NTNU

Norwegian University of
Science and Technology

Quantum key distribution prototype

Ole Christian Tvedt

Master of Science in Electronics
Submission date: July 2010
Supervisor: Johannes Skaar, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

The candidate will continue implementing the prototype of a quantum key distribution system according to the plan presented in his project report from the previous semester. The goal is to demonstrate quantum key distribution in the laboratory setting. The sub-goals are: a) build digital-to-anolog converters + amplifiers and, together with master student Eivind Sjøtun Simonsen, demonstrate controlled interference in the fiber-optic interferometer at the rate of 200 mbit/s; b) implement the Bennett-Brassard 1984 (BB84) quantum key distribution protocol with decoy states between Alice and Bob, to the largest extent possible under the readiness of hardware. Through this work, attention should be paid to avoid known security loopholes.

Assignment given: 22. February 2010
Supervisor: Johannes Skaar, IET

Quantum key distribution prototype

Master's thesis by
Ole Christian Tvedt

26. July 2010

Supervisors:
Postdoc. Vadim Makarov
Prof. Johannes Skaar

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem description

The candidate will continue implementing the prototype of a quantum key distribution system according to the plan presented in his project report from the previous semester. The goal is to demonstrate quantum key distribution in the laboratory setting. The sub-goals are:

a) build digital-to-anolog converters + amplifiers and, together with master student Eivind Sjøtun Simonsen, demonstrate controlled interference in the fiber-optic interferometer at the rate of 200 mbit/s;

b) implement the Bennett-Brassard 1984 (BB84) quantum key distribution protocol with decoy states between Alice and Bob, to the largest extent possible under the readiness of hardware.

Through this work, attention should be paid to avoid known security loopholes.

* * *

Assignment given: 19. February, 2010
Supervisors: Vadim Makarov, IET & Johannes Skaar, IET

Abstract

This thesis covers the basics of cryptography, both classical and the newer quantum-based approches. Further, it details an implementation of a BB84-based quantum key distribution system currently under construction, focusing on the controlling hardware and FPGA-based software. The overarching goal is to create a system impervious to currently known attacks on such systems.

The system is currently running at 100 Mbit/s, though the goal is to double this as the design nears its completion. The system currently chooses encoding base, bit value and whether a state is a socalled decoy state. However, the modulator for bit encoding is not yet operational. Output for decoy state generation, however, is fully functional.

Finally, the thesis describes what steps are necessary to reach a complete BB84-based quantum key distribution system implementing decoy states.

Contents

Chapter 1

About this thesis

First, chapters 2 and 3 will discuss the background for this thesis; cryptography in general and the *how* and *why* of quantum cryptography.

Chapter 4 contains the implementation-specific details, both hardware and software. So far, only the sender module *Alice* has been built. All hardware components except a true random number generator are in place, though there are some unimplemented software modules.

Finally, chapter 5 shows a sample of the finished intensity modulator input and explains the result.

Chapters 2 and 3, and section 4.2 have been copied from the previous semester's project report (with minor adjustments), in order to make the thesis self contained. Section 3.4 is an exception, which has been added for completeness.

Chapter 2

Introduction to cryptography

2.1 The classical approach

Cryptography deals with the process of turning data (i.e. *plaintext*) into a format a third party cannot read (*ciphertext*) without the use of some other secret (a *key*). Up until modern times data has usually referred to pure text, but the principles can just as easily be applied to any encoding of information.

The need for passing messages between parties in a confidential manner may seem obvious. At least enough so that simple encryptions can be found in comic books and other children's magazines to be cracked by people with no special skills in the field. The *how* of it may not be as obvious as the *why*, however.

The encryption variant usually found in puzzles is a form of *substitution cipher*, in which the original message can be retrieved by substituting each character in the ciphertext with another one. The secret the sender and the receiver shares is *which* character to replace with which. One could say that the key is a jumbled alphabet.

This form of encryption has its weaknesses. For example, it is known that in English the most frequent character is *E*. By looking at the ciphertext, one can then assume that one of the most frequent characters are to be replaced by *E*s. Follow this method to its conclusion, and you have found the plaintext without knowing the key. One could make this process harder by removing spaces and punctuation in the plaintext before encoding, but it would still be possible break the encryption. Trying to decrypt other types of signal (such as linear pulse code modulated audio) in this manner would certainly require a computer, but the principle remains the same. If you know something about what the decoded message should look like, you can use statistical methods to find out what it is.

