Custom Hardware to Eliminate Bottlenecks in QKD Throughput Performance*

Alan Mink

National Institute of Standards and Technology, 100 Bureau Dr., Gaithersburg, MD 20899
alan.mink@nist.gov

ABSTRACT

The National Institute of Standards and Technology (NIST) high-speed quantum key distribution (QKD) system was designed to include custom hardware to support the generation and management of gigabit data streams. As our photonics improved our software sifting algorithm couldn't keep up with the amount of data generated. To eliminate this problem we implemented the sifting algorithm into our programmable chip (FPGA) hardware, gaining a factor of 50x improvement in the sifting capacity rate. As we increased the distance and speed of our QKD systems, we discovered a number of other performance bottlenecks in our custom hardware. We discuss those bottlenecks along with a new custom hardware design that will alleviate them, resulting in an order of magnitude increase in capacity of secret key generation rate.

Keywords: Quantum Key Distribution, reconciliation, privacy amplification, FPGA, printed circuit board

1. INTRODUCTION

The NIST Quantum Information program has produced a number of record setting quantum key distribution (QKD) systems. Initially an 850 nm free space QKD system [1] achieved about 1 Mb/s of sifted key. Following that, an 850 nm fiber QKD system [2] achieved a 4 Mb/s sifted key rate. Because the 850 nm wavelength limits these systems to less than 20 km, these systems are more suitable to local area network (LAN) applications. More recently a 1310 nm QKD system [3] employing up-conversion techniques to take advantage of the faster, more economical silicon single photon detectors (Si-APDs – Silicon Avalanche Photo Diodes) achieved distances of 50 km, but at lower key rates. This initial 1310 nm prototype has potential for higher key rates as well as longer distances and is suitable for metropolitan area network (MAN) applications. Designing and implementing QKD systems to send photons at Gb/s to achieve Mb/s of secure key is extremely challenging. In addition to innovations in photonics, innovations [4] were required in both high-speed digital electronics and information theory algorithm development to achieve these results.

As the speed and distance of the photonics increases, the performance and capabilities of the high-speed digital electronics and information theory algorithms need to keep up. Initially our custom high-speed printed circuit boards (PCBs) were designed to handle the generation and management of the Gb/s quantum data streams and the sifting algorithm was implemented in software. This resulted in a sifting limit of about 1 Mb/s. In addition, the communication cycles and CPU cycles used by the sifting algorithm cut into those needed by the reconciliation and privacy amplification algorithms. To get maximum performance from our software reconciliation and privacy amplification algorithms we moved them to a separate computer and used parallel processing. To alleviate this bottleneck we enhanced the design by moving the software sifting algorithm to hardware. There was enough spare logic on our programmable chip (FPGA – Field Programmable Gate Array) that controls our PCB and spare capacity on our classical communications channel to support the sifting algorithm. This resulted in a sifting capacity of about 50 Mb/s from the PCBs and eliminated the need for a second computer at both Bob (receiver) and Alice (sender) to run the reconciliation and privacy amplification algorithms at full performance.

---

* The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology

We have identified three main performance bottlenecks in our current QKD system. It's not possible to enhance our current PCB to alleviate these bottlenecks, so we are designing a next generation PCB that can. The first bottleneck is the speed of the data streams. Our current system communicates at a rate of 1.25 Gb/s and thus limits the rate at which we can send photons as well as our detection widow size. If we can operate at a higher communication speed, then we can increase our photon transmission rate and narrow our photon detection window.

The second bottleneck is storage associated with the distance between Bob and Alice. Alice must store the Gb/s quantum data stream being sent to Bob, until Bob reports back to Alice which photons it received and their basis, but not their secret value. From this, Alice develops the final sifted list. This round-trip delay translates to memory size. The larger the distance between Alice and Bob, the longer the round-trip delay and the larger the memory size needed to store the quantum data stream. Bob has a similar round-trip storage problem, since Bob must store the list of photons it received until Alice reports back to Bob its final sifted list. Bob can then cull its list to comply.

