Benchmarking Hash Functions on the MSP430

ECE 493Final Report

Team Members:
Elio Andia
Fletcher Ta
Margaux McGivern

Faculty Advisor: Dr. Jens-Peter Kaps

Date: May 2, 2011

Table of Contents

I. Executive Summary

The objective of this project is to add the Texas Instruments MSP430FG4618 microcontroller to the eXternal Benchmarking eXtension (XBX), a system that builds upon SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) to evaluate cryptographic algorithms on embedded devices. Specifically, XBX extends the eBASH (ECRYPT Benchmarking of All Submitted Hash Functions) portion of SUPERCOP. As a part of this project, the eXternal Benchmarking Harness (XBH) must be acquired, modified, and programmed correctly in order to function. The XBH will serve as an interface between the eXternal Benchmarking Software (XBS) and the eXternal Benchmarking Device (XBD). In this case, the XBD is the MSP430FG4618. The XBS will compile the hash functions, which in turn will be downloaded onto the MSP430 via the XBH. The harness will report the execution time and stack usage of the hash function on the device to the software. The data collected by the XBS will be in a form compatible with SUPERCOP's data analysis scripts[1]. A successful implementation will result in the ability to produce these database entries for the execution of any hash function on the MSP430.

[1] Wenzel-Benner and Gräf assert that they should be compatible. See pg. 300 of [**4**].

II. Introduction

A. Identification of Need

The National Institute of Standards is holding a competition for the next hash function, called “SHA-3”, which was proposed in response to the recent advances in cryptanalysis and papers published showing weaknesses in the current standard, SHA-1 [**1**]. To measure the performance of the software implementation of a cryptographic algorithm, the Virtual Application and Implementation Research Lab (VAMPIRE) created a toolkit called SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) that evaluates the algorithm based on a set of timing and size criteria [**2**]. Additional tools for benchmarking hash functions were developed, forming the eBASH (ECRPYT Benchmarking of All Submitted Hash Functions) portion of SUPERCOP [**3**].

B. Problem Statement

SUPERCOP is restricted to platforms that support the POSIX standard; this eliminates mobiles phones, PDAs, and Smart Cards, devices that are often target platforms for cryptographic algorithms [**4**]. The eXternal Benchmarking eXtension (XBX) developed by Christian Wenzel-Benner and Jens Gräf supports the SUPERCOP eBASH system for microcontrollers, using a combination of software and hardware components. The device target for the eXternal Benchmarking Software (XBS) is called the eXternal Benchmarking Device (XBD) and is replaceable in the XBX system. The XBS is able to run the SUPERCOP framework to benchmark an algorithm based on a given implementation, compiler, and options on the XBD [**4**]. The XBD for this project will be changed to a MSP430FG4618 microcontroller, expanding the family of microcontrollers supported by the XBX.

C. Approach

Wenzel-Benner and Gräf designed the XBX setup with a XBD target that can be interchangeable by changing the cross compiler on the XBS and supplying low level functions that are supported by the generic bootloader interface. The remaining components, the XBH and XBS, are set up following Wenzel-Benner and Gräf's design.

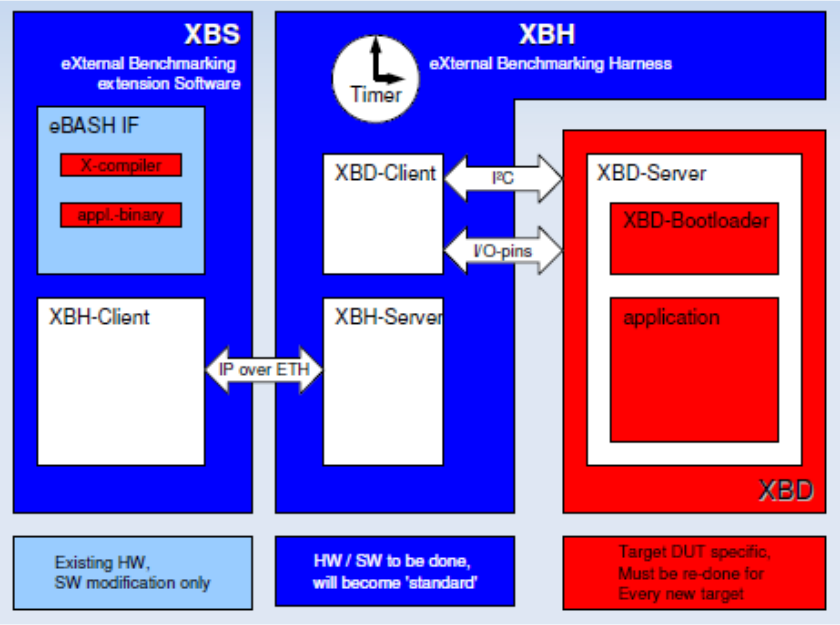


Figure 1 - Graphical representation of the standard and interchangeable components of the XBX [5].

Figure 1 illustrates the changes required when setting up the XBX with a new device. It also illustrates the operations of between the XBX components. XBS combines the application code of hash function to be benchmarked with the application framework, the communication services necessary for the XBD to communicate with the XBH. Application binaries are downloaded onto the XBD through the XBH, where they are executed and timed. The hardware abstraction layer, HAL, provides the functions necessary to benchmark the XBD:

- I^2C drivers for commands and data to be exchanged between the XBH and XBD.
- UART drivers for debugging output using the RS232 on the XBD.
- Routines for sending start and end execution signals from the device to the harness.

- Timing calibration to obtain the number of device cycles per harness cycles.
- Paint and count stack functions to obtain the stack usage measurement in bytes.

The necessary work required to add the MSP430 to the XBX system consists of both hardware and software. Initially the hardware was supposed to be simple, as currently implemented devices only had a six pin connections between the harness and the device itself [**5**]. Unlike previous implementations, the MSP430 runs at 3 V, whereas the harness runs at 5 V. The device datasheet for the MSP430 requires that the supply voltage not exceed 4.1 V. The I2C bus was also limited by the maximum supply voltage, prompting the need for a voltage interface between the harness and the device. The software followed the template of HAL functions, as enumerated above.

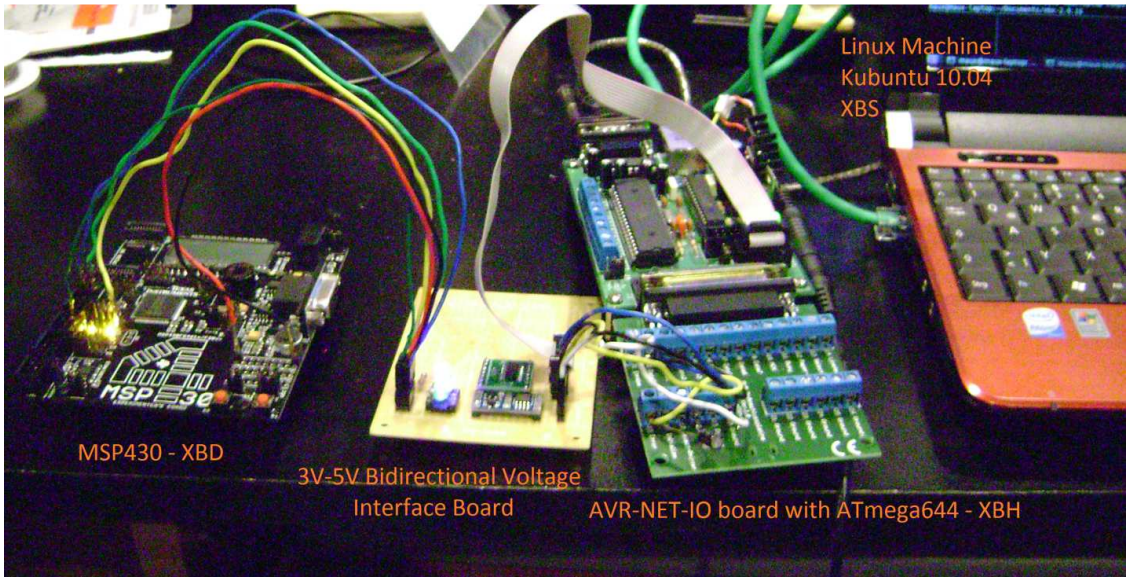


Figure 2 - XBX setup with the MSP430

As a part of adding the MSP430 to the XBX system, the existing setup must also be built.The XBH is built using an AVR-NET-IO board, a break-outboard for the DB-25 connector, and the ATmega644 microcontroller to replace the standard ATmega32 that ships with the AVR-NET-IO. The XBS requires a PC running Linux as well as an Ethernet port to connect (either straight-through or crossover).The contributions of each team member as followed are: the I^2C

communication were handled by Elio, the bootloader and applications were rewritten by Fletcher, and the integrations and set up coordination were handled by Margaux. A more detailed explanation on the descriptions of XBX components and task distributions can be referenced in the design document. Further instructions on how to benchmark are given from the XBX website.

III. Technical

A. The eXternal Benchmarking eXtension (XBX)

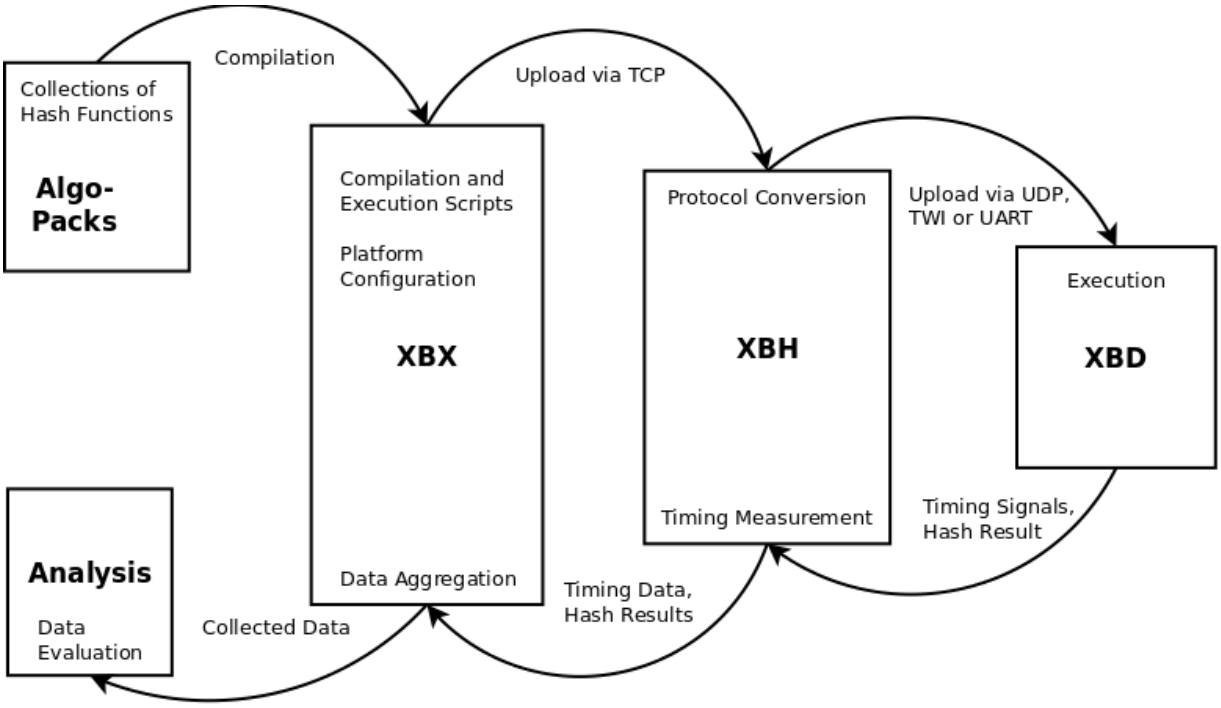


Figure 3 - Overview diagram from [5].

