR NOT NULL,
        work_path VARCHAR NOT NULL,
        data_path VARCHAR NOT NULL,
        impl_conf_path VARCHAR NOT NULL,
        xbh_addr VARCHAR,
        xbh_port INTEGER,
        platform_hash VARCHAR,
        operation_name VARCHAR,
```

```
        rerun BOOLEAN,
        drift_measurements INTEGER,
        checksum_tests INTEGER,
        operation_params TEXT,
        xbh_timeout INTEGER,
        exec_runs INTEGER,
        blacklist TEXT,
        blacklist_regex VARCHAR,
        whitelist TEXT,
        enforce_bwlist_checksums BOOLEAN,
        one_compiler BOOLEAN,
        parallel_build BOOLEAN,
        PRIMARY KEY (hash),
        FOREIGN KEY(platform_hash) REFERENCES platform (hash),
        FOREIGN KEY(operation_name) REFERENCES operation (name),
        CHECK (rerun IN (0, 1)),
        CHECK (enforce_bwlist_checksums IN (0, 1)),
        CHECK (one_compiler IN (0, 1)),
        CHECK (parallel_build IN (0, 1))
);
CREATE TABLE implementation (
        hash VARCHAR NOT NULL,
        name VARCHAR NOT NULL,
        operation_name VARCHAR NOT NULL,
        primitive_name VARCHAR NOT NULL,
```

```
        path VARCHAR,
        macros TEXT,
        PRIMARY KEY (hash),
        FOREIGN KEY(primitive_name, operation_name) REFERENCES
            primitive (name, operation_name)
);
CREATE TABLE build_session (
        id INTEGER NOT NULL,
        host VARCHAR,
        timestamp DATETIME,
        xbx_version VARCHAR,
        parallel BOOLEAN,
        config_hash VARCHAR NOT NULL,
        PRIMARY KEY (id),
        CHECK (parallel IN (0, 1)),
        FOREIGN KEY(config_hash) REFERENCES config (hash)
);
CREATE TABLE build (
        id INTEGER NOT NULL,
        build_session_id INTEGER NOT NULL,
        platform_hash VARCHAR NOT NULL,
        compiler_idx INTEGER NOT NULL,
        operation_name VARCHAR NOT NULL,
        primitive_name VARCHAR NOT NULL,
        implementation_hash VARCHAR NOT NULL,
```

```
    work_path VARCHAR,
    exe_path VARCHAR,
    hex_path VARCHAR,
    parallel_make BOOLEAN,
    text INTEGER,
    data INTEGER,
    bss INTEGER,
    timestamp DATETIME,
    hex_checksum VARCHAR,
    rebuilt BOOLEAN,
    build_ok BOOLEAN,
    log TEXT,
    PRIMARY KEY (id),
    UNIQUE (build_session_id, platform_hash, compiler_idx,
        operation_name, primitive_name, implementation_hash),
    FOREIGN KEY(build_session_id) REFERENCES build_session (id),
    FOREIGN KEY(platform_hash) REFERENCES platform (hash),
    FOREIGN KEY(primitive_name, operation_name) REFERENCES
        primitive (name, operation_name),
    FOREIGN KEY(operation_name) REFERENCES operation (name),
    FOREIGN KEY(implementation_hash) REFERENCES implementation (
        hash),
    FOREIGN KEY(platform_hash, compiler_idx) REFERENCES compiler
         (platform_hash, idx),
    CHECK (parallel_make IN (0, 1)),
```

```
        CHECK (rebuilt IN (0, 1)),
        CHECK (build_ok IN (0, 1))
);
CREATE TABLE run_session (
        id INTEGER NOT NULL,
        host VARCHAR,
        timestamp DATETIME,
        xbx_version VARCHAR,
        xbh_rev VARCHAR,
        xbh_mac VARCHAR,
        build_session_id INTEGER NOT NULL,
        config_hash VARCHAR NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY(build_session_id) REFERENCES build_session (id),
        FOREIGN KEY(config_hash) REFERENCES config (hash)
);
CREATE TABLE config_libsupercop_impl_join (
        implementation_hash VARCHAR NOT NULL,
        config_hash VARCHAR NOT NULL,
        PRIMARY KEY (implementation_hash, config_hash),
