Experimental Evaluation of Hash Function Performance on Embedded Devices

Matthew Fritter, Nadir Ould-Khessal, Scott Fazackerley, Ramon Lawrence
Department of Computer Science
University of British Columbia

***Abstract*—With embedded devices collecting, manipulating, and transmitting growing amounts of data in various Internet of Things applications, it is increasingly important to process data on device for performance and energy efficiency. A common data processing function is computing hash functions for use in hash-based data structures and algorithms. The limited computation and memory resources of embedded devices results in different performance characteristics compared to general purpose computers. This research implements and experimentally evaluates the performance of non-cryptographic hash functions. Seven hash function algorithms were chosen on the basis of implementation complexity, popularity, and compatibility with microcontroller architecture. These functions were implemented in C/C++ for the ATmega328P 8-bit microcontroller used in the Arduino Uno, and on the Microchip PIC24 16-bit microcontroller. Some optimizations were implemented to reduce memory usage. Experimental results demonstrate that there are platform specific performance differences.**

***Index Terms*—hash function, Arduino, embedded software, performance, Internet of Things**

I. INTRODUCTION

Hash functions on microcontrollers are useful in data storage and transmission applications, such as hash table data structures or checksums for verifying data quality. Microcontroller hardware constraints affect hash algorithm performance. Many cryptographic or otherwise complex hashing algorithms that are commonly used in desktop computing are impractical to implement on a microcontroller. Small RAM sizes inhibit the use of algorithms with large key sizes, such as SHA-512 or RSA-1024, where the key itself would take up a significant portion of the memory. Hash functions that require multiple rounds of calculation, such as MD5 [1], also present a problem as the low clock speeds result in long computation times.

The contribution of this work is an experimental evaluation of non-cryptographic hash functions on two popular, general purpose microcontrollers: the 8-bit ATmega328P used in the Arduino Uno, and the 16-bit Microchip PIC24. The results demonstrate that there are platform-specific differences to hash function performance, which implies certain functions should be preferably used over others depending on the platform.

The remainder of the paper provides background on embedded hardware platforms with focus on the ATmega328P and PIC24. Section 3 presents the experimental evaluation and results, and the paper closes with future work and conclusions.

II. BACKGROUND

The volume and diversity of embedded devices is rapidly increasing especially as the Internet of Things expands [2], [3]. Increasingly, embedded devices are performing more substantial data processing rather than just data collection and transmission as there is an advantage to processing data on an embedded device rather than transmitting it over the network for processing [4]. Projects such as MicroSearch [5] and IonDB [6] have aimed to provide an efficient means of indexing and searching data on embedded devices through hash-based data structures.

A. Embedded Hardware Platforms

The Arduino Uno [7] uses the 8-bit AVR ATmega328P-PU microcontroller. It has 32 KB of flash program memory, 2 KB of SRAM, 1 KB EEPROM, and supports clock speeds up to 20 MHz. The Uno is programmed in a language called Wiring that is a subset of C/C++, with Arduino-specific libraries providing extra functionality [7]. The Arduino was designed to be an easily programmable prototyping tool for students, however it has since become a popular and inexpensive option for rapid prototyping and sensor deployment in a variety of fields. The Arduino platform has been used for cognitive science technologies [8], automation of laboratory procedures [9], and remote surveillance robots [10]. Code can be stripped of Arduino dependencies and compiled via alternate toolchains to most standard AVR microcontrollers.

Microchip PIC24 is a family of 16-bit microcontrollers that offers a good balance of low cost, low power and high performance. The PIC24FJ1024 chip was used to implement and test the hash functions, as the chip offers the largest program memory (1 MB) in the PIC24 family along with 32 KB SRAM. The test environment had a 32 MHz operating frequency that produces an instruction cycle of 16 MHz, as two cycles are needed to execute an instruction. The chip also includes a 17 bit by 17 bit Single Cycle Multiplier and a 32 bit by 16 bit hardware divider [11]. The project was developed using the Microchip Explorer 16/32 development board. The MCU is mounted to the board as a plug-in module. The board itself offers several development hardware tools including an in-circuit programmer, serial to USB interface, I2C interface and a 16X2 LCD display [12]. Project development used the Microchip MPLAB X IDE and Microchip XC16 C compiler.