The overall operation of the XBX system is summarized in Figure 3. The required inputs for system operation are: algopack(s), compiler(s), compiler settings, platform settings (page size in bytes, communication mode, and clock frequency), and the hardware abstraction layer for the selected platform. The XBS has three main commands that can be run by the users: import, compile, and execute. Import will copy the algopack hash functions to a folder called algobase, which contains the hash functions to be executed on the XBD. The compile command reads the compiler options provided in the platform directory and compiles each hash function in the algobase with the application framework and stores these binaries in a platform specific directory. The execute command first reads the platform settings and then checks the XBH IP address argument for validity before initiating a TCP connection with the XBH. After the port number is set and the XBH revision number is received, the XBH issues a command to get the

bootloader status, beginning the first steps towards running the imported hash functions on the XBD. The execute script creates the result directories, performs timing error measurements, calculates the checksums for all the hash functions in the algobase, and issues commands to the XBH to upload the binary and reports the results. A short benchmark is run for each implementation of the hash functions with associated compiler and options. A stack usage measurement is performed at this time. Starting from the short benchmark, a 'best binary' is found and a detailed benchmark is then performed. This is where the XBD performs the hash for messages of different lengths[2].

The next step of benchmarking the hash functions is to pass the application binaries from the PC to the XBH by uploading them through TCP, and then striping the binaries from the TCP packets to send them to the device using I^2C. Hash functions are then executed on the XBD after receiving the appropriate commands from the harness. Results and timing signals are then sent to the XBH for timing measurements. Timing measurements on the XBH are calculated using the timestamps taken when the XBD sends start and end execution signals. The XBH returns the timing data and hash results to the PC for data aggregation.

The results from the compile and execution scripts are evaluated by using tools provided by the XBX top level directory. The errors, build stats, and binary information can be viewed in table format after the compilation. The execution script produces information about the checksum tests, quick benchmarks, detailed measurements, drift measurements (timing error calculation), and stack measurements. The implementations speeds of a cipher can be viewed in graph form provided the user has installed gnuplot[3].

[2] A summary of the execution command is included in [**4**]. The level of detail provided was found by an examination of the execute file included in the XBX software, available on the XBX website [**5**].
[3] A command line based plotting software.

The functions the XBS uses to perform benchmark execution include: ‘do_many_drift_measurements’, ‘get_timings’, ‘calculate_measured_cycles’, ‘checksum’, and ‘benchmark’. Each function sends a command to the XBH through a TCP connection and does not directly call the functions used by the XBD to perform timing measurements.

The XBH handles the protocol conversions and timing measurement. The functions within the XBH initialize and set up the connections of the PC and XBH as well as the XBH and XBD. Timing on the XBH is handled by interrupts on the input capture pins. The function ‘port_time_stamp_request’ calculates the time elapsed during execution. The remaining functions to be implemented are from the HAL, including ‘busy_loop_with_timing’, ‘load_string’, ‘switch_application’, ‘switch_bootloader’, ‘read_page and program_page’, ‘paint_stack and count_stack’. The support functions from the HAL will execute the commands from the XBH and provide debugging output.

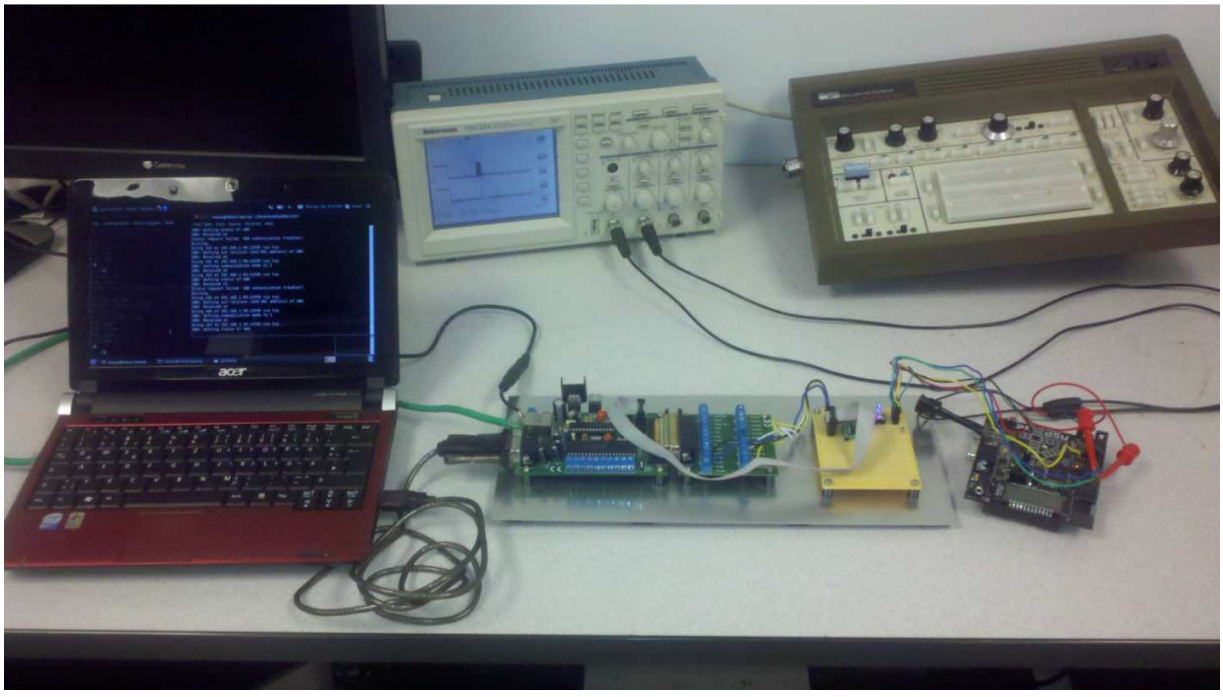


Figure 4 - Final XBX setup

B. XBD Circuit Diagram

The MSP430 requires an interface to act between itself and the XBH because the XBH is a 5 V device while the MSP430 is a 3 V device. Using the simplified schematic from the design document, a new interface board was designed to account for differing supply voltages.

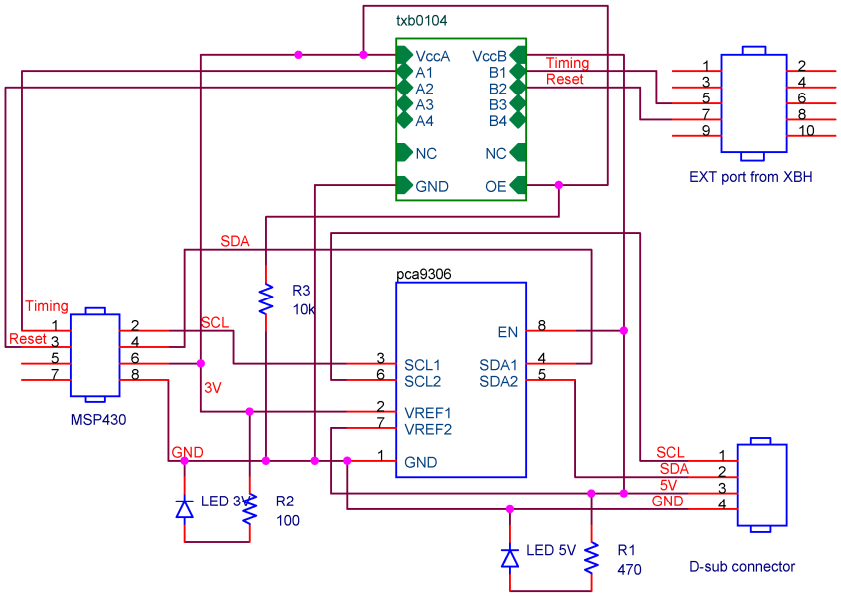


Figure 5 - Schematic for voltage level interface board between MSP430 XBD and ATmega644 XBH.

The power and ground from the MSP430 are connected to header pins soldered into the breadboard area (the MSP430FG4618 is mounted on an experimenter's board which provides various peripherals, such as a limited breadboard area and an RS232 port). The timing pin is connected to port 2 pin 1, which is Timer B0. The reset pin is connected to JTAG port 1, pin 11, as there is no connection on the breakout headers available.[4] The I^2C pins are connected to port 3 pins 1 and 2 for SDA and SCL, respectively. The circuit shown in the design document without the voltage interface was based on a schematic drawn by Christian Wenzel-Benner for the group to implement the 16 MHz ATmega644 XBD.

[4]There is a pull-up resistor connected internally on the MSP430 FG4618.

IV. Experimentation and Testing

The testing and experimental procedures for software were conducted by testing the rewritten bootloader functions in C for the MSP430. The previous XBX system used an ATmega1281 which called functions from its own library. Rewritten bootloader functions for the MSP430 were simulated and debugged on the microcontroller with the IAR embedded Workbench. Simulations in IAR were performed for the following MSP430 HAL functions: stack, flash, and communication. The data from these tests did not undergo any processing, but were used to verify functionality. The MSP430 was connected to the PC via the debugger, not the XBH. An IAR workspace with simulated inputs produced outputs that are then compared to the expected results. For the stack functions, the test verified that the RAM is filled with stack canaries and that the correct number of stack canaries are counted and returned. For the flash functions, patterned data was written to the flash and then verified through the debugger and read back into an array. The communication drivers under test were I^2C and UART. The I^2C driver was tested using the MSP430 experimenter's board, where the second chip, the F2013, was used as the master device. An echo communication took place, where the MSP430FG4618 sent a byte and the F2013 acknowledged the byte by returning it. The UART driver used a similar testing, where the characters received from the PC was sent back. The purpose of these tests was to catch any bugs before integrating the MSP430 with the XBH.

Testing and experimental procedures for the hardware were conducted by individually testing the operation of each electronic component before soldering and assembling. Once the XBH was assembled, setting up a connection between the XBH and XBS and XBD was conducted. Packets from the XBS to the XBH were successfully sent and received, confirmed by running the execute command from the XBS prompt. To test the XBX system, one of the SPHLIB hash functions

was compiled to run on the ATmega644 XBD. The execute command failed, retuning an I^2C non-acknowledgement. It was discovered that the I^2C driver on the XBH was masking out the least significant bit of the slave addresses, resulting in a constant zero address being sent to the device.

The MSP430 was connected to the XBH for testing. The first execute command failed, returning the same error as the ATmega644 XBD. This was expected, so the SDA and SCL pins of the both the MSP430 and the XBH were connected to a logic analyzer. The waveforms were as expected on the XBH side, but the waveform at the SDA and SCL pins of the MSP430 revealed a problem in the voltage interface board. This was fixed by removing a connection between the 3.3V output on the I^2C bus and the input voltage pin on the MSP430. The waveforms matched on both the XBH and XBD after this update. The MSP430 was still not responding to the request being sent from the XBH. It was after this test that the masking of the least significant bit of the slave address was discovered by looking closely at the I^2C bytes being sent on the oscilloscope. After changing the slave address to 2 on the XBH (effectively shifting the slave address left by one, making the least significant bit the read/write bit), the MSP430 began to respond to the XBH request, but only sent zeros back.

V. Administrative Detail

The tasks for this project were divided into software and hardware components. The software is the hardware abstraction layer, which must interface with the existing XBX system to support running the MSP430 as an XBD. The hardware portion of this project was divided between the existing implementation (XBH) using the test platform (ATmega644), and then the MSP430 as

the XBD and its associated voltage level shifter interface. The Progress summary table is shown below in Figure 6.