There are many forms of encryption, but until quantum cryptography came along, they all had one thing in common: They were based on problems that were computationally *hard* to solve, but not impossible. For example, to find out what data is being transmitted when the *https*-protocol is in use (often used in on-line shopping and personal bank account management), you would need extremely powerful hardware and many years of computing. In other words, you would have to record the data and analyse it afterwards.

There is one form of conventional encryption that is provably unbreakable: *one time pads* [1] (and derivatives). It is a variant of the substitution cipher where each character in the message is modified according to the character at the same position in the key. It has not seen much use, for practical reasons. The key has to be at least as long as the message, can only be used once (hence the name) and has to be completely random. This means that the sender has to share a very long key with the receiver, possibly even before knowing how long the message will be. If a person were to use this for his outgoing e-mail, he would first have to give all his recipients a unique key (one each) in a guaranteed eavesdropper-proof way, for example in person on a CD or a sheet of paper.

The one time pad is however very useful in quantum-based encryption schemes, as we shall see.

2.2 The quantum approach

Quantum encryption systems base themselves on fundamental physical quantum-level principles that ensures data can not be duplicated. The data sent by the transmitter cannot be read by an eavesdropper and simply passed on to the intended recipient. Any such attempt will be easily detected.

The most common implementation is known as *quantum key distribution* (or *QKD*). In such a system, the message itself is not passed over the secure channel. Instead, one part generates a completely random key that is encoded in some form of quanta (or *qubits*) that cannot be copied (such as the polarisation of a photon), and sends that across the secure channel to the recipient. This key can then be used for a one time pad encryption over a classical channel, such as an ethernet link.

QKD is the principle used in this project, and the details will be discussed in the next chapter.

2.3 A brief history of quantum encryption

Stephen Wiesner published a proposal for a mechanism that could be used for quantum encryption in 1983 [2]. He proposed that bit values could be encoded in "conjugate observables", which are possible properties for a bit carrier (such as a photon) were either one can be measured, but not both. This means that if you measure the wrong property, you have effectively destroyed the information. An eavesdropper would need to know what property to measure to be able to decode the information.

In 1984, Charles H. Bennett and Thomas J. Watson proposed a protocol based on Wiesner's work that would allow one legitimate user to generate and send a key to another [3]. Their protocol, often referred to as *BB84*, ensures that an eavesdropper will be detected with an arbitrarily high probability, assuming an ideal system.

In 1991, Artur Ekert proposed a mechanism based on quantum entanglement [4]. The principle is to generate pairs of entangled photons, and pass one to each legitimate user. If

they both measure one property of the photon, they will always get the same answer. Thus they can use it for key generation. This method is often referred to as *E91.*

Other relevant research include G. Brassard and L. Salvail's *cascade protocol* [5], which is a method of ensuring that the two legitimate users' keys are identical over a public channel (known as *information reconciliation*), and several different methods of *privacy amplification* [6][7][8][9] (also known as the *leftover hash-lemma*), in which the key is hashed and slightly shortened to generate a new unique key.

Chapter 3

Theory

3.1 Types of quantum key distribution

There are several ways of transmitting qubits from a sender (typically referred to as *Alice*) to a receiver (*Bob*), while at the same time measuring the interference from a potential eavesdropper (*Eve*). They can be separated into two classes (as described in section 2.3):

Entanglement based systems relies on how two separate particles can share a quantum state. If Eve tries to intercept and measure one particle, she will also alter the state of the other, and thus reveal her presence. E91 is an example of such a system.

Prepare-measure systems are based on transmitting data by several conjugate encodings, and afterwards comparing how the receiver measured the incoming bits with how they were encoded. Using this method, one can also calculate how much of the transmitted data has been intercepted. The BB84 protocol is an example of such a system, and is what the design in this project is based on.

3.2 BB84

In the BB84 protocol, each bit in a random key generated by Alice is encoded in one of two possible orthogonal bases. In the first base (+) the bit value 0 is encoded as a vertically polarised photon, and 1 is encoded as a horizontally polarised photon. In the second base (x) 0 is encoded as a 45° polarised photon, and 1 as a −45° polarised photon. Since these bases are orthogonal, measuring in the wrong base will give you a completely random result.