TABLE I
MICROCONTROLLER SPECIFICATIONS

	ATmega328P-PU	PIC24FJ1024
Program Memory	32 KB	1024 KB
SRAM	2 KB	32 KB
Architecture	8-bit RISC Harvard	16-bit Modified Harvard
IPS	20 MIPS@20MHz	16 MIPS@32MHz
ALU	8-bit	16-bit
Hardware Multiplier	8-bit $\times$ 8-bit	17-bit $\times$ 17-bit
Hardware Divider	None	32-bit $\times$ 16-bit
Multi-bit Shift	None	15-Bit Shift/Cycle

B. *Hash Functions*

A hash function maps an input value of variable size to a fixed size key value. Good hash functions minimize the chance of two different inputs producing the same key, known as a collision, by ensuring that small changes to the input induce large changes in the output, and that key values are evenly distributed across the output range of the hash function. A variety of both non-cryptographic [13] and cryptographic [14] hash functions have been developed to meet different requirements for collision resistance, domain size, and performance. Non-cryptographic hash functions are commonly used in a variety of data structures, including hash maps and bloom filters, as well as for search, indexing, and data integrity verification.

The hash functions chosen in this study are either in common use or had published standards, such as CRC-32, MD5, and Lookup3 (see Table II). MD5 has popularity in non-cryptographic applications and a key size of 128 bits. All other algorithms were standardized to a 32-bit key length.

Previous research into hash function performance has primarily been in the fields of cryptography [14] or on server computer systems. Previous work in hash algorithm performance on embedded devices has largely focused on the cryptographic hash functions, particularly algorithms associated with the Secure Hash Algorithm (SHA) and Advanced Encryption Standards (AES) [15]. MD5 performance on 8-bit microcontrollers has been tested as part of a larger suite of cryptographic functions, although the hardware used for the testing had greater memory and storage capacities [16], and our results produced a more compact and better performing MD5 implementation.

III. EXPERIMENTAL EVALUATION

All implementations were written in C/C++ and were identical for both platforms. Where possible exact code implementations from original author implementations were used. All functions were tested with known input/output pairs to ensure correct functionality before analysis. Analysis was performed by measuring the time taken to hash a byte array, ranging from 1 to 256 bytes in length. The byte array was randomly generated at runtime. Time was calculated for one thousand executions of the hash function for each length of input bytes. For this evaluation, the focus is on hash algorithm performance and not other factors like output key distribution.

A. *Arduino Results*

The Arduino results are shown in Figure 1, which graphs the average time to calculate a single hash in milliseconds as a function of the input length. CRC-32 provided the best performance per hash, followed by the Pearson hashing algorithm. All hash algorithms except for MD5 provided similar performance for small input sizes. The slower performance of the FNV-1a algorithm is due to performing multiplication with the prime number 0x1000193 for every byte, whereas Jenkins requires an XOR on every byte, in addition to shifts and additions. The SDBM algorithm provided performance similar to FNV-1a. The stepped nature of Lookup3's performance is explained by the block-wise system it uses; time per hash increases every twelve bytes as the algorithm is required to perform another mixing step, which requires six executions of the rotation function and six subtraction, XOR, and addition steps. MD5, the sole cryptographic hashing function tested, required far more time per hash than the other algorithms, even for the shortest input sizes. The poor MD5 performance for small inputs is the result of MD5's padding system, which extends partial blocks to a full 512 bits. Like Lookup3, the time required stays fairly constant during the block, but jumps when a second block is added, as each additional block requires another sixty-four rounds of mixing. For larger inputs, there is a difference of about 2 times for the faster algorithms (CRC-32, Pearson) compared to the others.