A. Progress Summary Table

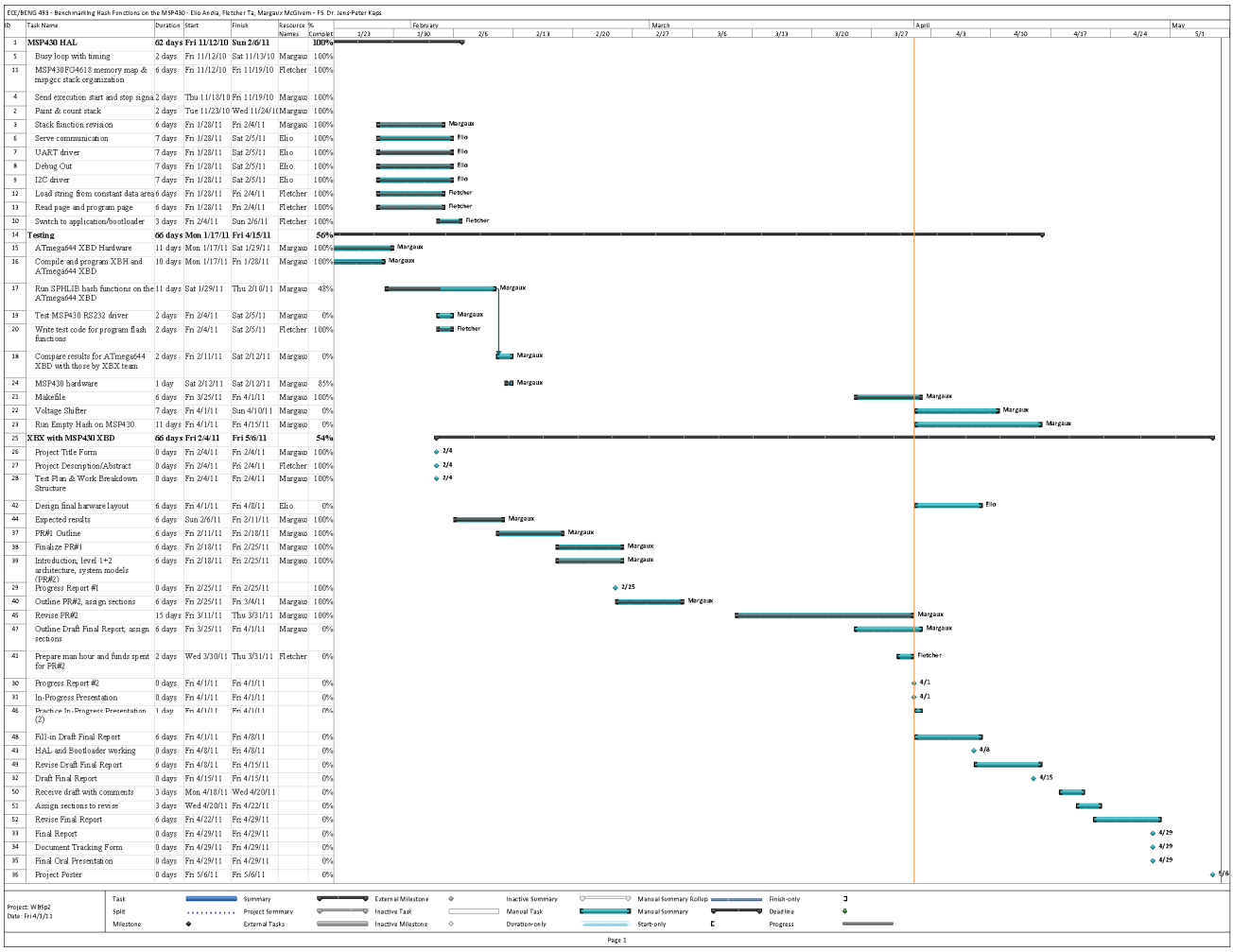


Figure 6 - Progress Summary Table

B. Fund Spent

Description	Quantity	Reciept Date	Vendor/Supplier	Elio	Fletcher	Margaux
AVR-NET-IO (payment 1)	1	10/13/10	Pollin Electronic			$ 62.70
AVR-NET-IO (payment 2)	1	10/29/10	Pollin Electronic			$ 51.68
Print and Paper supplies					$18.00	
ATmega644 MCU	2	11/8/11	AVNet			$ 11.99
Atmel AVRISPmkII	1	1/4/11	Newark			$ 34.00
Heat Sink	2	1/4/11	Newark			$ 0.83
PCA9306DCTT	2	1/4/11	Newark			$ 1.33
74LVX3245MTC	2	1/4/11	Newark			$ 1.79
2x5 Pin IDC Ribbon Cable	2	2/25/11	MDFly			$ 2.98
2x5 Pin Shrouded Header - 10 Pins	2	2/25/11	MDFly			$ 0.30
Breakboard for PCF8575	1	2/21/11	Sparkfun		$ 11.95	
Breakboard for Voltage translator	1	2/21/11	diydrones		$ 12.90	
Exp IC Perfboard	1	3/19/11	Radioshack			$ 2.99
LM317T Voltage Regulator	1	3/19/11	Radioshack			$ 2.49
40 Pin IC Socket	1	3/25/11	Radioshack			$ 0.99
8 Pin IC Socket	2	3/25/11	Radioshack			$ 0.48
Exp IC Perfboard	1	3/25/11	Radioshack			$ 2.99
TXB0104 Voltage Shifter	1	4/4/11	RobotShop			$ 5.06
9VDC Power Supply	1	4/4/11	RobotShop			$ 5.95
Seeedstudio Barrel Jack	1	4/4/11	RobotShop			$ 1.95
Ceramic Resonator	1	4/4/11	RobotShop			$ 0.95
		Total Money Spent by each member:		$ -	$ 42.85	$ 191.45
		Grand Total	**$ 234.30**			

Figure 7 - Funds Spent

The total amount of money spent purchasing the components to set up the XBX adds up to $234.30. After testing XBX on the control set up, with the original XBD ATmega644, we did not purchase the ATmega debugger which would have cost an additional $200. Our target XBD was the MSP430, and we had the debugger for the TI microcontroller from the lab kit purchased in ECE 447. We decided to just test and run the XBX system with the MSP430 as the XBD.

C. Man Hours

	Elio	Fletcher	Margaux
Total Time	110.0	128.0	160.0
Total Time from all team members			398.0
Total Labor Cost at $20/hour			$ 7,960.00
Labor Cost and Fund Spent			$ 8,194.30

Figure 8 - Labor Cost

Week	Week Ending	Elio		Fletcher		Margaux	
		Educational/ Learning	Productive Effort	Educational/ Learning	Productive Effort	Educational/ Learning	Productive Effort
1	3-Sep-10						
2	10-Sep-10						
3	17-Sep-10	1.0	3.0	3.0	3.0	2.0	3.0
4	24-Sep-10	4.0	2.0	2.0	2.0	4.0	2.0
5	1-Oct-10	2.0	5.0	1.0	5.0	1.0	5.5
6	8-Oct-10	2.0	1.0	3.0	2.0	1.0	2.0
7	15-Oct-10	2.0	1.0	2.0	2.0	3.0	4.0
8	22-Oct-10	1.0	1.0	2.0	1.0	1.0	2.0
9	29-Oct-10	1.0	2.0	2.0	1.0	6.5	2.0
10	5-Nov-10	2.0	0.0	2.0	3.0	4.0	1.0
11	12-Nov-10	2.0	5.0	2.0	3.0	3.0	4.0
12	19-Nov-10	0.0	1.0	2.0	4.0	6.0	6.0
13	26-Nov-10	1.0	1.0	1.0	2.0	2.0	1.0
14	3-Dec-10	0.0	0.0	2.0	2.0	2.0	3.0
15	10-Dec-10	0.0	0.0	1.0	2.0	1.0	2.0
16	17-Dec-10	1.0	0.0	0.0	0.0	0.0	0.0
17	24-Dec-10	2.0	1.0	0.0	0.0	0.0	0.0
18	31-Dec-10	0.0	0.0	0.0	0.0	0.0	0.0
19	7-Jan-10	1.0	0.0	0.0	0.0	2.0	1.0
20	14-Jan-10	1.0	0.0	4.0	0.0	5.0	0.0
21	21-Jan-10	2.0	0.0	2.0	0.0	1.0	0.0
22	28-Jan-10	5.0	1.0	4.0	0.0	5.0	2.0
23	4-Feb-10	0.0	2.0	2.5	3.0	2.0	10.0
24	11-Feb-10	2.0	4.0	3.8	1.0	1.0	5.0
25	18-Feb-10	4.0	1.0	3.0	1.0	1.0	3.0
26	25-Feb-10	1.0	8.0	6.0	3.0	1.0	4.0
27	4-Mar-10	0.0	2.0	1.0	2.0	1.0	3.0
28	11-Mar-10	0.0	1.0	0.0	0.0	1.0	1.0
29	18-Mar-10	4.0	0.0	2.0	10.0	0.0	0.0
30	25-Mar-10	1.0	4.0	2.0	4.0	1.0	4.0
31	1-Apr-10	4.0	2.0	0.0	4.0	1.0	5.0
32	8-Apr-10	2.0	1.0	0.0	4.0	5.0	5.0
33	15-Apr-10	0.0	1.0	1.0	4.0	1.0	4.0
34	22-Apr-10	5.0	7.0	1.0	3.0	4.0	7.0
35	29-Apr-10						
36	6-May-10						
37	13-May-10						
	Time Spent:	53.0	57.0	57.3	71.0	68.5	91.5
	Total Time	110.0		128.3		160.0	

Figure 9 - Man hours devoted

A large majority of the time devoted was on self-education and debugging. The JTAG interface and application of the bootstrap loader on the MSP430 had to be researched as the libraries of AVR library could not be used. Some of these functions had to be rewritten in assembly using the MSP430x instruction set. The I^2C-bus specification and user manual, as well as applying the I^2C Master-Slave Library had to be further researched as the topic was briefly covered in ECE447. The set up between the XBH and XBS was not immediately obvious either. In the code for the web server on which the XBH program is based, an interface via the serial port is provided to change the MAC and IP addresses of the XBH. Upon programming, the MAC address of the XBH was initialized to all F's, rendering it unable to communicate via the Ethernet port. A mismatch between the baud rate of the terminal and the device prevented this from being discovered, as the default baud rate of the web server program is 9600, while the program itself initializes the port at 115200 baud at first.

Immediately after the resolution of the IP and MAC addresses, the XBX scripts were able to connect to the XBH but failed to connect to the ATmega644 XBD, reporting possible problems with the I^2C connections. The debugging output suggests that the problem is related to the CRC, which was found to be related to the slave address and the read/write bit on the XBH, rather than any I^2C issue on the XBD. Most of the man hours were spend trying to find this particular bug in both the ATmega644 XBD and the MSP430 XBD.

With the use of logic analyzer, we concluded that there was a problem with the hardware components. The pull up resistor was increased and the voltage shifter was fixed. The problem now resides within the I^2C communication driver on the XBD, or the given communication on the XBH. Further detail on self-education is explained in the next section.