Alice chooses which base to transmit in randomly. At this stage Bob doesn't know which base to measure in, so he simply chooses one randomly too. If he chooses the correct base (the same one Alice used), he will receive the bit intended by Alice. If he chooses the wrong base, there is still a 50 % chance he chose the right one. In other words, the amount of incorrectly received bits approaches 25 % as the key length grows.

After Alice and Bob are done transmitting the raw key data, they use a classical unencrypted channel to compare which bases they chose. The bits where Bob guessed the wrong

base are discarded, and the rest are kept and used as a key. They will be identical in an ideal system where no one attempted to intercept any of the bits. This process is known as *sifting*.

A short example of such a transfer is shown in table 3.1. In this case we see that Bob chooses the wrong base in 4 out of the 8 bits. In two of those cases he still got the correct result, but neither him nor Alice has a way of knowing that, since the actual key is not compared. Therefore, they are discarded along with the cases he chose the wrong bases on.

Alice	Generated key:	0	0	1	0	1	1	1	0
	Chosen base:	x	+	x	+	+	x	x	+
Bob	Chosen base:	+	x	x	+	+	+	x	x
	Received key:	0	1	1	0	1	0	1	0
	After sifting:			1	0	1		1	

Table 3.1: Example of an uninterrupted key distribution

It is worth noting that even though the BB84 protocol uses two bases of linearly polarised photons, there are other possibilities. The protocol only requires bases where the result of choosing the wrong base is random (the bases are *conjugate*). It is for example possible to use linearly polarised light for one base and circularly polarised light for the other.

3.2.1 Eavesdropping on BB84

Since the encoding bases are only known by Alice, the only way Eve can intercept photons is by guessing a base the same way Bob does. She must also make sure Bob receives something, and her best possibility is to resend what she received (which may be incorrect). This is known as an *intercept and resend* attack.

For her attack to succeed, it is not necessary for her to choose the correct base. What is important is that she chooses the same bases Bob does. Presume that she is listening in on the classical channel as well. The possibilities are listed in table 3.2.

Bob's and Eve's choices	Action during sifting	Bob's bit value
Both chose the correct base:	Kept	Correct
Both chose the *in*correct base:	Discarded	-
Only Bob chose the correct base:	Kept	Correct 50 %, incorrect 50 %.
Only Eve chose the correct base:	Discarded	-

Table 3.2: Possibilities for an intercepted bit

In the event that Bob chooses the correct base, the bit will be kept during sifting. In half of those cases, Eve will also have chosen the right base, so the bit value is correct. In the other half of the cases, there is a 50 % chance that she received the wrong bit value. In total, this will introduce a 25 % *qubit error rate* (or QBER).

By choosing to use a subset of the key for comparison, Bob and Alice can easily detect such a high QBER and abort their attempt to communicate. As you can see, QKD does not ensure communication (as Alice can prevent it). It *does* ensure that the communication is kept private.

3.3 Decoy states

The BB84 discussion assumed an ideal system. Unfortunately, quantum key distribution systems suffer from implementation difficulties. One of these are the difficulty of sending single photons for each bit. A typical implementation uses attenuated laser pulses which sends out groups of photons. The average number of photons can be adjusted by adjusting the attenuation, but keeping the variance at 0 is not trivial.

This enables Eve to perform a *photon number splitting* attack [10], where she simply intercepts some (but not all) of the photons in each pulse and allows the others to pass undisturbed. If there is only one photon in a pulse, she can block it to ensure that the bit will not be used by Alice and Bob.

Decoy states has been proposed as a way of preventing such attacks [11]. If Alice intentionally replaces some of the signal pulses with a decoy pulse of more than one photon, she and Bob will be able to compare the loss in such pulses with the loss of legitimate pulses. Eve has no way of knowing which pulses are decoy pulses (hence the name), and must therefore treat them the same as signal pulses. However, since decoy pulses has a higher average photon count, more decoy states are likely to get through to Bob than regular signal pulses. The losses should be comparable. If the losses are significantly lower for decoy states, the communication can be aborted.

The implementation under construction in this project uses such decoy states to increase security.