The third bottleneck is the performance of the reconciliation and privacy amplification algorithms. Our software reconciliation and privacy amplification algorithms, currently running on a modern dual CPU computer, have a capacity of about 1 - 1.6 Mb/s of secure key derived from about 2+ Mb/s of sifted key. Our error rates have been between 1% – 4%, yielding about 70% – 45% of sifted keys as secure keys, respectively. To increase the performance of these algorithms our software implementation uses threads [5] on a single computer, or processes on separate computers, to spawn a number of lightweight parallel tasks. Each parallel task performs the complete set of reconciliation and privacy amplification algorithms on disjoint parts of the sifted stream. A separate task accesses and parcels out the sifted bits from the PCB. Another task collects the privacy amplified bits for distribution to applications and acts as our session key manager. Each reconciliation and privacy amplification task is categorized as coarse grain computation because they require large amounts of computation. Coarse grain computation is known to execute efficiently in a parallel processing environment and in our limited experiments on a few computers we have seen linear speedup. Thus as the sifted rate increases, one solution is to use multiple computers for parallel processing. Although this is a relatively easy solution for a research environment, it is cumbersome for practical deployment as the number of computers grows and it also adds additional load to the LAN being used. An alternative to software parallelization on multiple computers is to implement the reconciliation and privacy amplification algorithms in hardware along with our existing sifting algorithm as well as the generation and management of the high-speed quantum data streams. This requires a significant re-engineering of these algorithms to adopt into an FPGA environment.

We discuss our next generation PCB design and the solutions we developed to resolve these three bottlenecks.

2. ENHANCED PCB CONFIGURATION

Our current PCBs use external SerDes (parallel to serial converter) chips that have a fixed serial speed of 1.25 Gb/s, see Fig. 1. The newer generations of FPGAs include a set of programmable transceivers that can be configured as SerDes, thus allowing us to eliminate the two SerDes chips on our PCBs, see Fig 2. One of the programmable options is the serial data rate, which is selectable from a predefined set between 1 and 6 GHz. This is a straightforward solution to solving our first bottleneck, but the board must be designed to handle the higher speed signals. Also, the FPGA transceivers are more sensitive to noise and jitter than our current external SerDes chips and thus require a more extensive PCB design effort. Being able to program the speed of the SerDes has an additional benefit. The serial signals of high-speed SerDes use a differential, non-return-to-zero signaling convention. So without recovering the clock from the data stream, there is no way to distinguish multiple 1's or 0's in a row. This is the case for the quantum stream that we use to drive our lasers. For example, three 1's in a row at 1.25 Gb/s looks like a single long pulse to the laser. A solution, using programmable speed SerDes, is to double the transmission rate and insert a zero after each bit, thus transforming the signal to a return-to-zero convention. For our current system we built an additional board that would recover the clock and convert the quantum streams to a return-to-zero convention for full 1.25 Gb/s operations. At slower than 1.25 Gb/s transmission rates, we send multiple bits for each data unit, so we insert 0's into the data stream to implement a return-to-zero convention.

The solution to our second bottleneck is to have more memory to store the information during the round-trip time. Although the newer FPGAs have more memory than before, they don't have enough. So our solution is to add a large dedicated memory chip on the PCB. The downside is, this adds complexity to the PCB design and to the logic necessary to integrate the memory into the FPGA algorithms. To size that memory we need to determine the maximum number of bits in flight during a round-trip. Photons in fiber travel at about 5 μs/km. If we allow for a QKD distance of 250 km, which is a round-trip distance of 500 km, then that round-trip would take about 2.5 ms. Alice generates two parallel data streams, one for bit value and the other for basis value. If we increase the serial communications rate to 3.125 GHz, this means the data Alice needs to store is being generated at 6.25 Gb/s. Thus in 2.5 ms Alice must store 15.625 Mbits, which would require a 2 Mbyte memory. At a 6.25 GHz transmission rate Alice must store about 31 Mbits, which would require a 4 Mbyte memory.