VI. Conclusion and Education

The last testing period for the MSP430 involved the I2C communication between the XBD and the XBH. The error reported through the RS232 port on the XBH was a non-acknowledgment on the MSP430 side. As mentioned in the experimentation and testing section, the problem was found in the XBH I^2C driver. Further oscilloscope readings showed that the correct request (bootloader status request: XBD03vir + crc), was being sent to the MSP430, but the response form the device was incorrect; all zeroes. Preliminary research into the actions taken by the bootloader when the status request is received suggests the problem can be isolated to three areas: the bootloader wrapper provided by the XBX team, the load string function in the HAL, or the condition for sending the response in the I^2C communication. Further testing will reveal which of the areas causes the sending of zeros to the XBH.

The team's education primarily consisted of the communication protocols required to implement the project, mainly because a majority of the debugging was done in this area. In addition to I^2C and UART, the issue of programming the upper portion of the flash was resolved by using 20-bit pointers. Unfortunately, the mspgcc compiler does not support these extended pointers easily, and IAR does not provide very sophisticated methods of handling them either, so the MSP430x instruction set was used to access 20-bit registers while read and programming flash.

A summary of the XBX system is that it was created to benchmark hash function. Researchers would benefit most from this project as the MSP430 is a new device that is now added to the XBX system. There would be no cost or profit, as the system is open source. If any new updates are done, it would most likely just involve updating the software on the XBS. The only time to add new hardware onto the design, is if the structure of XBX changes. Disposal of

any components of XBX should be handled similarly to the disposing of any other electronic devices.

VII. Bibliography

[1] (2010, June) National Institute of Standards and Technology. [Online]. http://www.nist.gov/itl/csd/ct/hash_competition.cfm

[2] (2011, May) SUPERCOP. [Online]. http://bench.cr.yp.to/supercop.html

[3] (2011, May) eBASH: ECRYPT Benchmarking of All Submitted Hashes. [Online]. http://bench.cr.yp.to/ebash.html

[4] Christian Wenzel-Benner and Jens Gräf, "XBX: eXternal Benchmarking eXtension for the SUPERCOP Crypto Benchmarking Framework," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, Stefan Mangard and François-Xavier Standaert, Eds.: Springer Berlin / Heidelberg, 2010, vol. 6225/2010, pp. 294-305.

[5] Christian Wenzel-Benner and Jens Gräf. (2010, November) XBX: eXternal Benchmarking eXtension. [Online]. https://xbx.das-labor.org/trac/wiki/HowItWorks

[6] Christian Wenzel-Benner and Jens Gräf. (2009, October) The Conferences - XBX: eXtenal Benchmarking eXtension. [Online]. http://www.hyperelliptic.org/SPEED/slides09/wenzel-XBX-benchmarking.pdf

Appendix A

Design Proposal

ECE-492 DESIGN DOCUMENT

BENCHMARKING HASH FUNCTIONS ON THE MSP430

The objective of the Benchmarking Hash Functions on the MSP430 project is to add the MSP430 microcontroller from TI to the eXternal Benchmarking eXtension (XBX), a system that builds on SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) performance evaluation suite for cryptographic algorithms. Specifically, XBX extends eBASH (ECRYPT Benchmarking of All Submitted Hash Functions). As a part of the project the eXternal Benchmarking Harness (XBH) must be acquired, modified, and programmed in order to function correctly. The XBH will serve as the interface between the eXternal Benchmarking Software (XBS) and the eXternal Benchmarking Device (XBD), in this case the MSP430. The XBS will be able to run the imported hash functions on the MSP430 via the XBH and take timing and power measurements while the MSP430 executes the hash functions. The output of the XBS will be in a form compatible with the database entry output given by the SUPERCOP program for

October 15, 2010

J.P. Kaps

Team Members:
Elio Andia
Fletcher Ta
Margaux McGivern

Executive Summary

The objective of the Benchmarking Hash Functions on the MSP430 project is to add the MSP430 microcontroller from TI to the eXternal Benchmarking eXtension (XBX), a system that builds on SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) performance evaluation suite for cryptographic algorithms. Specifically, XBX extends eBASH (ECRYPT Benchmarking of All Submitted Hash Functions). As a part of the project the eXternal Benchmarking Harness (XBH) must be acquired, modified, and programmed in order to function correctly. The XBH will serve as the interface between the eXternal Benchmarking Software (XBS) and the eXternal Benchmarking Device (XBD), in this case the MSP430. The XBS will be able to run the imported hash functions on the MSP430 via the XBH and take timing and power measurements while the MSP430 executes the hash functions. The output of the XBS will be in a form compatible with the database entry output given by the SUPERCOP program for CPUs in personal computers. A successful implementation will result in the ability to produce these database entries for the execution of any hash function on the MSP430.

Problem Statement

The National Institute of Standards is holding a competition for the next hash function, called “SHA-3”, which was proposed in response to the recent advances in cryptanalysis and papers

published against the approved standard, SHA-1 [1]. To measure the performance of a cryptographic algorithm the Virtual Application and Implementation Research Lab (VAMPIRE) created a toolkit called SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) that evaluates the algorithm based on a set of timing and size criteria. Additional tools for benchmarking hash functions were developed, forming the eBASH (ECRPYT Benchmarking of All Submitted Hash Functions) portion of SUPERCOP. The only problem with SUPERCOP is that it restricts its use to platforms that support the POSIX standard, which eliminates mobiles phones, PDAs, and Smart Cards, devices that are often target platforms for cryptographic algorithms [2]. The eXternal Benchmarking eXtension (XBX) developed by Christian Wenzel-Benner and Jens Gräf supports the SUPERCOP eBASH system for microcontrollers, using a combination of software and hardware components. The device target for the eXternal Benchmarking Software (XBS) is called the eXternal Benchmarking Device (XBD) and is replaceable in the XBX system. The XBS is able to run the SUPERCOP framework to benchmark an algorithm based on a given implementation, compiler, and options on the XBD.

The XBD for this project is the MSP430. The motivation for this selection is based on the current microcontroller used in ECE 447 and ECE 511, which provides background knowledge to successfully implement the hardware abstraction layer (HAL) and drivers for the XBD. Furthermore, the MSP430 is not a part of the existing XBX implementations. Given the XBX system's capability with the SUPERCOP framework, its future applications are not limited to hash functions; stream ciphers, Diffie-Hellman (DH) functions, and public-key encryption and signatures are all possible extensions (SUPERCOP, 2010).

Approach

The purpose of the eXternal Benchmarking eXtension (XBX) system is to extend SUPERCOP's benchmarking capabilities to small devices. The goal of this project is to add the MSP430 to the current XBX implementation.

Wenzel-Benner and Gräf designed the XBX in three parts: one software part and two hardware parts. The software portion of the system is called eXternal Benchmarking Software (XBS) and is responsible for the compilation of the hash functions and overhead code (called the application framework) necessary to successfully benchmark an algorithm on the target device. The hardware setup consists of the eXternal Benchmarking Harness (XBH) and the eXternal Benchmarking Device (XBD). The XBH handles the communication between the XBS and XBD as well as the timing measurements of the executed hash functions.

The XBS performs in a similar fashion to the SUPERCOP scripts it was modeled after, with the exception that it requires a user to specify which platform the software will be benchmarking. XBS first combines the application code of the hash function to be benchmarked with the application framework, the communication services necessary for the XBD to communicate with the XBH. A size check on this binary is performed and a communication session between the XBS, XBH, and XBD is made in order to download the binaries. After download the XBD executes the code and the results are sent back to the XBH and then to the XBS running on the PC. The most important part of the XBS for this project is the hardware abstraction layer, HAL. The HAL will provide the necessary drivers to allow I^2C communication between the XBH and XBD, which is the channel through which the XBS commands are given to the XBD. The

locations of program memory, constant data storage, and the methods of manipulating the data there are hidden from the XBS by the HAL.

To accomplish the goal of the project, a hardware setup for the XBX system is required to connect the MSP430 while it runs the hash functions to take measurements of the time and power consumptions. In order to do this, an intermediary interface for the XBS and the XBD is required. The eXternal Benchmarking Harness (XBH) performs this function. The commands from the XBS are sent via TCP to the XBH and then via I^2C or UART to the XBD using the XBH protocol. The firmware for the XBH is a modified version of an embedded web server program by Ulrich Radig and is provided by the XBX system developers. In order to add the MSP430 to the platforms benchmarked (and possibly add similar microcontrollers in future projects) it is necessary to create the communication drivers that will allow the MSP430 to communicate via I^2C or UART with the XBH as well as firmware that will execute the XBS binaries and provide debugging output. These combined functions serve as the hardware abstraction layer (HAL) for the XBS. The hardware connections to the XBH must also be mapped to the MSP430. The data connection signal (UART or I^2C) is established between the XBH and XBD by the bootloader, which also handles signals that come from the control I/O lines for hard reset and timing measurements.

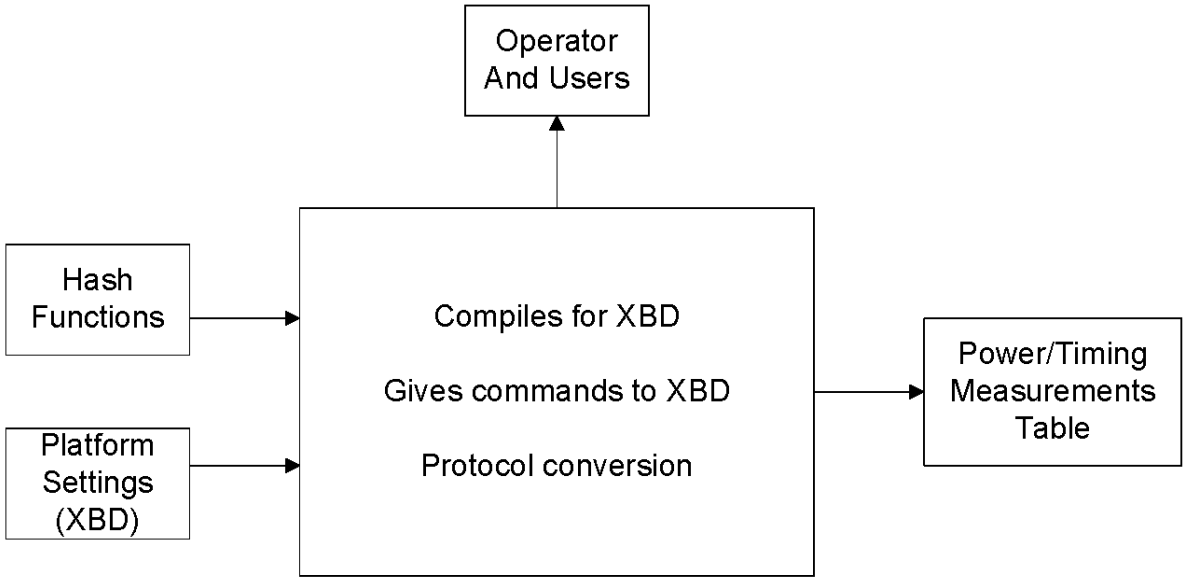
Our XBH design will be similar to the original XBX design, setting up an Atmel ATmega644 running at 20 MHZ opposed to the 16 MHZ chip used by Wenzel-Brenner and Gräf, an ENC28J60 Ethernet controller, and a MAX232 TTL/RS232 voltage level shifter on the AVR-NET-IO board.