3.4 True random number generators

Since quantum key distribution relies on having random data available, a source is needed that is not subject to the same weaknesses found in classical cryptography. Thus, there is not much use in pseudo-random number generators, which generate deterministic but hard to predict sequences.

Hardware RNGs can generally be sorted into two categories:

1. Generators based on impossible to predict quantum effects.

2. Generators based on noise, typically thermal noise.

The first category contains systems such as those based on a radiation source and a geiger counter (where the time between clicks is random), and systems based on a partially opaque window (where photon transmission is random).

The second category is based on the caotic nature of certain phenomena, such as thermal noise in a resistor or avalanche noise in a reverse biased p-n semiconductor junction. Some of this noise is caused by true quantum unpredictability, as with the processes in category one.

Chapter 4

Implementation

4.1 Overview

A diagram of the complete implementation can be seen in figure 4.1. Note that the bus width of the DACs should be at least 8, not 2 as shown. Also, the *field programmable gate array* (FPGA) to PC communication will most likely be implemented as an ethernet link instead of USB 2.0, though the decision is not final.

Also note that the system is currently running on 100 MHz, half of the intended rate. This was done to lower the requirements of the digital to analog converter and the amplifiers. It should be possible to increase the rate once the system in functional, assuming that a very fast true random number generator is added (or the internal pseudo-RNG is duplicated).

The interferometer has been detailed and demonstrated in *Security of quantum key distribution source* [12], and this chapter will focus on the controlling circuits.

One can follow the information path through the optical system. For each bit sent from Alice, the controlling hardware does the following:

- Sets the intensity modulator to a determined level, depending on whether this is a decoy pulse or not.
- Sets the phase modulator to a determined level, based on the chosen quantum state to send.
- Pulses the main 1.55 μm laser when the modulators are stabilised.

The synchronisation laser is pulsed at regular intervals to maintain synchronisation between Alice and Bob. The two lasers are multiplexed together in a *wavelength division multiplexer* (WDM) and sent out on the transmission line to Bob.

In Bob's end, the received laser is demultiplexed and the synchronisation pulse redirected to a separate detector and input (see section 4.5). For each received bit, Bob's FPGA does the following:

- Sets the phase modulator to a determined level, based on which of the orthogonal bases the attempted measurement should be done in.