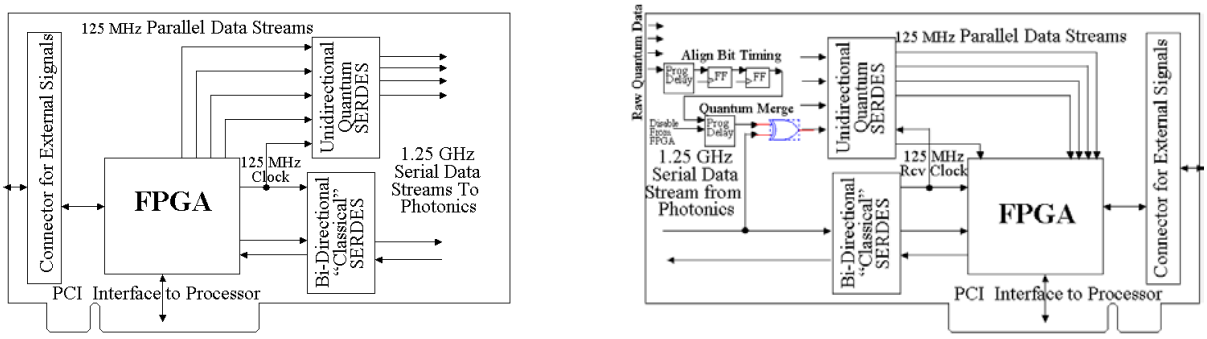

Figure 1. Current PCB Functional Block Diagrams of Alice (left) and Bob (right), four quantum channels and one classical channel.

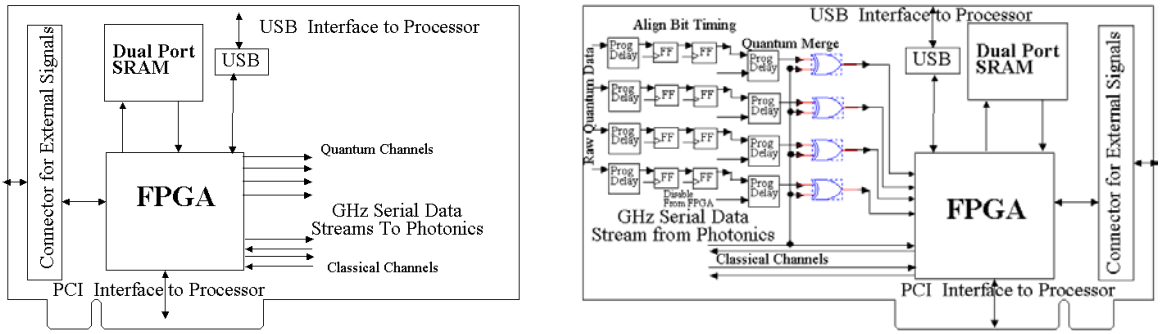

Figure 2. Next Generation PCB Functional Block Diagrams of Alice (left) and Bob (right), four quantum channels and two classical channels.

Bob must also deal with that same round-trip time to store the detection events. The maximum detection rate per APD is 10 MHz and each detection event requires 16 bits of storage. Bob's four APDs can generate detection events at a maximum of 40 MHz. During the round-trip time Bob must be able to store 100,000 detection events, which is 1.6 Mbits and would require a 2 Mbyte memory.

Our current PCB design uses dual-ported memory on-board the FPGA to store this data because independent, parallel processes fill and empty those memories. These external memories are much larger, but they still need to be dual-ported

to support these independent processes. These processes are all pipelined so that they can operate on data at wire rates. They will have to be modified to adjust to the extra, off-chip memory access delay.

The bit generation and management processes and the sifting algorithms share a single classical channel on our current PCB. The periodic Sync messages for the quantum stream are time critical, while the sifting messages are not, but they are short and easily fit in between Sync messages. Although the classical channel has some spare capacity, it's distributed in small chunks in between Sync messages. Such small chunks would make sending relatively long messages associated with reconciliation difficult due to the necessary segmentation and assembly of the messages. Also, as the quantum event detection rate increases, the spare capacity decreases. So we've added a dedicated classical channel to our next PCB for use by the reconciliation and privacy amplification algorithms. A dedicated channel allows the incoming data to be streamed (processed at wire speed), since there are no stalls in the data stream that would require the data to be buffered until reception and assembly is completed.