Due to the memory constraints of the ATmega328P, the compiled program size and global variable SRAM usage were recorded for all algorithms on the Arduino, shown in Fig 2 and Fig 3. These statistics are provided by the Arduino IDE after compilation, using the avr-size utility. Compiled program size is based on the compiled binary and includes the Arduino bootloader and initialized data, while the SRAM usage includes initialized and uninitialized global variables. Algorithms with particularly high global variable SRAM usage, ranging from 23% for the Pearson algorithm to 60% for CRC-32, were considered as candidates for optimization.

B. *PIC24 Results*

The same test benchmark as the ATmega328P was used with a randomly generated input message of variable byte length processed through each hashing function while measuring the time taken over 1000 executions. Two 32-bit hardware timers of elapsed processor time in ticks gave an accurate running time of each algorithm. Most of the selected algorithms, with exception of MD5, did not require extensive RAM, but did require 16-bit arithmetic and logical operations and were as such much faster running on a 16-bit chip. The effective instruction speed of the PIC24 was 16 MHz, about the same as the Arduino, so differences in performance are related to their 8-bit versus 16-bit architectures.

The PIC24 results are shown in Fig 4 and Fig 5. For most hash functions, the average time per hash was lower on the PIC24, due primarily to better support for 32-bit data types and more available SRAM. CRC-32 and SDBM provided the best performance, with FNV-1a and Jenkins only marginally

TABLE II
HASH FUNCTIONS

Function	Description
CRC-32 [17]	Cyclic Redundancy Check ; 32-bit output. Outlined in RFC 1952 for use in the GZIP Utility.
FNV-1a [18]	Developed in 1991 ; variable-length output. Used in a wide variety of applications, including DNS servers and search indexing.
Jenkins [19]	Developed by Bob Jenkins ; 32-bit output. Used in Perl 5.8.0 as the internal hashing algorithm.
Lookup3 [20]	Developed by Bob Jenkins in 2006 ; 32-bit output. Specifically designed for hash table lookup.
MD5 [1]	128-bit output ; Formerly used cryptographically, still in widespread use non-cryptographically.
Pearson [21]	Developed by Peter K. Pearson in 1990 ; 8 bit output ; chained to produce 32-bit output. Uses 256-byte lookup table.
SDBM [22]	Developed by Ozan Yigit in 1989 as part of SDBM, an open-source database software. Also used in Berkeley DB.