        FOREIGN KEY(implementation_hash) REFERENCES implementation (
            hash) ON DELETE CASCADE,
        FOREIGN KEY(config_hash) REFERENCES config (hash) ON DELETE
            CASCADE
);
```

```
CREATE TABLE config_impl_dep_assoc (
        config_hash VARCHAR NOT NULL,
        implementation_hash VARCHAR NOT NULL,
        PRIMARY KEY (config_hash, implementation_hash),
        FOREIGN KEY(config_hash) REFERENCES config (hash) ON DELETE
            CASCADE,
        FOREIGN KEY(implementation_hash) REFERENCES implementation (
            hash) ON DELETE CASCADE
);
CREATE TABLE config_impl_dep_join (
        config_hash VARCHAR NOT NULL,
        dependent_impl_hash VARCHAR NOT NULL,
        dependency_impl_hash VARCHAR NOT NULL,
        PRIMARY KEY (config_hash, dependent_impl_hash,
            dependency_impl_hash),
        FOREIGN KEY(config_hash) REFERENCES config (hash) ON DELETE
            CASCADE,
        FOREIGN KEY(dependent_impl_hash) REFERENCES implementation (
            hash) ON DELETE CASCADE,
        FOREIGN KEY(dependency_impl_hash) REFERENCES implementation
            (hash) ON DELETE CASCADE
);
CREATE TABLE drift_measurement (
        id INTEGER NOT NULL,
        abs_error INTEGER,
```

```
        rel_error INTEGER,
        cycles INTEGER,
        measured_cycles INTEGER,
        run_session_id INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(run_session_id) REFERENCES run_session (id)
);
CREATE TABLE build_exec (
        id INTEGER NOT NULL,
        build_id INTEGER NOT NULL,
        run_session_id INTEGER NOT NULL,
        test_ok BOOLEAN,
        PRIMARY KEY (id),
        UNIQUE (run_session_id, build_id),
        FOREIGN KEY(build_id) REFERENCES build (id),
        FOREIGN KEY(run_session_id) REFERENCES run_session (id),
        CHECK (test_ok IN (0, 1))
);
CREATE TABLE run (
        id INTEGER NOT NULL,
        measured_cycles INTEGER,
        reported_cycles INTEGER,
        time INTEGER,
        stack_usage INTEGER,
        min_power INTEGER,
```

```
        max_power INTEGER,
        avg_power INTEGER,
        median_power INTEGER,
        total_energy INTEGER,
        timestamp DATETIME,
        build_exec_id INTEGER,
        run_type VARCHAR NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY(build_exec_id) REFERENCES build_exec (id)
);
CREATE TABLE test_run (
        id INTEGER NOT NULL,
        checksumsmall_result VARCHAR,
        checksumfail_cause VARCHAR,
        test_ok BOOLEAN,
        PRIMARY KEY (id),
        FOREIGN KEY(id) REFERENCES run (id),
        CHECK (test_ok IN (0, 1))
);
CREATE TABLE crypto_hash_run (
        id INTEGER NOT NULL,
        msg_len INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(id) REFERENCES run (id)
);
```

```
CREATE TABLE power_sample (
        id INTEGER NOT NULL,
        run_id INTEGER NOT NULL,
        power FLOAT,
        current FLOAT,
        voltage FLOAT,
        timestamp INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(run_id) REFERENCES run (id)
);
CREATE TABLE crypto_aead_run (
        id INTEGER NOT NULL,
        data_id INTEGER,
        plaintext_len INTEGER,
        assoc_data_len INTEGER,
        mode VARCHAR(3),
        PRIMARY KEY (id),
        FOREIGN KEY(id) REFERENCES run (id),
        CHECK (mode IN ('enc', 'dec', 'frg'))
);
```

Bibliography

[1] Analog Devices. *AD8210 (Rev. D)*. URL: http://www.analog.com/media/en/technical-documentation/data-sheets/AD8210.pdf (visited on 07/09/2015).

