Programming Microcontrollers for Random Number Generation in Quantum Computing

Love Arreborn, Nadim Lakrouz

ABSTRACT
TODO

1 INTRODUCTION

In cryptography, there are many applications for randomly generated numbers. However, the process of producing these random numbers tends to be pseudo-random, e.g. utilizing the current states of various modules. These numbers do not generate true randomness, and in order to heighten security other methods of generating these are required. In this project, we will be writing firmware for one such solution, which reads from an optical signal which has a variable amount of jitter. This optical signal will then be converted to a random string of bits through an Analog to Digital Converter (ADC), which will then be processed by a microcontroller. Said microcontroller needs firmware which can interpret the bitstream, and process it through Toeplitz-hashing [1] to produce a random number. In producing this firmware, several key considerations have to be made in order for this system to be usable in a production environment. These considerations include reliability, processing speed as well as security. To wit, two key questions will be investigated in this article:

Research question 1 (RQ1): How can the code for this firmware be optimized to utilize as few CPU-cycles as possible, while still implementing Toeplitz-hashing in a secure and reliable manner?

Toeplitz-hashing has been optimized quite well, and previous research can be utilized for this. However, there are still considerations when implementing the firmware for the microcontroller in order to optimize the code. Our goal is to attempt several implementations in order to find the most optimal implementation with the least amount of CPU-cycles.

Research question 2 (RQ2): What considerations can be made in order to ensure this firmware is reliable in a real-world scenario?

Low-level programming entails a fair amount of pitfalls, including but not limited to overflow issues, null-pointer references as well as race conditions. While defensive programming in C does mitigate the potential bugs that may occur, we aim to investigate other potential avenues for higher reliability.

We identify three preliminary bottlenecks which are out of our control, namely the USB-transfer speed of the microcontroller and the conversion speed of the ADC. Optimizing the code running on the microcontroller in order to process the data fast enough to match the USB transfer speed is paramount, and the main goal of RQ1. We begin by analyzing the limitations of the hardware in order to find the limitations we have to conform to.

[1] X. Zhang, Y.-Q. Nie, H. Liang, and J. Zhang, “FPGA implementation of toeplitz hashing extractor for real time post-processing of raw random numbers,” in *2016 IEEE-NPSS real time conference (RT)*, 2016, pp. 1–5. doi: 10.1109/RTC.2016.7543094.