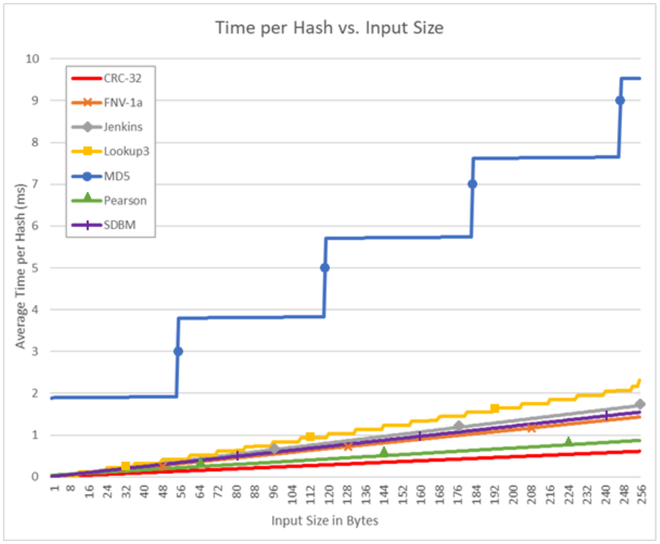

Fig. 1. Arduino Performance of Hash functions

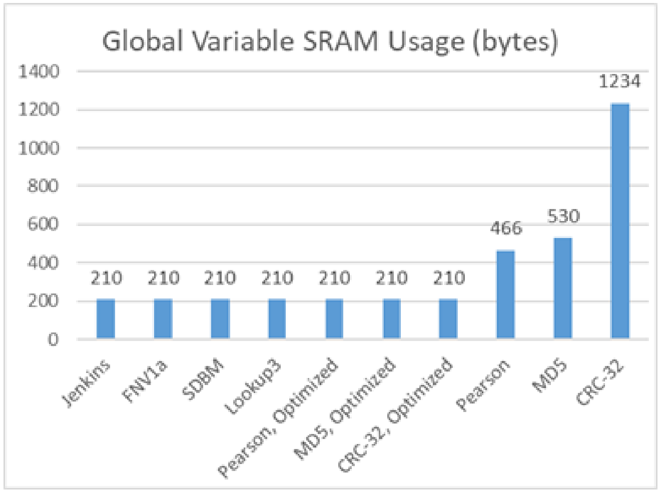

Fig. 3. Global Variable Memory Usage for Arduino

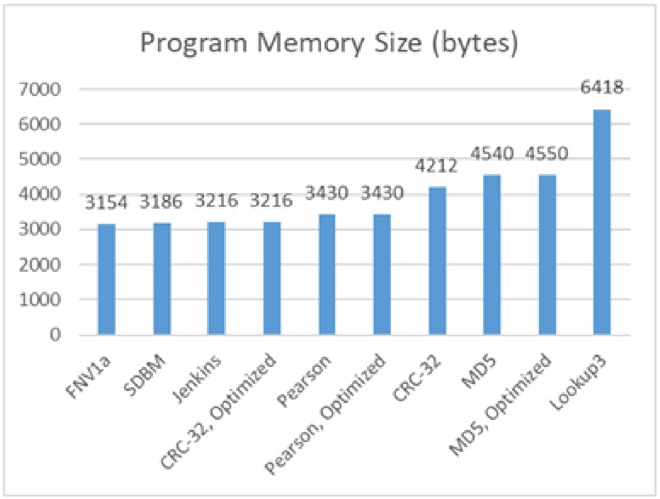

Fig. 2. Compiled Program Size for Arduino

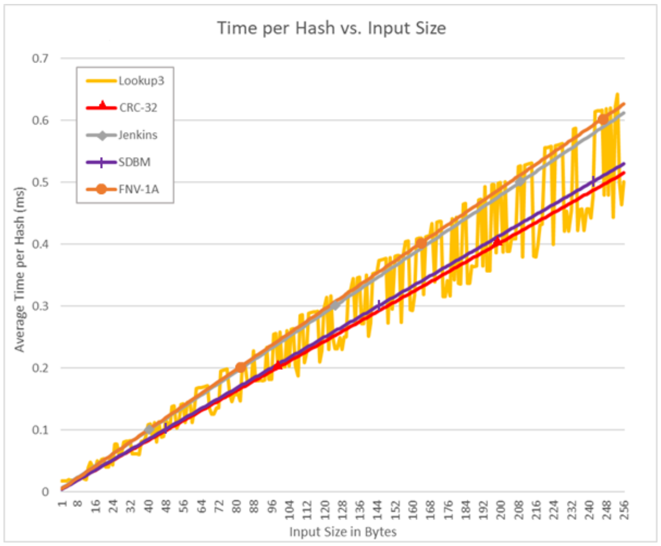

Fig. 4. PIC24 Performance for Lookup3, CRC-32, Jenkins, SDBM, and FNV-1a functions

slower. Lookup3 provided performance that spiked between being faster than CRC-32, and slower than FNV-1a; this is the result of Lookup3 using a 12-byte block system and use of fall-through switch cases dependent on the number of bits in the last block. The Pearson and MD5 algorithms were major outliers, both having worse performance on the PIC24 than on the ATmega328P. The Pearson hash algorithm had an average time per hash of more than double the next slowest PIC24 function (FNV-1a) and the ATmega328P Pearson implementa-