[2] Jean-Philippe Aumasson et al. "Sha-3 proposal blake". In: *Submission to NIST* (2008). URL: http://aumasson.jp/blake/blake.pdf (visited on 07/30/2015).

[3] *Bausatz AVR-NET-IO - Bausätze / Module - Bausätze - - Pollin Electronic*. URL: http://www.pollin.de/shop/dt/MTQ5OTgxOTk-/Bausaetze_Module/Bausaetze/Bausatz_AVR_NET_IO.html (visited on 06/19/2015).

[4] Daniel J. Bernstein. *Cache-timing attacks on AES*. Nov. 11, 2004. URL: http://cr.yp.to/papers.html#cachetiming.

[5] Jeremy H. Brown and Brad Martin. "How fast is fast enough? choosing between Xenomai and Linux for real-time applications". In: *proc. of the 12th Real-Time Linux Workshop (RTLWS'12)*. 2010, pp. 1–17. URL: https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf (visited on 06/28/2015).

[6] Carsten Rolfes et al. "Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents". In: *Smart Card Research and Advanced Applications*. Springer, 2008, pp. 89–103. URL: http://link.springer.com/chapter/10.1007/978-3-540-85893-5_7 (visited on 05/12/2015).

[7] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. "Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI". In: *Low Power Electronics and Design, 2000. ISLPED'00. Proceedings of the 2000 International Symposium on*. IEEE,

2000, pp. 185–190. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=876779 (visited on 06/19/2015).

[8] Shu-jen Chang et al. *Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition*. NIST IR 7896. Gaithersburg, MD: National Institute of Standards and Technology, Nov. 2012. URL: http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf (visited on 07/30/2015).

[9] Christian Wenzel-Benner et al. "XBX benchmarking results". In: *The Third SHA-3 Candidate Conference*. The Third SHA-3 Candidate Conference. Washington Marriott Hotel, Washington, DC USA, Mar. 2012.

[10] *Crypto++ Library 5.6.2 - a Free C++ Class Library of Cryptographic Schemes*. URL: http://www.cryptopp.com/ (visited on 07/03/2015).

[11] Cryptographic Engineering Research Group, GMU. *ATHENa*. URL: https://cryptography.gmu.edu/athena/index.php?id=about (visited on 07/29/2015).

[12] *CryptoLUX > FELICS*. URL: https://www.cryptolux.org/index.php/FELICS (visited on 07/27/2015).

[13] Daniel Bernstein and Tanje Lange. "Software benchmarking of SHA-3 candidates." Quo Vadis Cryptology? SHA-3 Contest. LORD Hotel, Warsaw, May 24, 2011. URL: http://cr.yp.to/talks/2011.05.24/slides.pdf (visited on 05/20/2015).

[14] Daniel J. Bernstein. *Crypto competitions: CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness*. Cryptographic Competitions. May 28, 2015. URL: http://competitions.cr.yp.to/caesar.html (visited on 06/08/2015).

[15] Daniel J. Bernstein. *end of CAESAR round 1*. E-mail. July 7, 2015. URL: https://groups.google.com/forum/?hl=en#!original/crypto-competitions/upaRX2jdVCQ/ajIionLi45QJ (visited on 07/30/2015).

[16] Daniel J. Bernstein. *Salsa20*. URL: http://cr.yp.to/salsa20.html (visited on 07/27/2015).

[17] *DB Browser for SQLite*. URL: http://sqlitebrowser.org/ (visited on 07/26/2015).

[18] Daniel Dinu et al. "Triathlon of Lightweight Block Ciphers for the Internet of Things". In: (). URL: https://eprint.iacr.org/2015/209.pdf (visited on 07/27/2015).