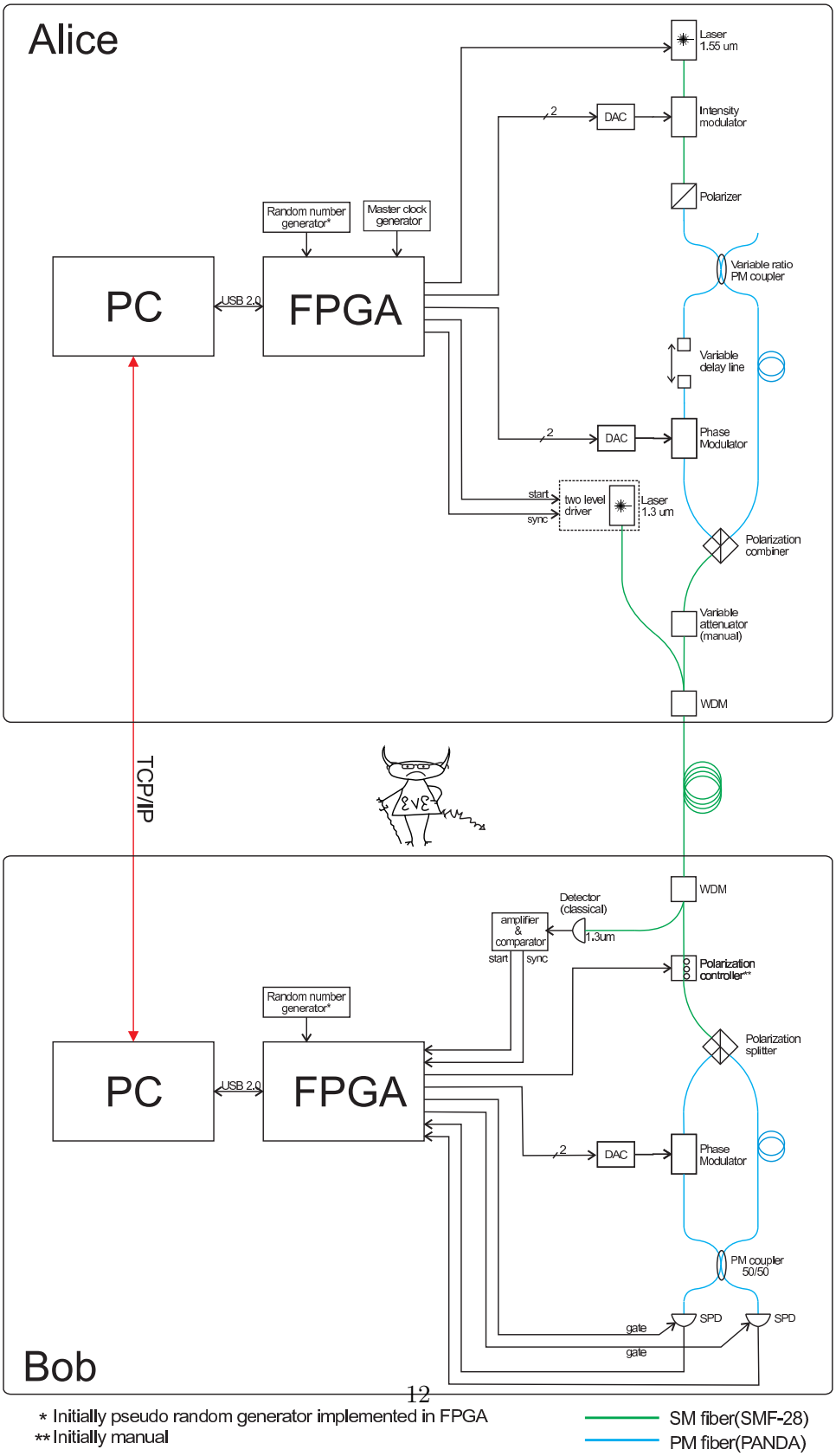


Figure 4.1: Initial hardware plan, by M. Ulianov [13]

- Gates the APDs with a sinusoidal pulse. The APDs will only detect incoming photons when there is a gate voltage.
- Listens for pulses on the APD inputs.

4.2 Communication

The BB84 protocol described in section 3.2 requires several steps of communication between Alice and Bob. Both sides have a PC communicating two-way with the FPGA on their side, and the PCs are communicating with each other through an ethernet link. The FPGAs only communicate directly through the optical one way link from Alice to Bob. The steps (shown in figure 4.2) are:

1. The software driver in the PCs are in communication with the FPGAs, monitoring the hardware while waiting for the user to initiate the sequence.
2. Either Bob or Alice initiates the communication, and lets the other part know via ethernet.
3. Alice's PC signals the FPGA on her side to initiate the key distribution.
4. The hardware is synchronised via timed pulses from Alice's side on the optical link.
5. Alice's FPGA generates and transmits the key, storing the time stamps, the bits and the randomly chosen bases to RAM. She also stores which pulses are decoy states. Bob's FPGA guesses bases randomly and stores the time stamps, the measured bits and the guessed bases to RAM.
6. Bob's guessed bases are uploaded to the PC, transmitted through ethernet to Alice's PC and downloaded to Alice's FPGA.
7. As Alice's FPGA is receiving Bob's guesses, it compares them to the actual bases used during encoding. The entries in memory where the incorrect bases were used by Bob are marked as junk. At the same time, the fraction of received bits for each intensity (decoy or not) is calculated (as Bob doesn't send the chosen base for bits he didn't receive). If the losses are significantly higher for the lower intensities, it indicates an eavesdropper is stopping all single photon signals. If so, the FPGA will signal an abortion to the PC.
8. If there is no significant discrepancy in the losses, Alice's FPGA uploads the bits and bases of the entries not marked as junk to the PC.
9. The timestamps of the bits to use are passed back to Bob. At this point, other techniques can be used to improve security, such as the information reconciliation and privacy amplification described in section 2.3.

These steps have not been finalised, and the process may change slightly in the future.