The XBS for this project is required to run on a Linux machine. On recommendation, Kubuntu 10.04 was chosen as the development environment to add the MSP430 to the XBX platforms. In

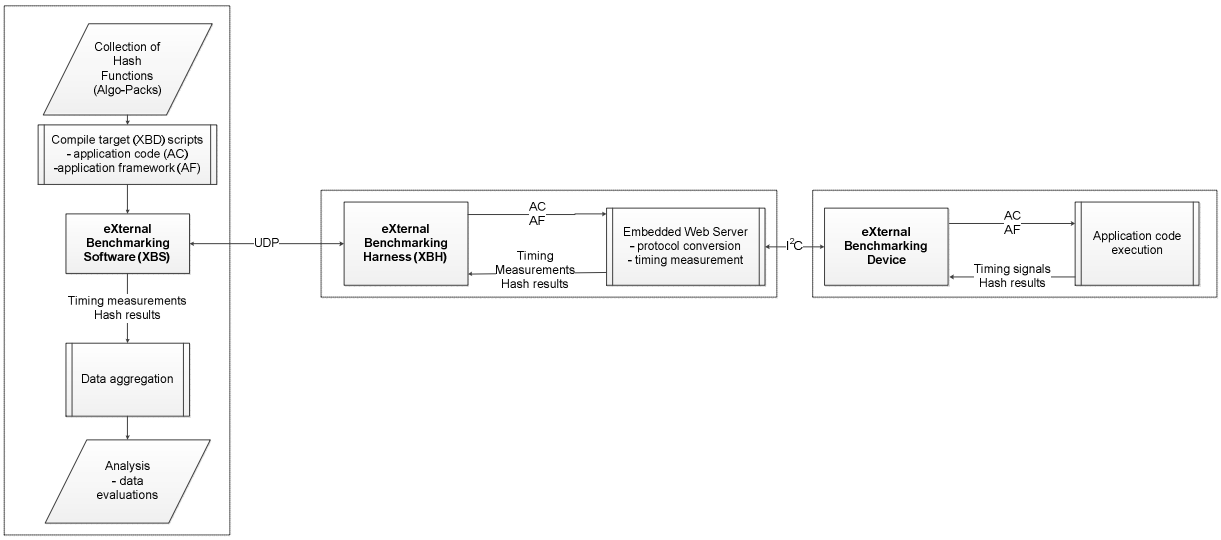
order to run the hash functions on the MSP430, some assembly conversion is required for the chosen algorithms provided in C. The assembly versions for the MSP430 will be provided to the team. The specific platform version that will be used in the project is the MSP430FG4618 in order to reduce monetary and time costs as this is the device currently used in ECE 447 (Single-Chip Microcomputers), which all team members are taking.

Preliminary Design

Black Box



White Box



The XBH setup is comprised of an Atmel ATmega644 microcontroller, a Microchip ENC28J60 Ethernet controller and a MAX232 TTL/RS232 voltage level shifter. The XBD is the device on which the actual benchmarking takes place; the MSP430. The basic set up will be a PC that runs XBS and the XBH acting as a fixed interface to the XBD.

The XBH will be connected to the PC running the XBS via its Ethernet port. This will provide an interface with any computer that supports the SUPERCOP framework regardless of its operating system. An RS232 port is available for low level configuration and debugging output during development [2]. The communication between the XBD and the XBH will be handled through a data connection and control I/O lines. As previously stated, the data connection will be implemented using either I^2C or UART. The digital I/O lines will be used for control purposes, such as resetting the device and timing measurement.

The control I/O lines are connected to the reset pin on the XBD. This allows the XBH to issue a hardware reset on the XBD, either due to a timeout or because of a command received

from the PC. In the event that the XBD crashes, e.g. due to stack overflow, the XBH will be able to recover communication to the XBD in a situation where the data connection would fail [2]. Timing measurement is another reason for dedicated control I/O lines. An output pin on the XBD is connected to the XBH timer event capture pin. The times when a function begins executing and when it stops executing are captured in the XBH and the amount of time in clock cycles can be calculated.

In order to perform such functions as I^2C communication, USART for debugging, sending signals, stack measurement, switching between the application program and the bootloader, and others, the hardware abstraction layer (HAL) must be written. This allows the XBS to create binaries with functions that are suited to the particular platform being benchmarked. The HAL also provides a timing calibration service, which accounts for the time difference between clock cycles on the XBH and XBD. This allows for accurate timing measurements to be computed on the XBH using the timestamps from the XBD.

Achieving a successful implementation of the hardware abstraction layer and development of hardware specific drivers for the microcontroller will enable the MSP430 to run the hash function as the XBD.

Preliminary Experimentation Plan

- Set up and test XBX and XBH with a preexisting platform, Atmega644, to make sure that the XBH works correctly.
- Compile the XBX for the Atmega644 platform.
- Run hash functions on the Atmega644 which acts as a prototype XBD.
- Compare results file with already acquired results available online.
- Write the firmware for MSP430 (bootloader).

- Write the drivers for MSP430 (HAL).
- Test drivers outside of XBS to see if they work with the MSP430
- Compile with XBS.
- Use the SPY debugger to test data communication
- Test the firmware on the MSP430 by means of a debugger
- Test the drivers via the XBH
- Run various hash functions multiple times and compare the result tables.
- Preliminary List and Brief Description of Tasks and Allocation of Responsibilities

Hardware

- eXternal Benchmarking Harness — Margaux McGivern
    - AVR-NET-IO
    - Heat sink
    - D-sub connector
- Benchmarking Platform Test Microcontrollers — Margaux McGivern
    - Atmega644
- MSP430FG4618 — Elio Andia, Fletcher Ta, Margaux McGivern
    - MSP430FG4618 Data pin connections (XBD): Elio Andia, Fletcher Ta

Software

- eXternal Benchmarking Software — Margaux McGivern
    - Test platforms: Atmega644
- MSP430 Communication Drivers — Elio Andia
    - I^2C/SPI
- MSP430 Firmware — Fletcher Ta

Milestones and Tasks

Date	Milestone/Task
5-Nov-10	Existing XBX system running
12-Nov-10	Design Review Presentation
19-Nov-10	MSP430 electrically connected to the XBX system HAL and firmware (bootloader) ready to test Draft Design Document
25-Nov-10	HAL and firmware (bootloader) working
3-Dec-10	Design Document
14-Jan-11	One hash function running on the MSP430 via the XBX
4-Feb-11	Project Title Form Project Description/Abstract Test Plan and WBS
25-Feb-11	Progress Report #1
1-Apr-11	In-Progress Presentation and Review Progress Report #2
15-Apr-11	Draft Final Report
29-Apr-11	Oral Presentation Final Report Document Tracking Form
6-May-11	Project Poster

References

[1] (2010, June) National Institute of Standards and Technology. [Online]. http://www.nist.gov/itl/csd/ct/hash_competition.cfm

[2] Christian Wenzel-Benner and Jens Gräf, "XBX: eXternal Benchmarking eXtension for the SUPERCOP Crypto Benchmarking Framework," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, Stefan Mangard and François-Xavier Standaert, Eds.: Springer Berlin / Heidelberg, 2010, vol. 6225/2010, pp. 294-305.

[3] (2010, Oct.) SUPERCOP. [Online]. http://bench.cr.yp.to/supercop.html

Appendix B

Design Document

Benchmarking Hash Functions on the MSP430

ECE 492 Design Document

Team Members:
Elio Andia
Fletcher Ta
Margaux McGivern

Faculty Advisor: Dr. Jens-Peter Kaps

Date: 12/03/2010

Table of Contents

I. Introduction

A. Identification of Need

The National Institute of Standards is holding a competition for the next hash function, called “SHA-3”, which was proposed in response to the recent advances in cryptanalysis and papers published showing weaknesses in the current standard, SHA-1 [**1**]. To measure the performance of the software implementation of a cryptographic algorithm, the Virtual Application and Implementation Research Lab (VAMPIRE) created a toolkit called SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) that evaluates the algorithm based on a set of timing and size criteria. Additional tools for benchmarking hash functions were developed, forming the eBASH (ECRPYT Benchmarking of All Submitted Hash Functions) portion of SUPERCOP.

B. Problem Statement

A problem with SUPERCOP is that it restricts its use to platforms that support the POSIX standard, which eliminates mobiles phones, PDAs, and Smart Cards; devices that are often target platforms for cryptographic algorithms [**2**]. The eXternal Benchmarking eXtension (XBX) developed by Christian Wenzel-Benner and Jens Gräf supports the SUPERCOP eBASH system for microcontrollers, using a combination of software and hardware components. The device target for the eXternal Benchmarking Software (XBS) is called the eXternal Benchmarking Device (XBD) and is replaceable in the XBX system. The XBS is able to run the SUPERCOP framework to benchmark an algorithm based on a given implementation, compiler, and options on the XBD.

Wenzel-Benner and Gräf designed the XBX in three parts: one software and two hardware parts. The software portion of the system is called eXternal Benchmarking Software (XBS) and is responsible for the compilation of the hash functions and an overhead code (called

the application framework) necessary to successfully benchmark an algorithm on the target device. The hardware setup consists of the eXternal Benchmarking Harness (XBH) and the eXternal Benchmarking Device (XBD). The XBH handles the communication between the XBS and XBD as well as the timing measurements of the executed hash functions.

The XBS performs in a similar fashion to the SUPERCOP scripts it was modeled after, with the exception that it requires a user to specify which platform the software will be benchmarking. XBS combines the application code of the hash function to be benchmarked with the application framework, the communication services necessary for the XBD to communicate with the XBH. A size check on this binary is performed to ensure that it does not exceed the XBD's storage and a command is sent to the XBD, through the XBH, in order to download the binaries. After download, the XBD executes the code and the results are sent back to the XBH and then to the XBS running on the PC. The most important part of the XBS for this project is the hardware abstraction layer, HAL. The HAL provides the necessary drivers to allow I^2C communication between the XBH and XBD, which is the channel through which the XBS commands are given to the XBD. The locations of program memory, constant data storage, and the methods of manipulating the data are hidden from the XBS by the HAL.

The XBD for this project is the MSP430. The motivation for this selection is based on the current microcontroller used in ECE 447 and ECE 511, which provides background knowledge to successfully implement the hardware abstraction layer (HAL) and drivers for the XBD. Furthermore, the MSP430 is not a part of the existing XBX implementations. The current XBD devices include some of Atmel's AVR ATmega processors, Texas Instruments' Luminary Micro, and a few Linux compatible routers.

Atmel	TI	Linux Routers
ATmega1284p	LM3S811	NSLU2 (Linksys)
ATmega1281		FRITZ!box Fon WLAN 7170 (AVM)
ATmega644		Artilla M-501

Table 1 The XBX currently supported platforms.

These platforms are grouped according to their hardware abstraction layer and compiler compatibility. The devices in Table 1 (shown by manufacture/device type) fall under the following categories: AVR based (Atmel processors), ARM based (LM3S811), and embedded Linux devices (LM3S811, NSLU2, Artilla M-501). The addition of the MSP430 will provide a template for TI's line of MSP430 microcontrollers, which have a unique RISC instruction set. TI's Luminary Micro microcontrollers use a modified Thumb instruction set and an ARM GCC or ARM C compiler, whereas the MSP430 will use its own version of the GNU cross compiler to compile the hardware abstraction layer and the application framework.

Given the XBX system's capability with the SUPERCOP framework, its future applications are not limited to hash functions; stream ciphers, Diffie-Hellman (DH) functions, and public-key encryption and signatures are all possible extensions (SUPERCOP, 2010).