Our current PCB communicates with the computer via a PCI Bus (parallel) interface. Manufacturers are moving towards PCI-Express, a serial interface, and away from the PCI parallel interface. Also, this requires the PCB to be plugged into a PCI slot within the computer, something that no longer exists for laptop computers, and entails the computer be shutdown for PCB insertion. In anticipation of the diminishing access to a parallel PCI bus, we have added a USB interface to our next generation PCBs. A USB interface, a widely available interface for the foreseeable future, allows live insertion of devices external to the computer. This will provide additional ease of use and portability – desirable features for practical use. Of course this necessitates the development of a corresponding USB device driver for the computer and its counterpart on the FPGA.

3. RECONCILIATION AND PRIVACY AMPLIFICATION

Porting the reconciliation and privacy amplification algorithms to the PCB required dealing with the different way hardware and software access memory and the computational functions available for support.

Our sifting process presents our reconciliation algorithms with a set of ordered bits that don't need any distinction with either Alice or Bob. The peer reconciliation algorithms are asymmetrical, but are independent of Alice and Bob, and we designate them as Active and Passive. As in our software implementation, we use multiple parallel tasks. Parallelism within the FPGA is true parallelism whereas software parallelism on the same processor is actually interleaved execution. Thus even though the FPGA cycle time is slower than a computer, we gain increased execution performance by tailoring the algorithms to use less cycles and capitalizing on true parallelism. Another FPGA consideration is memory allocation to conserve a limited resource. The thread memory requirement of Active is more than that of Passive. Instead of allocating all Passive threads on Alice and all Active threads on Bob, we allocate a number of both Passive and Active threads on Alice and the opposite combination of threads on Bob.

Our Active and Passive threads that implement the reconciliation and privacy amplification algorithms have four main phases:

1. Estimate errors and compute total error probability; if probability low enough to process, make one correction pass, otherwise discard the data and wait for new data.
2. Estimate errors in the just processed data and compute new total error probability; if probability above a threshold A, execute another correction pass.
   Repeat phase 2 until the error probability drops below threshold A or N repetitions have been completed.
3. Estimate errors in the just processed data and compute new total error probability; if probability above a threshold B, execute a special final correction pass.
   Compute a hash code on the remaining bits and compare it against the hash code from its peer, if different discard data and return to step 1 to wait for new data.
4. Privacy Amplification.

Implementing these algorithm phases within an FPGA did not prove difficult, but unlike software that has no limits to accessing the memory data paths, special attention had to be given to FPGA memory allocation and access to the data paths to that memory. Implementing the probability computations for algorithm decision making was a problem because

no floating point, log or sqrt functionality was available within the FPGA. Through imbedded computing, which is using a processor on-board the FPGA, that functionality can be implemented, usually through 3rd party intellectual property. In addition to expense and complexity, such functionality uses a significant portion of the FPGA resources and would take a significant number of cycles. An alternative, that we used, is a profile approach based on the software algorithm execution characteristics. Using the error rate estimates that we obtain from the measurements of the sifted data and later the processed data, we project the probability estimates from a simple table lookup. This approach gives good results and is simple, fast and inexpensive to implement.

	1% err rate	2% err rate
Phase 1 Thread	2.6 ms	2.6 ms
Phase 2 Thread	6.9 ms	8.3 ms
Phase 3 Thread	2.9 ms	2.8 ms
Phase 4 Thread	3.6 ms	3.5 ms
Total Thread Time	16.0 ms	17.2 ms
Secret Key Bits Generated	49,096	45,616
1 Thread Key Rate	3.07 Mb/s	2.65 Mb/s
4 Thread Key Rate	12.2 Mb/s	10.5 Mb/s
Software Key Rate	1.6 Mb/s	1.2 Mb/s

Table 1. FPGA simulated performance and measured software implementation performance.