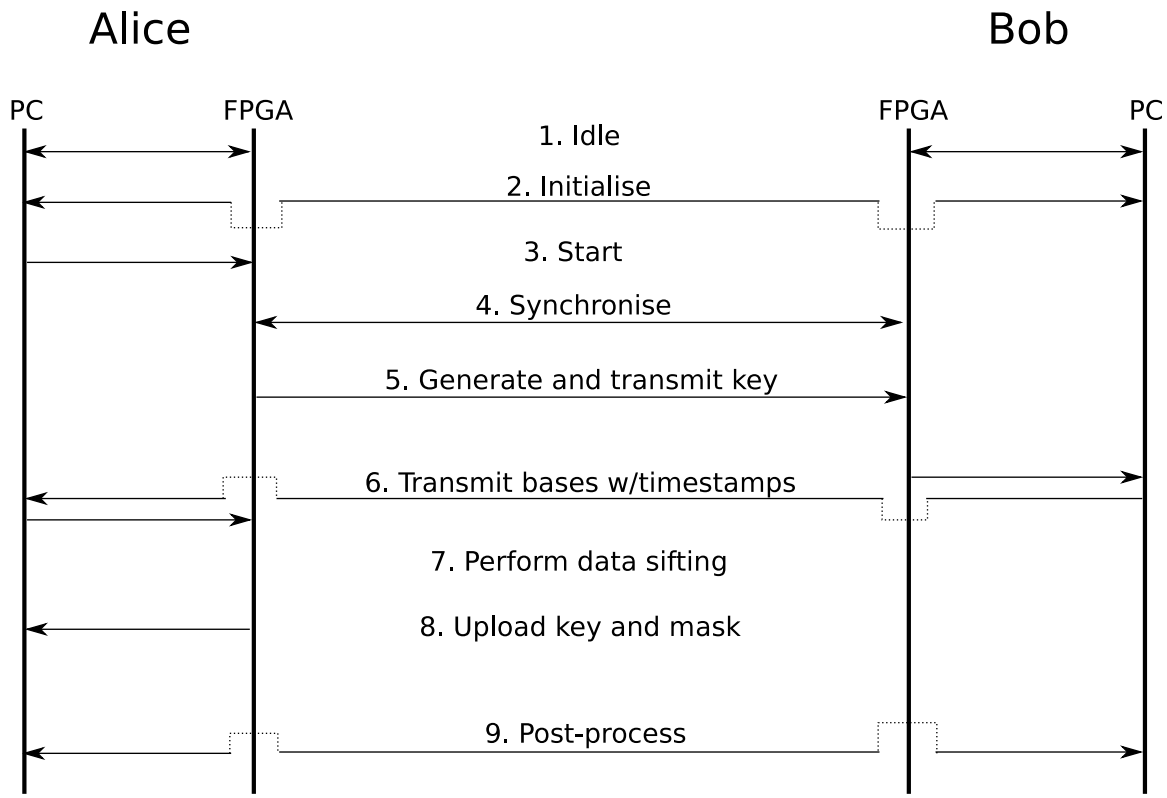


Figure 4.2: Communication overview

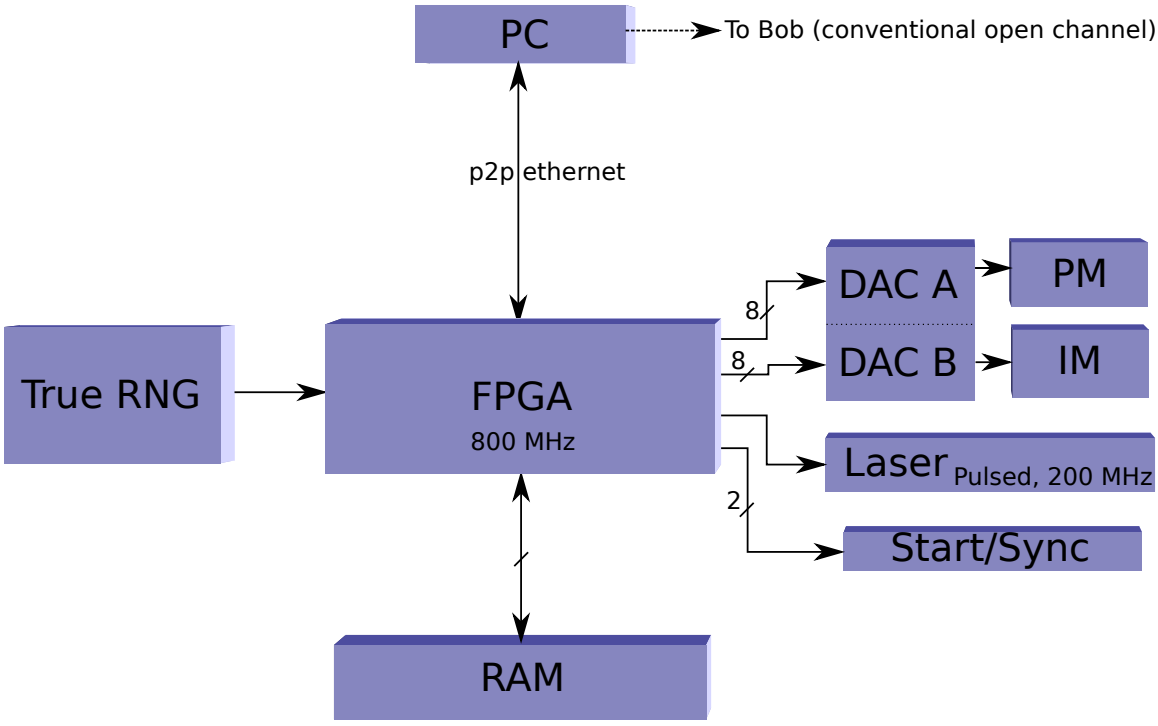


Figure 4.3: Simplified Alice side controller overview

4.3 Controlling hardware

The controlling part of the hardware in Alice and Bob will be very similar. Even though Bob only needs one DAC channel, the DAC chosen for this project (see 4.6) is a dual channel variant. The only differences are thus in the optical path, and not in the controller (though obviously the FPGA software will be unique to each part).

However, it may be prudent (for economic reasons) to choose different physical random number generators on each end, as Alice's requirements are higher than Bob's. Both run at 100 MHz, and both must choose one out of two bases for every bit. However, Alice must also control the intensity modulator in a similar fashion (depending on the ratio of decoy states) and choose a bit to encode. Naïvely, this means 200 Mbit/s for Alice (no decoy states) and 100 Mbit/s for Bob (though see section 4.8.1).

4.3.1 Alice

The main logic blocks of Alice is shown in figure 4.3. The modules and their connections are as follows:

FPGA : Altera Stratix III 3SL150. This is the main hardware controller, handling the coordination of all other units. The main code is running on 100 MHz, though faster

clocks are used internally for some tasks.

PC : The PC initiates the procedure, and acts as an intermediary between the two FPGAs during sifting. Even though the FPGA and the PC are connected through an ethernet link, this should not be a network but a direct connection.

RAM : The random access memory stores entries in the form [base] (1 bit), [decoy state] (1 bit), [bit value] (1 bit) for each bit sent. The memory address will act as a timestamp. The FPGA board is equipped with one 1 GiB DDR2 module. The communication with this cannot be implemented until the main final state machine (see section 4.4.7) is finished.