II. Requirements Specification

To accomplish the goal of the project, a hardware setup for the XBX system is required to connect the MSP430 while it runs the hash functions to take measurements of the time to compute. In order to do this, an intermediary interface for the XBS and the XBD is required. The eXternal Benchmarking Harness (XBH) performs this function. The commands from the XBS are sent via TCP to the XBH and then via I^2C or UART to the XBD using the XBH protocol. This protocol uses ASCII encoded commands with ASCII encoded hex digits and the parameters [**2**]. The firmware for the XBH is a modified version of an embedded web server program by Ulrich Radig and is provided by the XBX system developers. In order to add the MSP430 to the platforms benchmarked (and possibly add similar microcontrollers in future projects) it is necessary to create the communication drivers that will allow the MSP430 to communicate via I^2C or UART with the XBH as well as addition support functions that will execute the XBS commands and provide debugging output. These combined functions serve as the hardware abstraction layer (HAL) for the XBS. The hardware connections to the XBH must also be mapped to the MSP430. The data connection signal (UART or I^2C) between the XBH and XBD is established by the bootloader of the XBD, which also handles signals that come from the control I/O lines for hard reset and timing measurements.

Our XBH design will be similar to the original XBX design, setting up an Atmel ATmega644 running at 16MHz using the on board clock, an ENC28J60 Ethernet controller, and a MAX232 TTL/RS232 voltage level shifter on the AVR-NET-IO board for debugging output.

The XBS for this project is required to run on a Linux machine. On recommendation, Kubuntu 10.04 was chosen as the development environment to add the MSP430 to the XBX platforms. In order to run the hash functions on the MSP430, some assembly conversion is

required for the chosen algorithms provided in C. The assembly versions for the MSP430 will be provided to the team. The specific platform version that will be used in the project is the MSP430FG4618. This will reduce monetary and time costs as this is the device currently used in ECE 447 (Single-Chip Microcomputers), which all team members are currently enrolled in.

III. System Architecture

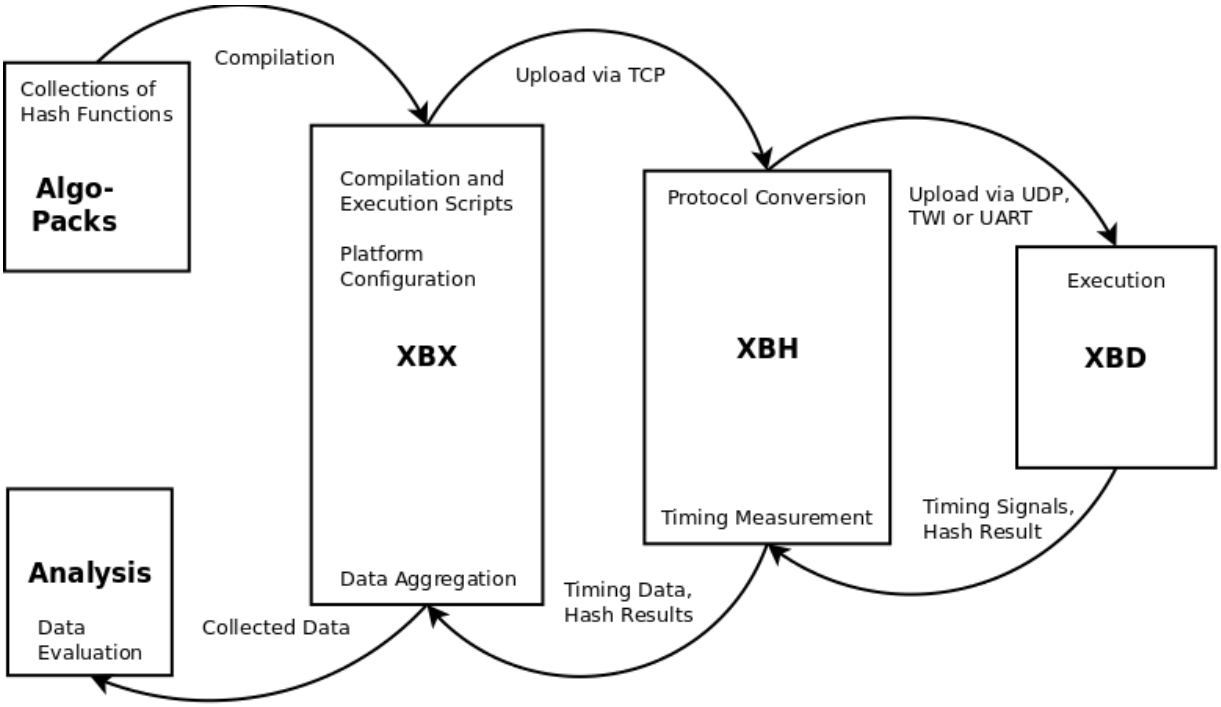


Figure 2 Overview diagram from [4].

The overall operation of the XBX system is shown in Figure 1. The required inputs for system operation are: algopack(s), compiler(s), compiler settings, platform settings (page size in bytes, communication mode, and clock frequency), and the hardware abstraction layer for the selected platform. The XBS has three main commands that can be run by the users: import, compile, and execute. Import will copy the algopack hash functions to a folder called algobase, which contains that hash functions to be executed on the XBD. The compile command reads the compiler options provided in the platform directory and compiles each hash function in the algobase with the application framework and stores these binaries in a platform specific directory. The execute command first reads the platform settings and then checks the XBH IP address argument for validity before initiating a TCP connection with the XBH. After the port number is set and the XBH revision number is received, the XBH issues a command to get the bootloader and application framework beginning the first steps towards running the imported

hash functions on the XBD. The execute script creates the result directories, performs timing error measurements, calculates the checksums for all the hash functions in the algobase,and issues commands to the XBH to upload the binary and reports the results. A short benchmark is run for each implementation of the hash functions with associated compiler and options. A stack usage measurement is performed at this time. Starting from the short benchmark, a 'best binary' is found and a detailed benchmark is then performed. This is where the XBD performs the hash for messages of different lengths.

The results from the compile and execution scripts are evaluated by using tools provided by the XBX top level directory. The errors, build stats, and binary information can be viewed in table format after the compilation. The execution script produces information about the checksum tests, quick benchmarks, detailed measurements, drift measurements (timing error calculation), and stack measurements. The implementations speeds of a cipher can be viewed in graph form provided the user has installed gnuplot.

The next step of benchmarking the hash functions is to pass the hash functions from the PC to the XBH by uploading them through TCP, and then converting the protocol to upload from XBH to XBD by UART for execution. This process is done by the XBH initializing the Ethernet port, UART, and I^2C channels and setting up the TCP and UDP channels with the PC and XBD, respectively. Hash functions are then executed on the XBD. Results and timing signals are then returned to the XBH for timing measurements. Timing measurements on the XBH are handled with the port time stamp request function. XBH then returns the timing data and hash results to the PC for data aggregation.

IV. Detail Design

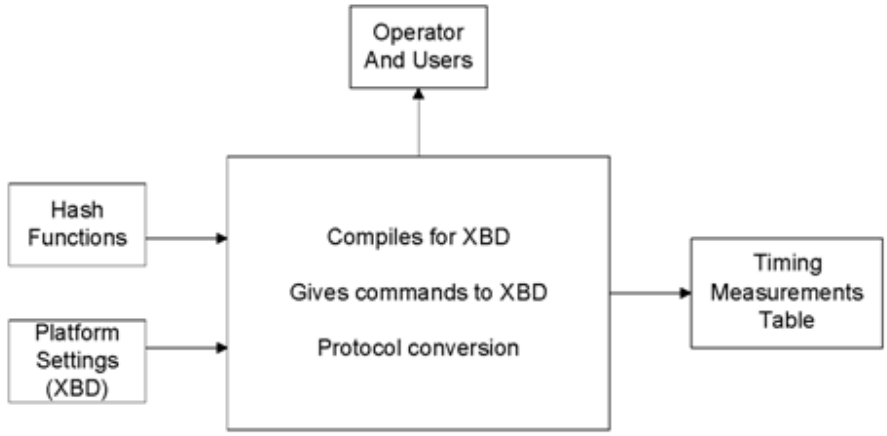


Figure 3 Level 0 Black Box Design from the Preliminary Design of the Design Proposal.

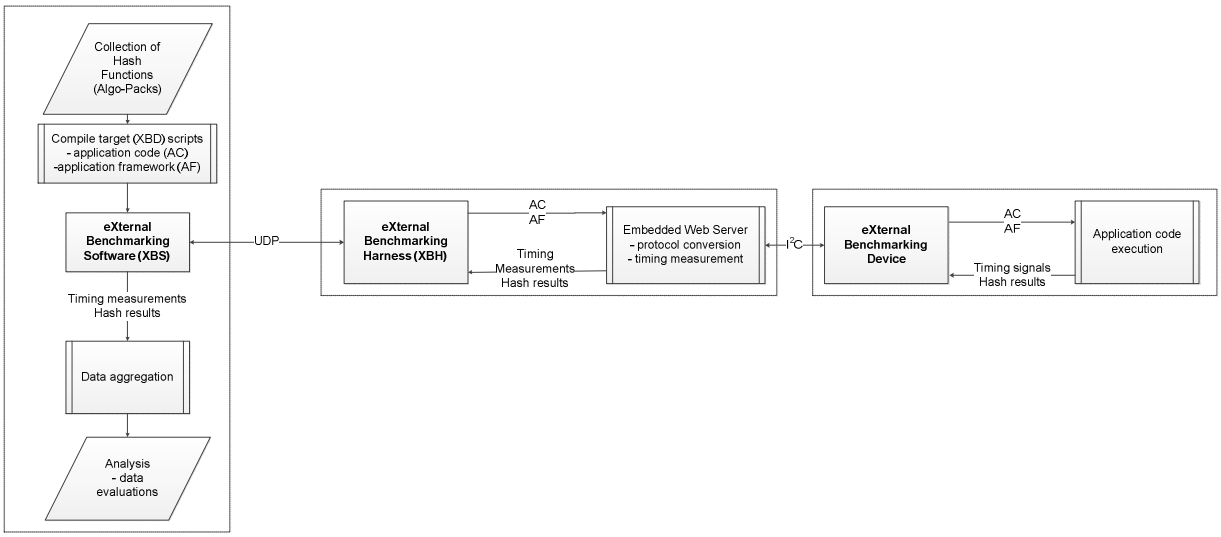


Figure 4 Level 1White Box Design from the Preliminary Design of the Design Proposal.

The top level designs are shown in Figure 2 and Figure 3. Expected inputs of the design are the hash functions provided and the platform settings of the XBD. The design will allow compilation of hash functions and commands, such as writing and reading flash memory and taking measurements on the XBD. The results will be collected and the execution time calculated by the XBH and formatted for SUPERCOP scripts by the XBS. A summary of our preliminary design and a description of benchmarking hash functions on the XBX were

explained in System Architecture. This section will restate the necessary changes for replacing the XBD with the MSP430, explain what functions are used by the XBX, and provide a more detailed top level view for further clarification.