[19] *Energia*. URL: http://energia.nu/ (visited on 07/27/2015).

[20] Eric S. Raymond. *The Lost Art of C Structure Packing*. URL: http://www.catb.org/esr/structure-packing/ (visited on 07/23/2015).

[21] Niels Ferguson et al. "The Skein hash function family". In: *Submission to NIST (round 3)* 7.7.5 (2010), p. 3. URL: https://www.schneier.com/skein1.3.pdf (visited on 07/30/2015).

[22] *File:MSP430-USB-LP.PNG - Texas Instruments Wiki*. URL: http://processors.wiki.ti.com/index.php/File:MSP430-USB-LP.PNG (visited on 07/29/2015).

[23] BeagleBoard org Foundation. *English: Picture of BeagleBone*. Oct. 2011. URL: https://commons.wikimedia.org/wiki/File:BeagleBone.jpg (visited on 07/28/2015).

[24] *FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions*. URL: http://www.freertos.org/ (visited on 07/07/2015).

[25] *Frequently Asked Questions - RTwiki*. URL: https://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions (visited on 06/29/2015).

[26] Kris Gaj. "Cryptographic Contests: Toward Fair and Comprehensive Benchmarking of Cryptographic Algorithms in Hardware". In: *DSD*. 2011, p. 11. URL: http://dsmc2.eap.gr/dsd2011/docs/DSD2011_Keynote_Speaker2.pdf (visited on 07/22/2015).

[27] Kris Gaj, Ekawat Homsirikamol, and Marcin Rogawski. "Fair and comprehensive methodology for comparing hardware performance of fourteen round two SHA-3 candidates using FPGAs". In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010, pp. 264–278. URL: http://link.springer.com/chapter/10.1007/978-3-642-15031-9_18 (visited on 07/30/2015).

[28] Kris Gaj et al. “ATHENa - Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs”. In: IEEE, Aug. 2010, pp. 414–421. ISBN: 978-1-4244-7842-2. DOI: 10.1109/FPL.2010.86. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5694286 (visited on 07/22/2015).

[29] I. Garcia-Vargas et al. “ROM-based finite state machine implementation in low cost FPGAs”. In: *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*. IEEE, 2007, pp. 2342–2347. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4374972 (visited on 05/19/2015).

[30] *GCC - Open Source Compiler for MSP430 Microcontrollers*. URL: http://www.ti.com/tool/msp430-gcc-opensource (visited on 07/27/2015).

[31] *GCC toolchain for MSP430*. SourceForge. URL: http://sourceforge.net/projects/mspgcc/ (visited on 07/27/2015).

[32] Danilo Gligoroski et al. “Cryptographic Hash Function BLUE MIDNIGHT WISH”. In: (2009). URL: http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf.

[33] Greg Davis. *Guidelines for writing efficient C/C++ code | Embedded*. Embedded.com. URL: http://www.embedded.com/design/mcus-processors-and-socs/4006634/Guidelines-for-writing-efficient-C-C--code (visited on 07/23/2015).

[34] B. Guido et al. “The K reference”. In: (2011). URL: http://www.sk-kari.put.poznan.pl/Stoklosa/2013-2014/Keccak-reference-3.0.pdf (visited on 07/30/2015).

[35] Ekawat Homsirikamol, Marcin Rogawski, and Kris Gaj. “Throughput vs. area trade-offs in high-speed architectures of five round 3 SHA-3 candidates implemented using xilinx and altera FPGAs”. In: *Cryptographic Hardware and Embedded Systems–CHES 2011*. Springer,

2011, pp. 491–506. URL: http://link.springer.com/chapter/10.1007/978-3-642-23951-9_32 (visited on 07/30/2015).

[36] *Intel AES-NI: Securing the Enterprise*. URL: http://www.intel.com/content/dam/doc/white-paper/enterprise-security-aes-ni-white-paper.pdf (visited on 07/25/2015).