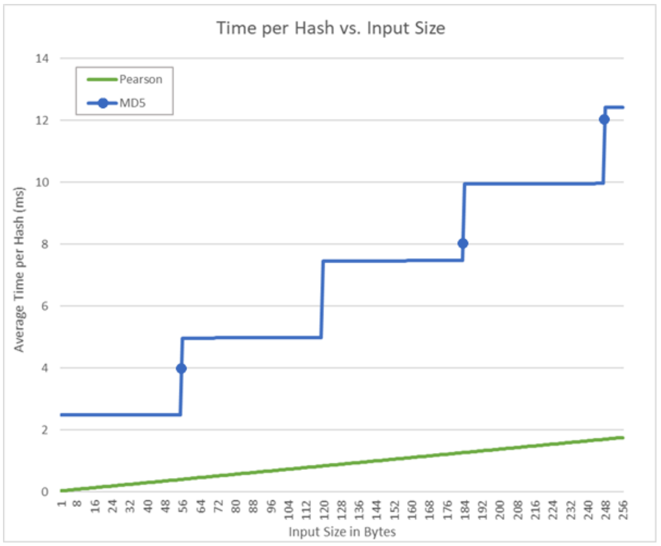

Fig. 5. PIC24 Performance for Pearson and MD5 functions

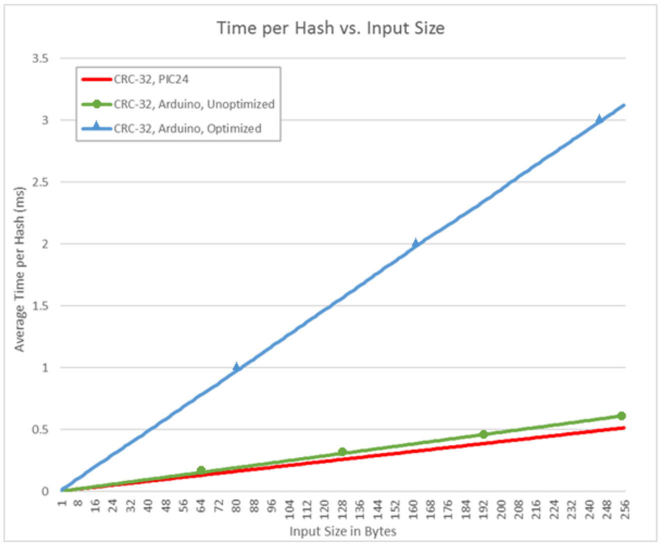

Fig. 6. Comparison of CRC-32 Performance on Arduino and PIC24

tion. The PIC24 MD5 implementation was approximately 30 percent slower than the ATmega328P MD5 function. Given that the source code for both implementations was identical, this performance difference must be related to differences in compiler optimization and processor execution.

C. Memory Optimizations

Some functions were memory optimized to reduce SRAM usage on the ATmega328P with a trade-off on execution time. Algorithms that rely on arrays of precomputed values such as the CRC-32, MD5, and Pearson hashing functions can occupy large percentages of the available SRAM, making practical use difficult when considering the memory requirements of the program beyond the hash function implementation. The CRC-32 precomputed table for example, is an array of 256 32-bit unsigned integers of total size 1024 bytes, leaving only 1024 bytes of SRAM for other program variables.

Two distinct strategies were used to minimize memory usage. The first strategy employed was to replace precomputed arrays with a function that computes the required value at runtime for a given input. This is done with the optimized version of CRC-32, replacing the array of 256 32-bit constants with a function that calculates individual CRC constants on the fly. This increased the average execution time of the CRC-32 algorithm by approximately 6 times, but is preferable in many cases to using 1024 bytes of SRAM. Fig 6 shows similar performance between the PIC24 and the Arduino for the CRC-32 algorithm when using a constant array, but heavily decreased performance when using the memory-optimized version. This method could also apply to the precomputed array of the floor of sines used in MD5, but implementation is complicated by the need for 64-bit math support. In this case, a second memory optimization strategy was used, shifting the MD5 precomputed array to the program memory. This is supported by the avr/pgmspace library included in avr-libc, which allows constants to be stored in the 32 KB flash program memory and accessed as needed [23]. Accessing from flash has slower read times and requires extra cycles per instruction to load data compared to SRAM. However, when compiled using the default Arduino toolchain of avr-gcc and avr-g++, the memory-optimized version of the MD5 algorithm performed slightly better than the original, with the performance difference increasing marginally for each 512-bit block. The 256 byte lookup table used in the Pearson hash function was similarly moved to the program memory, which had a minor negative impact on performance. A performance comparison of the optimized and non-optimized functions is shown in Fig 7. These results demonstrate that the trade-off of moving the lookup tables to flash versus SRAM is a reasonable strategy.