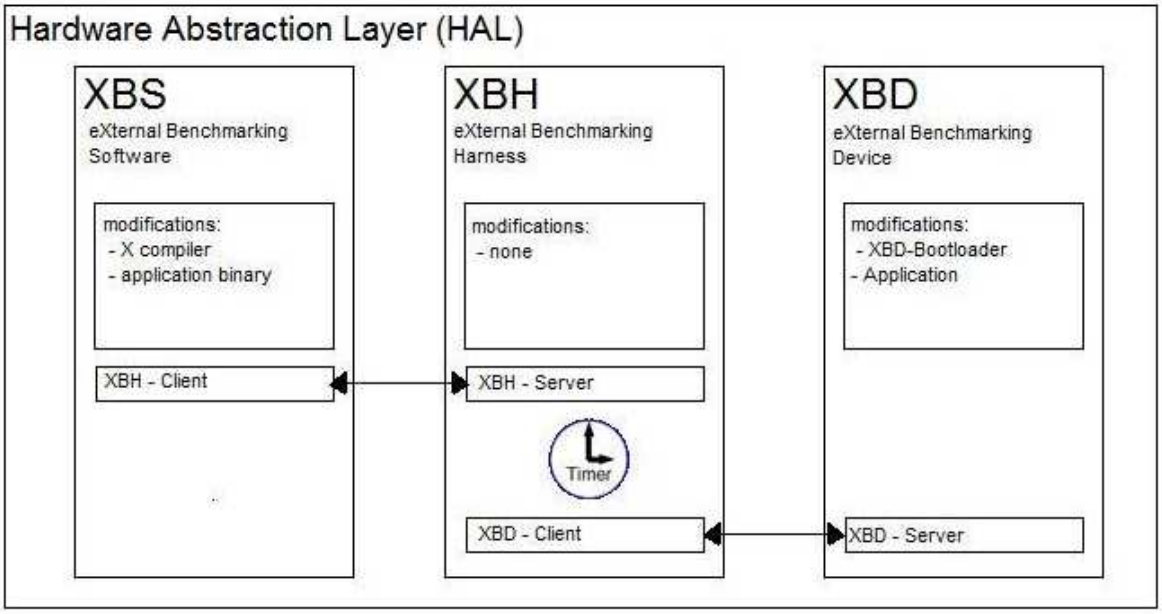


Figure 5 A Redrawn Diagram of [5].

A restatement of the necessary changes to be done is represented by Figure 4, a redrawn diagram from Wenzel's power point [5]. These changes will be: to replace the previously used XBD (the Atmega644) with the MSP430, to modify the XBD bootloader, to write the hardware abstraction layer for the XBD, and to modify the compiler and platform settings on the XBS. The XBH will remain unchanged. As defined in the introduction, the HAL is used by the XBS and the XBD bootloader to provide the necessary drivers for I^2C communication between the XBH and XBD.

Compilation and execution scripts, platform configuration, and data aggregation are handled by the XBS. The functions the XBS uses to perform benchmark execution include: 'do many drift measurement', 'get timings', 'calculate measured cycles', 'checksum', and

'benchmark'. Each function sends a command to the XBH through a TCP connection and does not directly call the functions used by the XBD to perform timing measurements.

The XBH handles the protocol conversions and timing measurement. The functions within the XBH initialize and set up the connections of the PC and XBH as well as the XBH and XBD. Timing on the XBH is handled by interrupts on the input capture pins. The function 'port time stamp request' calculates the time elapsed during execution. The remaining functions to be implemented are from the HAL, including 'busy loop with timing', 'load string', 'switch application', 'switch bootloader', 'read and program page', 'paint and count stack'. The support functions from the HAL will execute the commands from the XBH and provide debugging output.

C. XBD Circuit Diagram

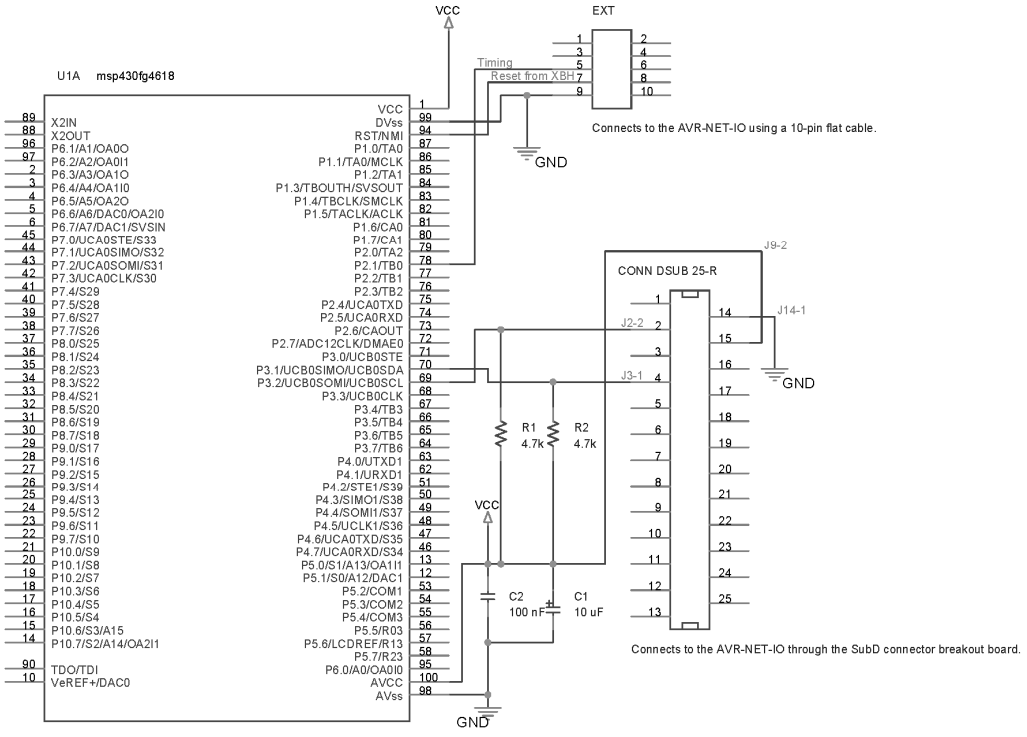


Figure 6 Schematic for the MSP430FG4618 XBD connections to the XBH. External circuit provided by Christian Wenzel-Benner and modified for the MSP430FG4618.

V. Prototyping Progress Report

The stack and flash programming code has been written for the XBD. The two ATmega644 MCUs, the AVR-NET-IO, D-sub connector, RS232 to TTL converter, USB to RS232 cable, and the test platform crystal and (2) 22 pF decoupling capacitors have all been acquired. The (2) 100 nF and 10 uF capacitors and 10k and (2) 4.7k resistors used in the XBD power supply circuit and the I2C data and clock lines were already owned by the team.

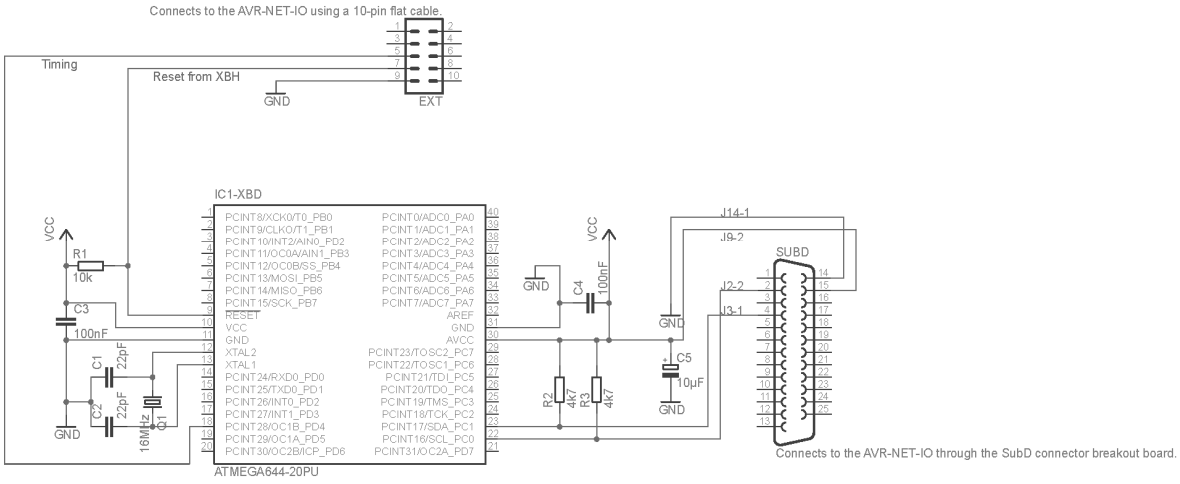


Figure 7 ATmega644 XBD circuit diagram, provided by Christian Wenzel-Benner.

The power supply will remain as described below, but the interface between the AC to DC adapter and XBH as well as the voltage level shifting between the XBH and the MSP430 may change as the design approaches the end of the prototyping phase.

A +12 V supply for the XBH is provided by an AC to DC adapter from an external hard drive enclosure, which conveniently includes a switch and jumpers to easily connect power and ground lines. The enclosure serves as a temporary heat sink, allowing the 7805 voltage regulator on the AVR-NET-IO to maintain a temperature that is not uncomfortable to touch. Test runs without the heat sink resulted in unacceptable temperature levels. The second regulator, the LM317, for the +3 V output on the AVR-NET-IO experienced no noticeable increase in

temperature during test operations with or without the heat sink on the 7805. The AVR-NET-IO has various +5 V outputs and ground terminals to source to an external device; these will serve as the power to the XBD.

The outputs from the XBH require no additional components other than the pull-up resistors and decoupling capacitors for the ATmega644 XBD because it operates at +5 V. The MSP430 is a 3 V device, not tolerate to +5 V inputs. This requires voltage level shifting at the inputs. Currently two solutions are being considered: a voltage divided using resistors between the output from the XBH and the MSP430, or a specific IC that performs the voltage level shifting. If the latter option is chosen, the PCA9306 bidirectional I2C-bus voltage-level translator will serve for the I2C lines, which would also remove the need for pull-up resistors on the data and clock lines. The TXS0104E voltage-level translator or something similar would serve as a general bidirectional interface between the XBH and XBD. Once the voltage shift is achieved, no additional components are needed because the experimenter's board includes decoupling capacitors for the clock crystal and a capacitor between the input voltage and ground.

VI. Testing Plan

1. Set up and test XBX and XBH with a preexisting platform, the ATmega644, to make sure that the XBH works correctly.
2. Compile the XBX for the Atmega644 platform.
3. Run hash functions on the Atmega644 which acts as a prototype XBD.
4. Compare results file with already acquired results available online.
5. Write the firmware for MSP430 (bootloader).
6. Write the drivers for MSP430 (HAL).
7. Test drivers outside of XBS to see if they work with the MSP430
8. Compile with XBS.
9. Use the USB FET debugger to test data communication
10. Test the firmware on the MSP430 by means of a debugger
11. Test the drivers via the XBH
12. Run various hash functions multiple times and compare the result tables.
13. Report Results to XBX project and SUPERCOP.

VII. List and Description of Tasks

D. Fall 2010 Task Allocation

The Fall 2010 task list is focused on the setup of the existing system implementation and the software required to add the MSP430 to the XBX platforms.

Task Name	Duration	Start	Finish	Resource Names
BHFMSP430 (Fall 2010)	**55 days**	**Fri 10/29/10**	**Sat 1/14/11**	
mspgcc toolchain & programming MSP430	29 days	Fri 11/12/10	Wed 12/22/10	Elio
Existing XBX system running	7 days	Fri 11/26/10	Mon 12/6/10	Margaux
HAL and bootloader ready to test	**26 days**	**Fri 10/29/10**	**Mon 12/6/10**	
Paint & count stack	2 days	Tue 11/23/10	Wed 11/24/10	Margaux
Send execution start and stop signal	2 days	Thu 11/18/10	Fri 11/19/10	Margaux
Busy loop with timing	2 days	Fri 11/12/10	Sat 11/13/10	Margaux
Serve communication	3 days	Sat 11/27/10	Tue 11/30/10	Elio
UART driver	22 days	Fri 10/29/10	Mon 11/29/10	Elio
Debug Out	2 days	Fri 11/26/10	Mon 11/29/10	Elio
I2C Driver	22 days	Fri 10/29/10	Mon 11/29/10	Elio
Switch to application/bootloader	2 days	Fri 12/3/10	Mon 12/6/10	Fletcher
MSP430FG4618 memory map & mspgcc stack organization	6 days	Fri 11/12/10	Fri 11/19/10	Fletcher
Load string from constant data area	2 day	Sat 11/27/10	Sun 11/28/10	Fletcher
readPage & programPage	5 days	Fri 11/19/10	Wed 11/24/10	Fletcher
Expected results	13 days	Fri 12/3/10	Tue 12/21/10	Margaux
HAL and bootloader working	12 days	Mon 12/6/10	Wed 12/22/10	
One hash function running on the MSP430 via the XBX	6 days	Fri 1/7/11	Fri 1/14/11	Margaux

E. Spring 2011 Task Allocation

The Spring 2011 task list moves forward from the hardware implementation and revises the code written last semester. The goal is to produce expected results similar to the test XBD (ATmega644) and integrate the remaining sphlib hash functions into the MSP430 algopack.