[37] *Introducing Xenomai 3 – Xenomai*. URL: https://xenomai.org/introducing-xenomai-3/#What8217s_new_with_Xenomai_3 (visited on 06/29/2015).

[38] *IOTLIST - Discover the Internet of Things*. URL: http://iotlist.co/ (visited on 07/21/2015).

[39] *IPython Interactive Computing*. URL: http://ipython.org/ (visited on 07/26/2015).

[40] Jens-Peter Kaps et al. "Lightweight implementations of SHA-3 candidates on FPGAs". In: *Progress in Cryptology–INDOCRYPT 2011*. Springer, 2011, pp. 270–289. URL: http://link.springer.com/chapter/10.1007/978-3-642-25578-6_20 (visited on 05/07/2015).

[41] Jerome Radcliffe. "Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System". Blackhat 2011. Caesar's Palace, Las Vegas, NV, Aug. 3, 2011. URL: http://blackhat.com/html/bh-us-11/bh-us-11-briefings.html#Radcliffe (visited on 05/11/2015).

[42] John Pham. *Change I71e9a366: Enable hla_serial for TI ICDI devices | openocd.zylin Code Review*. URL: http://openocd.zylin.com/#/c/2527/ (visited on 07/25/2015).

[43] John Pham. *.rodata section incorrectly marked read-write with msp430-elf-gcc using msp430f5529.ld script*. TI E2E Community. URL: https://e2e.ti.com/support/development_tools/compiler/f/343/t/418312 (visited on 07/25/2015).

[44] Bernhard Jungk. "Evaluation of compact FPGA implementations for all SHA-3 finalists". In: *The Third SHA-3 Candidate Conference*. 2012. URL: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/papers/JUNGK_paper.pdf (visited on 07/30/2015).

[45] Stéphanie Kerckhof et al. “Compact FPGA implementations of the five SHA-3 finalists”. In: *Smart Card Research and Advanced Applications*. Springer, 2011, pp. 217–233. URL: http://link.springer.com/chapter/10.1007/978-3-642-27257-8_14 (visited on 07/30/2015).

[46] *lwIP - A Lightweight TCP/IP stack - Summary [Savannah]*. URL: http://savannah.nongnu.org/projects/lwip/ (visited on 07/06/2015).

[47] T. L Martin and D. P Siewiorek. “Nonideal battery and main memory effects on CPU speed-setting for low power”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9.1 (Feb. 2001), pp. 29–34. ISSN: 1063-8210. DOI: 10.1109/92.920816.

[48] Motor Industry Software Reliability Association, ed. *MISRA-C:2004: guidelines for the use of the C language in critical systems*. 2. ed, [Stand]: October 2004. Nuneaton: MIRA, 2008. 106 pp. ISBN: 978-0-9524156-2-6 978-0-9524156-2-6 978-0-9524156-4-0.

[49] *MSP430F5529 USB LaunchPad Evaluation Kit - MSP-EXP430F5529LP - TI Tool Folder*. URL: http://www.ti.com/tool/msp-exp430f5529lp (visited on 07/27/2015).

[50] *Open On-Chip Debugger*. URL: http://openocd.org/ (visited on 07/25/2015).

[51] Pete Semig. *A Current Sensing Tutorial–Part 1: Fundamentals | EE Times*. URL: http://www.eetimes.com/document.asp?doc_id=1279404 (visited on 06/30/2015).

[52] *Porting a Linux application to Xenomai dual kernel – Xenomai*. URL: http://xenomai.org/2014/08/porting-a-linux-application-to-xenomai-dual-kernel/ (visited on 06/29/2015).

[53] *Raspberry Pi 1 Model B*. URL: https://www.raspberrypi.org/products/model-b/ (visited on 06/25/2015).

[54] Richard F. Kayser. “Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family”. In: *Federal Register* (Nov. 2, 2007), pp. 62212–62220. URL: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf (visited on 07/29/2015).

