A Reconfigurable Arduino Crypto FPGA Shield

Ryan Bornhorst
Gomathy Venkata Krishnan
Meiqi Zhao
Dustin Schnelle

Version 1.2 1/24/18

Table of Contents

Background <Dustin>

Inexpensive open hardware platforms, such as the popular Arduino, have had a huge impact on the embedded systems industry. Especially in the world of the Internet-of-Things (IoT). Arduino's open-source microcontroller platform has become popular to both hobbyists and educators because of its easy-to-use hardware and software interface. The low cost of the Arduino platform also helps. As Arduino has become more popular in academics, the amount of documentation for the Arduino platform has greatly increased. There are already two course at the University of California that teach Arduino [1], its programming environment, and interfacing principles for an IoT specialization.

With the massive growth of the IoT the world is changing, allowing for the innovation of new designs and embedded devices within the home. This growth of devices depending on a connection to the internet means that we need to focus on the security as a first class component when communicating sensitive data. It's becoming increasingly necessary to ensure the security of embedded systems connected to the network, so adding cryptography to the functionality is important.

One solution to the problem is using an FPGA shield to perform the encryption and add to the functionality of the Arduino. The current Arduino FPGA shields on the market however don't guarantee high-assurance of the software, firmware, and hardware. High-assurance is needed for the embedded system to provide objective evidence of the system's correctness and security. Usually this evidence comes from a hand-written test bench, but we are looking to obtain this evidence from formal assurance or what is known as formal methods.

Behavioral and Non-Behavioral Requirement

- The system must be affordable and be built for under $100.
- The system's API must interface the FPGA via microcontroller.
- The system must exhibit high assurance to provide system correctness.
- The system must be high performance by being 10x faster than OpenSSL current benchmarks.
- The system should use model checking as a formal assurance technique.
- The system should use implementation tools to synthesis, validate, and verify cryptographic systems.
- The system should be built on an Arduino compatible shield.
- The system should be built from scratch.

[1] http://online-journals.org/index.php/i-jep/article/view/6845/4454

Objective <Gomathy>

- Gain expertise in Arduino programming.
- Gain expertise in FPGA programming especially for a specific FPGA shield.
- Design a custom Arduino FPGA board.
- Learn to use cryptographic libraries to implement high-assurance algorithms.
- Learn how to create APIs.
- The Arduino crypto FPGA shield should be dynamically reconfigurable.
- Learn to use Cryptol, SAW, and NaCl libraries.
- Learn how to create a PCB and interface the Arduino board with an FPGA embedded shield.

Deliverables <Ryan>

Delivered by Capstone Deadline

- *Proposal*: Document describing the background, motivation, and initial design requirements for implementing a Crypto FPGA Shield for the Arduino.
- *Test Plan*: This document will be used to verify that all of our system requirements can be met by identifying required testing instrumentation as well as test results.
- *Code Repository and Wiki*: All project documentation including schematics, source code, and general information will be located here.
- *Crypto FPGA Shield*: System Verilog code will be generated from Cryptol software provided to us from Galois. That code will then be uploaded to an FPGA shield. An Arduino will then be able to interact with the FPGA shield through an API.
- *Wrapper Cryptographic Software Library*: A variant of an AES library, such as Sodium, will be used within the Arduino development environment to configure the FPGA shield.
- *Demonstration*: Benchmarks and test applications will be used to demonstrate the use of the API as well as validate our requirements for "formal assurance."

Time Permitting or Future Deliverables

- *Stretch Goal 1*: Our team would like to contribute our software library to the Arduino website and gain visibility to it by publishing periodic updates and announcements on related forums.
- *Stretch Goal 2*: Design a daughterboard containing an FPGA shield from scratch that can interface with an Arduino, use our software library, and meet all of the design requirements within this document.

System Requirements <Meiqi>

-

Marketing Requirements

- The shield should design should be compact.
- The system should be plug and play.
- The system should be low cost while maintaining high functionality.
- The system will consume low power.
- There will be an online forum that supports the consumers and gives them constant updates about the device and features.
- The system will cater to the requirements of the cybersecurity market and will have features that will be constantly updated.
- Future software updates will remain compatible with the hardware and the firmware used on the system.
- The system will be durable within its intended operating environment.

Engineering Requirements

Performance

- The system should guarantee “formal assurance.”