Task Name	Duration	Start	Finish	Resource Names
BHFMSP430 (Spring 2011)	**35 days**	**Fri 1/7/11**	**Fri 2/25/11**	
Project Assessment & Task Assignment	0 days	Fri 1/7/11	Fri 1/7/11	
Prepare Results Table for Hash Function	11 days	Fri 1/7/11	Fri 1/21/11	Fletcher
Code Revision & Commenting	11 days	Fri 1/7/11	Fri 1/21/11	Margaux
Compiler options	11 days	Fri 1/7/11	Fri 1/21/11	Elio
Next available set of hash functions	12 days	Sat 1/15/11	Sat 1/28/11	Elio
Project Update Presentation	11 days	Sat 1/29/11	Fri 2/11/11	
Third set of hash functions	11 days	Fri 2/11/11	Fri 2/25/11	Fletcher

VIII. Schedule and Milestones

Project Milestones	Date
Existing XBX system running	**Mon 12/6/10**
HAL and bootloader working	**Wed 12/22/10**
One hash function running on the MSP430 via the XBX	**Fri 1/14/11**
Next available set of hash functions	**Sat 1/22/11**
Third set of hash functions	**Fri 2/25/11**

F. Fall 2010 Administrative Schedule

Task Name	Duration	Start	Finish
Fall 2010 ECE 492	**77 days**	**Fri 9/17/10**	**Fri 12/3/10**
Project Title Form	0 days	Fri 9/17/10	Fri 9/17/10
Draft Proposal	0 days	Fri 10/8/10	Fri 10/8/10
Oral Proposal Presentation	0 days	Fri 10/15/10	Fri 10/15/10
Oral Design Review Presentation	0 days	Fri 11/12/10	Fri 11/12/10
Draft Design Document	0 days	Fri 11/19/10	Fri 11/19/10
Document Tracking Form	0 days	Fri 12/3/10	Fri 12/3/10
Design Document	0 days	Fri 12/3/10	Fri 12/3/10

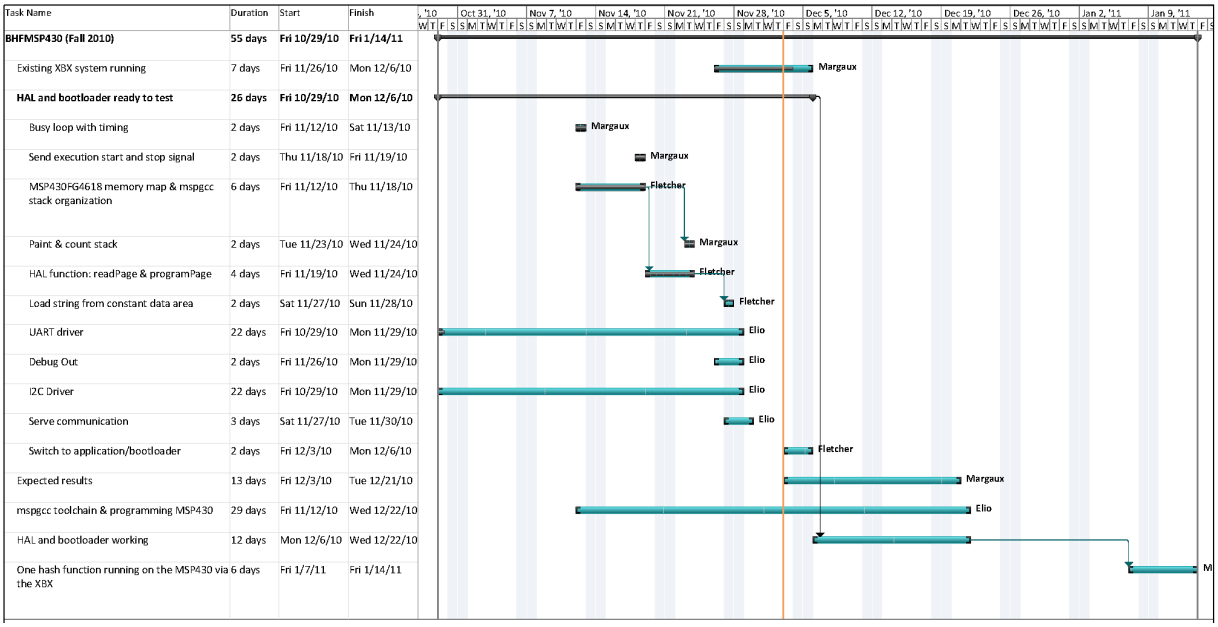
G. Spring 2011 Administrative Schedule

Task Name	Duration	Start	Finish
Spring 2011 ECE 493	**91 days**	**Mon 2/7/11**	**Mon 5/9/11**
Project Title Form	0 days	Mon 2/7/11	Mon 2/7/11
Project Description/Abstract	0 days	Mon 2/7/11	Mon 2/7/11
Progress Report #1	0 days	Fri 2/25/11	Fri 2/25/11
Test Plan & Work Breakdown Structure	0 days	Mon 2/7/11	Mon 2/7/11
Progress Report #2	0 days	Fri 4/1/11	Fri 4/1/11
Draft Final Report	0 days	Mon 4/18/11	Mon 4/18/11
Final Report	0 days	Mon 5/2/11	Mon 5/2/11
Document Tracking Form	0 days	Mon 5/2/11	Mon 5/2/11
In-Progress Presentation	0 days	Mon 4/4/11	Mon 4/4/11
Final Oral Presentation	0 days	Mon 5/2/11	Mon 5/2/11
Project Poster	0 days	Mon 5/9/11	Mon 5/9/11

IX. Gantt Charts

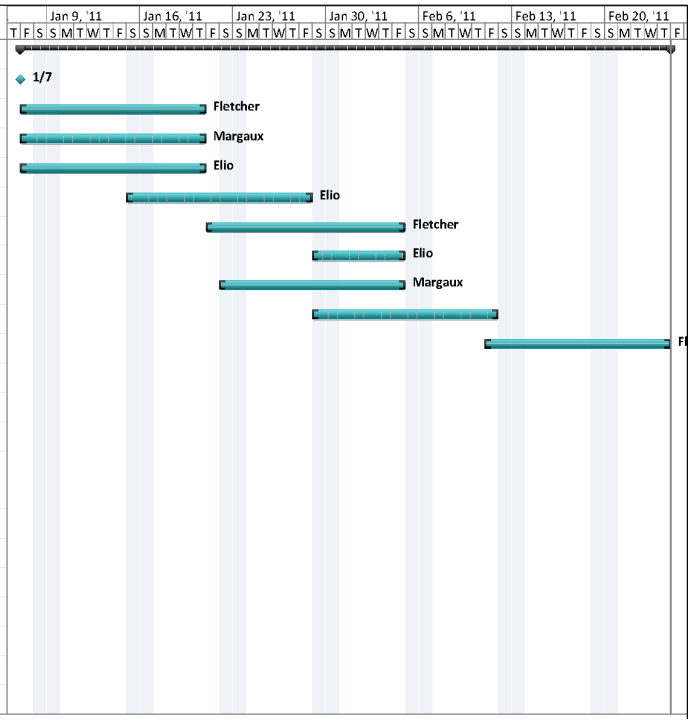
H. Fall 2010

Task Name	Duration	Start	Finish
BHFMSP430 (Fall 2010)	**55 days**	**Fri 10/29/10**	**Fri 1/14/11**
Existing X8X system running	7 days	Fri 11/26/10	Mon 12/6/10
HAL and bootloader ready to test	**26 days**	**Fri 10/29/10**	**Mon 12/6/10**
Busy loop with timing	2 days	Fri 11/12/10	Sat 11/13/10
Send execution start and stop signal	2 days	Thu 11/18/10	Fri 11/19/10
MSP430FG4618 memory map & mspgcc stack organization	6 days	Fri 11/12/10	Thu 11/18/10
Paint & count stack	2 days	Tue 11/23/10	Wed 11/24/10
HAL function: readPage & programPage	4 days	Fri 11/19/10	Wed 11/24/10
Load string from constant data area	2 days	Sat 11/27/10	Sun 11/28/10
UART driver	22 days	Fri 10/29/10	Mon 11/29/10
Debug Out	2 days	Fri 11/26/10	Mon 11/29/10
I2C Driver	22 days	Fri 10/29/10	Mon 11/29/10
Serve communication	3 days	Sat 11/27/10	Tue 11/30/10
Switch to application/bootloader	2 days	Fri 12/3/10	Mon 12/6/10
Expected results	13 days	Fri 12/3/10	Tue 12/21/10
mspgcc toolchain & programming MSP430	29 days	Fri 11/12/10	Wed 12/22/10
HAL and bootloader working	12 days	Mon 12/6/10	Wed 12/22/10
One hash function running on the MSP430 via the X8X	6 days	Fri 1/7/11	Fri 1/14/11



I. Spring 2011

Task Name	Duration	Start	Finish
BHFMSP430 (Spring 2011)	**35 days**	**Fri 1/7/11**	**Fri 2/25/11**
Project Assesment & Task Assignment	0 days	Fri 1/7/11	Fri 1/7/11
Prepare Results Table for Hash Function	11 days	Fri 1/7/11	Thu 1/20/11
Code Revision & Commenting	11 days	Fri 1/7/11	Fri 1/21/11
Compiler options	11 days	Fri 1/7/11	Fri 1/21/11
Next available set of hash functions	12 days	Sat 1/15/11	Fri 1/28/11
Presentation Slides	11 days	Fri 1/21/11	Fri 2/4/11
Research & Select Parts	6 days	Sat 1/29/11	Fri 2/4/11
Design Final Hardware Setup	11 days	Sat 1/22/11	Fri 2/4/11
Project Update Presentation	11 days	Sat 1/29/11	Fri 2/11/11
Third set of hash functions	11 days	Fri 2/11/11	Fri 2/25/11



X. References

[1] (2010, June) National Institute of Standards and Technology. [Online]. http://www.nist.gov/itl/csd/ct/hash_competition.cfm

[2] Christian Wenzel-Benner and Jens Gräf, "XBX: eXternal Benchmarking eXtension for the SUPERCOP Crypto Benchmarking Framework," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, Stefan Mangard and François-Xavier Standaert, Eds.: Springer Berlin / Heidelberg, 2010, vol. 6225/2010, pp. 294-305.

[3] (2010, Oct.) SUPERCOP. [Online]. http://bench.cr.yp.to/supercop.html

[4] Christian Wenzel-Benner and Jens Gräf. (2010, November) XBX: eXternal Benchmarking eXtension. [Online]. https://xbx.das-labor.org/trac/wiki/HowItWorks

[5] Christian Wenzel-Benner and Jens Gräf. (2009, October) The Conferences - XBX: eXtenal Benchmarking eXtension. [Online]. http://www.hyperelliptic.org/SPEED/slides09/wenzel-XBX-benchmarking.pdf