[55] *SQLAlchemy - The Database Toolkit for Python*. URL: http://www.sqlalchemy.org/ (visited on 07/25/2015).

[56] *SQLite Home Page*. URL: https://www.sqlite.org/ (visited on 07/25/2015).

[57] *Start Here – Xenomai*. URL: https://xenomai.org/start-here/#How_does_Xenomai_deliver_real-time (visited on 06/29/2015).

[58] Texas Instruments. *EK-TM4C1294XL*. URL: http://www.ti.com/ww/en/launchpad/img/launchpad-tivac-04.jpg (visited on 07/28/2015).

[59] Texas Instruments. *INA225 | Current Shunt Monitor Analog Voltage Output | Shunt-based/Non-Isolated | Online datasheet*. URL: http://www.ti.com/product/INA225/datasheet (visited on 07/09/2015).

[60] Texas Instruments. *MSP430F551x, MSP430F552x Mixed Signal Microcontroller (Rev. L) - msp430f5529.pdf*. URL: http://www.ti.com/lit/ds/symlink/msp430f5529.pdf (visited on 07/09/2015).

[61] Texas Instruments. *Tiva C Series TM4C1294NCPDT Microcontroller Data Sheet (Rev. B)*. URL: http://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf (visited on 07/15/2015).

[62] Texas Instruments. *Zerø-Drift, Bi-Directional CURRENT/POWER MONITOR with I2CTM Interface (Rev. F)*. URL: http://www.ti.com/lit/ds/sbos448f/sbos448f.pdf (visited on 06/29/2015).

[63] *The GNU MP Bignum Library*. URL: https://gmplib.org/ (visited on 07/02/2015).

[64] Stefan Tillich, Christoph Herbst, and Stefan Mangard. "Protecting AES software implementations on 32-bit processors against power analysis". In: *Applied Cryptography and Network Security*. Springer, 2007, pp. 141–157. URL: http://link.springer.com/chapter/10.1007/978-3-540-72738-5_10 (visited on 07/24/2015).

[65] *TivaTM C Series LaunchPad Evaluation Kit - EK-TM4C123GXL - TI Tool Folder*. URL: http://www.ti.com/tool/ek-tm4c123gxl (visited on 07/27/2015).

[66] *TivaWareTM for C Series (Complete) - SW-TM4C - TI Software Folder*. URL: http://www.ti.com/tool/sw-tm4c (visited on 07/07/2015).

[67] U. S. Department of Commerce, NIST, and Q. H. Dang. *Secure Hash Standard (SHS): Federal Information Processing Standards Publication 180-4*. CreateSpace Independent Publishing Platform, July 2, 2012. 36 pp. ISBN: 978-1-4781-7807-1.

[68] Ulrich Radig. *Ulrich Radig, mikrocontroller and more :: AVR Webserver Software*. URL: http://www.ulrichradig.de/home/index.php/software/avr-webserver-software (visited on 07/06/2015).

[69] SparkFun Electronics from Boulder USA. *www.sparkfun.com/products/11837*. May 13, 2013. URL: https://commons.wikimedia.org/wiki/File:Raspberry_Pi_-_Model_A.jpg (visited on 07/28/2015).

[70] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1". In: *Advances in Cryptology–CRYPTO 2005*. Springer, 2005, pp. 17–36. URL: http://link.springer.com/chapter/10.1007/11535218_2 (visited on 07/29/2015).

[71] Christian Wenzel-Benner and Jens Gräf. "XBX: eXternal Benchmarking eXtension for the SUPERCOP crypto benchmarking framework". In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010, pp. 294–305. URL: http://link.springer.com/chapter/10.1007/978-3-642-15031-9_20 (visited on 05/20/2015).

[72] Hongjun Wu. "The hash function JH". In: *Submission to NIST (round 3)* (2011), p. 6. URL: http://www.ntu.edu.sg/home/wuhj/research/publications/2011_JH_Round3.pdf (visited on 07/30/2015).