No memory optimization was performed on the PIC24 code, as the increased size of the SRAM (32 KB) was enough to allow the required constant arrays to be stored in memory.

IV. Conclusions and Future Work

In conclusion, many popular non-cryptographic hashing algorithms can be effectively implemented on microcontroller hardware. The contribution of this work is the experimental evaluation of the algorithms on various platforms, and the insight that relative performance is platform-specific. Analysis indicates that choice of a fast hashing algorithm is hardware and compiler dependent, as some hash functions, such Pearson and MD5, are faster on the ATmega382P, while SDBM provided greater performance on the PIC24. CRC-32 was found to be fastest on both platforms, but memory use optimization of the algorithm impacted performance heavily on the ATmega382P. However, memory optimization was shown to be effective in reducing the footprint of the MD5 and Pearson hash functions at minimal cost, and actually increasing the average time per hash performance of the MD5 algorithm. For applications where a hash function is needed frequently, such as a hash table or index system, the simpler

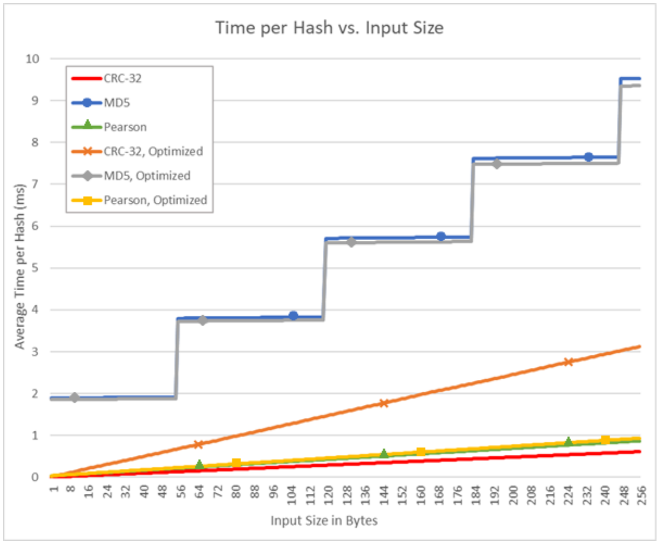

Fig. 7. Comparison of Arduino Performance for Optimized and Non-Optimized Functions

Jenkins, SDBM, and FNV-1a algorithms have consistently good performance across both test platforms. Future work will expand the scope to include more hash functions, investigate the cause of the outlier results such as MD5 on PIC24, and perform additional analysis to identify optimal hashing algorithms for other platforms.

References

[1] R. L. Rivest, “The MD5 Message-Digest Algorithm,” Internet Requests for Comments, RFC 1321, April 1992. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1321.txt

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] Y. Qin, Q. Z. Sheng, N. J. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos, “When things matter: A survey on data-centric internet of things,” *Journal of Network and Computer Applications*, vol. 64, pp. 137 – 153, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516000606

[4] G. J. Pottie and W. J. Kaiser, “Wireless Integrated Network Sensors,” *Commun. ACM*, vol. 43, no. 5, pp. 51–58, May 2000. [Online]. Available: http://doi.acm.org/10.1145/332833.332838

[5] C. Tan, B. Sheng, H. Wang, and Q. Li, “Microsearch,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 4, pp. 1–29, Mar 1, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1721709

[6] S. Fazackerley, E. Huang, G. Douglas, R. Kudlac, and R. Lawrence, “Key-value store implementations for Arduino microcontrollers,” in *IEEE 28th Canadian Conference on Electrical and Computer Engineering*, 2015, pp. 158–164. [Online]. Available: http://dx.doi.org/10.1109/CCECE.2015.7129178