Table 1 lists the simulated FPGA execution times and the number of bits retained for a single Active & Passive thread pair for our reconciliation and privacy amplification phases for a sifted data set containing a 1% and 2% error rate, respectively. As the error rate increases, Phase 2 processing time increases because it takes more iterations of the data to attain the desired error rate threshold. As a consequence, the remaining number of bits decreases and results in decreased processing time in Phases 3 and 4. However, the net result is an increase in overall processing time as the error rate increases. Based on Table 1, the secret key generation capacity of a single thread is 3.07 Mb/s and 2.65 Mb/s for 1% and 2% error rates, respectively. Our initial design has four parallel reconciliation and privacy amplification threads resulting in 12.2 Mb/s and 10.5 Mb/s secret key generation capacity. These values were verified by simulation. We compare that to our dual processor computer implementation of 1.6 Mb/s and 1.2 Mb/s, respectively. These times include communication time, but do not include latency (i.e., distance). We used essentially zero distance. Communication latency (round-trip time) would equally effect the software implementation and the hardware implementation. This proposed hardware implementation would result in about an order of magnitude increase in the capacity of secret key generation rate compared to our current software implementation.

The above performance capacity is what we expect from our initial implementation of our next generation PCB design. There are a few ways in which we can further speed up our hardware implementation. First, privacy amplification is self contained and doesn't require any communication. Privacy amplification, phase 4, uses a 2-way parallel implementation. Without any parallelism, our privacy amplification algorithm would take twice as long. We can implement 4-way parallelism to further cut the execution time in half; 8-way and 16-way parallelism is possible, but gets increasingly complex to implement. A second enhancement is to increase the number of parallel threads from four to six to gain a 50% increase in performance. Six is the maximum number of threads that will fit in our target FPGA. A third enhancement is to increase the clock rate by 25%. This can easily be done, since the FPGA clocks are programmable and a roughed out design indicates that the implementation can run at this higher clock rate. Implementing these three enhancements would result in 25.9 Mb/s and 22.1 Mb/s capacity rate of secret key generation for a 1% and 2% error rate, respectively. More than double our initial target capacity.

4. Summary

Our high-speed QKD system was designed to include custom hardware to support the generation and management of gigabit data streams. As our photonics improved our software sifting algorithm couldn't keep up with the amount of data generated. As a result we were able to implement the sifting algorithm in our FPGA gaining a factor of 50 improvement in the sifting rate capacity. As we increased the distance and speed of our QKD systems, we discovered a number of performance bottlenecks in our custom hardware. We have described those bottlenecks along with a new custom hardware design that will alleviate them, resulting in an order of magnitude increase in secret key generation rate capacity. Further enhancements were also discussed that would double that capacity.

ACKNOWLEDGEMENT

This work was partially supported by the Defense Advanced Research Projects Agency under the DARPA QuIST program. I'd like to thank Joshua Bienfang for the technical discussions and support during our QKD system design as well as LiJun Ma, Barry Hershman, Hai Xu and Xiao Tang for their support.

REFERENCES

1. J.C. Bienfang, A.J. Gross, A. Mink, B.J. Hershman, A. Nakassis, X. Tang, R. Lu, D.H. Su, C.W. Clark, C.J. Williams, E.W. Hagley, and J. Wen, "Quantum key distribution with 1.25 Gbps clock synchronization," Optics Express, Vol. 12 (9), 2011 (2004).
2. X. Tang, L. Ma, A. Mink, T. Nakassis, H. Xu, B. Hershman, J. Bienfang, D. Su, R. Boisvert, C. Clark, and C. Williams, "Quantum Key Distibution System Operating at Sifted-key Rate Over 4 Mbits/s," SPIE Defense & Security Symposium, Orlando, FL, 17-21 April 2006, Vol. 6244-25, pp. 62440P-1 -7.
3. H. Xu, L. Ma, A. Mink, B. Hershman, and X. Tang "1310-nm quantum key distribution system with up-conversion pump wavelength at 1550 nm", Optics Express, Vol. 15, No. 12, pg 7247, June 11, 2007.
4. A. Mink, X. Tang, L. Ma, T. Nakassis, B. Hershman, J. Bienfang, D. Su, R. Boisvert, C. Clark, and C. Williams, "High Speed Quantum Key Distribution System Supports One-Time Pad Encryption of Real-Time Video," Proceddings of SPIE Defense & Security Symposium, Orlando, FL, 17-21 April 2006, Vol. 6244-22, pp. 62440M1-7.
5. http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html, accessed Feb 2007.