Functionality

- The Arduino environment and FPGA will be able to interface with each other.
- The system should have a reset (software or hardware) incase it hangs.
- The system API will be able to use at least 4 crypto algorithms.
- The FPGA should be chosen in a way that it will be dynamically reconfigurable.

Economic

- The total cost and manufacturing cost should not exceed $100.

Energy

- The system will operate for at least 5V or 3.3V logic.
- The system will function using the embedded system power input.
- The system will operate indefinitely while plugged in.

Legal

- The system will be 100% open source.

Maintainability

- Future software updates will remain compatible with the hardware and the firmware used on the system.

Manufacturability

- The custom board should be able to easily interface with the arduino headers.
- The custom board should be manufactured with readily available parts.

Reliability and Availability

- The product will have frequent software updates to ensure that it works as expected.
- The product will be operational 100% of the time.
- Future hardware or firmware updates will be communicated to the consumers months in advance.

Social and Culture

- The development team should contribute a library on the Arduino website.
- The team will publish announcements on related blogs and forums to gain visibility for the software library.

Usability

- Users of the system should be able to learn 80% of its functionality within 2 hours.
- The system API will be well documented.
- The API will be user friendly and can be subject to refactoring to enhance functionality at any time.
- Documentation for the hardware, software, and firmware will be provided.

System Component Breakdown (Block Diagram) <Dustin>

Software Flowchart <Ryan & Gomathy>

Testing/Debugging <All>

Feel free to add suggestions

Appendix A - Bill of Materials <Gomathy>

Sno	Product	Quantity	Cost/Piece	Total Cost
1	Papilio One 250k	2		
2	Mojo v3	2		
3	Spartan 3E	1		
4	Spartan 6	1		
5	Arduino Mega	2		
6	PCB	2		
7	Soldering tools:	1		

	Liquid solder, solder wick, reflux			
8	M/M, F/M, F/F header pins or wires	1		
9	Heat sink	2		

Appendix B - Gantt Chart <Dustin>

Appendix C - Mnemonics

Appendix D - Markdown for hyperlinks(Oxygen)

SAW: https://galois.com/project/software-analysis-workbench/

Appendix E - Code Reference <Meiqi>

Appendix F - Glossary <Gomathy>

High Security - Using cryptography to prevent unauthorized access to digital information. Data integrity and authenticity.
Formal Assurance/Formal Methods - Needs to be reliable, 99.999% functional over a legitimate timespan. Library should be functioning for all possible inputs. No loss of data from memory. API should be functional and work as desired. Use a model checking tool (Formal Methods technique) like SAW for exhaustive logic based testing. Cryptol does exhaustive testing and proves it mathematically.
Crypto Algorithms - Mathematical algorithms, usually implemented in software, that are able to encrypt or decrypt data as a measure of security.
SAW (Software Analysis Workbench) - Formal verification software that is primarily used to verify cryptographic algorithms.
AES (Advanced Encryption Standard) - Software standard used implement reasoning, performance and accuracy.
Threat Modeling - Optimizing security by identifying vulnerabilities and defining countermeasures to prevent threats to the system.
Model Checker - Technique for automatically verifying correctness of all possible states within a system.

High Performance - Take reference benchmark, small program or case study, check before and after loading using the Sodium wrapper (Sodium wrapper dictates benchmark specs). Motivation for this is to exceed the processing speed of the original software benchmarks.
Random Number Generator (RNG) - An algorithm that generates a random number between some specified minimum and maximum value.
Hash Functions - A function that verifies that input data maps to a given hash value. This value is usually stored in a hash table that links the input to the corresponding hash value.
Ciphers - It is an algorithm in cryptography for performing encryption and decryption. To encipher or encode is to convert information into cipher or code.
Symmetric Ciphers - In a symmetric cipher(also known as secret key), the key that deciphers the cipher text is the same of can be derived from the key enciphers that clear text. Ex. AES and DES
Asymmetric Ciphers - The asymmetric cipher uses two keys : private and public that define who can view the content. These two keys cannot be derived from each other. Ex. RSA and DSA.
Digital Signature Algorithms -
API (Application Programming Interface) - A set of clearly defined methods and/or specifications for communicating between various software components.

Bibliography

[1]

M. El-Abd, “A Review of Embedded Systems Education in the Arduino Age: Lessons Learned and Future Directions,” *International Journal of Engineering Pedagogy*, vol. 7, no. 2, pp. 79–93, Apr. 2017.