[7] J. M. Hughes, *Arduino: A Technical Reference*, 1st ed. O’Reilly Media, 2016. [Online]. Available: http://lib.myilibrary.com?ID=922911

[8] T. Schubert, A. D’Ausilio, and R. Canto, “Using Arduino microcontroller boards to measure response latencies,” *Behavior Research Methods*, vol. 45, no. 4, pp. 1332–1346, Dec 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23585023

[9] J. A. Arizaga, J. de la Calleja, R. Hernandez, and A. Benitez, “Automatic control for laboratory sterilization process based on arduino hardware,” in *IEEE 22nd International Conference on Electrical Communications and Computers*, 2012, pp. 130–133. [Online]. Available: http://ieeexplore.ieee.org/document/6189895

[10] M. S. Shah and P. B. Borole, “Surveillance and rescue robot using Android smartphone and the Internet,” in *IEEE International Conference on Communication and Signal Processing*, 2016, pp. 1526–1530. [Online]. Available: http://ieeexplore.ieee.org/document/7754413

[11] “Pic24fj1024ga610/gb610 family datasheet,” 2015. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/30010074e.pdf

[12] “Explorer 16/32 development board datasheet,” 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/40001854A.pdf

[13] C. Estebanez, Y. Saez, G. Recio, and P. Isasi, “Performance of the most common non-cryptographic hash functions,” *Software: Practice and Experience*, vol. 44, no. 6, pp. 681–698, Jun 2014. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/spe.2179/abstract

[14] M. A. AlAhmad and I. F. Alshaikhli, “Broad view of cryptographic hash functions,” *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 4, p. 239, Jul 1, 2013. [Online]. Available: https://search.proquest.com/docview/1471054558

[15] J. Balasch, B. Ege, T. Eisenbarth, B. Gérard, G. Zheng, T. Güneysu, S. Heyse, S. Kerckhof, F. Koeune, T. Plos, T. Pöppelmann, F. Regazzoni, F.-X. Standaert, G. V. Assche, R. V. Keer, L. van Oldeneel tot Oldenzeel, and I. von Maurich, “Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices,” ser. Lecture Notes in Computer Science, vol. 7771. Berlin Heidelberg: Springer-Verlag, 2013. [Online]. Available: http://hdl.handle.net/2078.1/129985

[16] K. J. Choi and J.-I. Song, “Investigation of feasible cryptographic algorithms for wireless sensor network,” in *8th International Conference on Advanced Communication Technology*, vol. 2. IEEE, 2006, p. 1381. [Online]. Available: http://ieeexplore.ieee.org/document/1625834

[17] L. P. Deutsch, J.-L. Gailly, M. Adler, L. P. Deutsch, and G. Randers-Pehrson, “Gzip file format specification version 4.3,” Internet Requests for Comments, RFC Editor, RFC 1952, May 1996, http://www.rfc-editor.org/rfc/rfc1952.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1952.txt

[18] G. Fowler, L. Noll, K.-P. Vo, and D. Eastlake, “The FNV Non-Cryptographic Hash Algorithm,” Working Draft, IETF Secretariat, Internet-Draft draft-eastlake-fnv-03, March 2012, http://www.ietf.org/internet-drafts/draft-eastlake-fnv-03.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-eastlake-fnv-03.txt

[19] J. Hietaniemi, “Perl 5.8.0 perldelta,” 2002. [Online]. Available: http://search.cpan.org/dist/perl-5.8.0/pod/perldelta.pod#Performance_Enhancements

[20] B. Jenkins, “Lookup3.c,” 2006. [Online]. Available: http://burtleburtle.net/bob/c/lookup3.c

[21] P. Pearson, “Fast hashing of variable-length text strings,” pp. 677–680, Jun 1, 1990. [Online]. Available: http://dl.acm.org/citation.cfm?id=78978

[22] Y. Ozan, “Sdbm.bun.” [Online]. Available: http://www.cse.yorku.ca/~oz/sdbm.bun

[23] “avr/pgmspace.h: Program space utilities,” 2016. [Online]. Available: http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html