RNG : A random number generator. This simply generates a stream of random bits according to the requirements noted above. This is not a critical part of the system during development, as the FPGA can run in *test mode* and generate its own (cryptographically weak) pseudo-random bits.

Start/Sync : A 2 bit output signal triggered at regular intervals from the start of the key transmission. This controls a separate laser that is mixed onto the transmission line in a WDM, and heps Alice's and Bob's FPGAs to remain synchronised.

Laser : The laser needs to be driven by stable 200 ps pulses. To achieve the short pulses and the required stability, a hardware driver was created. The driver is controlled through a duty-cycle adjustable 100 MHz output from the FPGA that can be tuned to within 8 % (at worst).

DAC : Analog Devices AD9746 dual channel 14 bit digital to analog converter. The output is attenuated to fit the amplifiers used by the modulators.

PM : The phase modulator is used to encode each bit value in one of the two possible bases.

IM : The intensity modulator is used to produce the decoy states explained in section 3.3. Is is biased so that the maximum dampening is at 0 V input.

4.3.2 Bob

The main logic blocks of Bob is shown in figure 4.4. The modules and their connections are as follows:

FPGA : Hardware-wise identical to Alice, but with different software loaded.

PC : The PC initiates the procedure and downloads the guessed bases and received bit values. The communication is identical to Alice.

RAM : Identical to Alice, except with no decoy bit and with an extra bit to store double clicks (photon detected in *both* detectors).

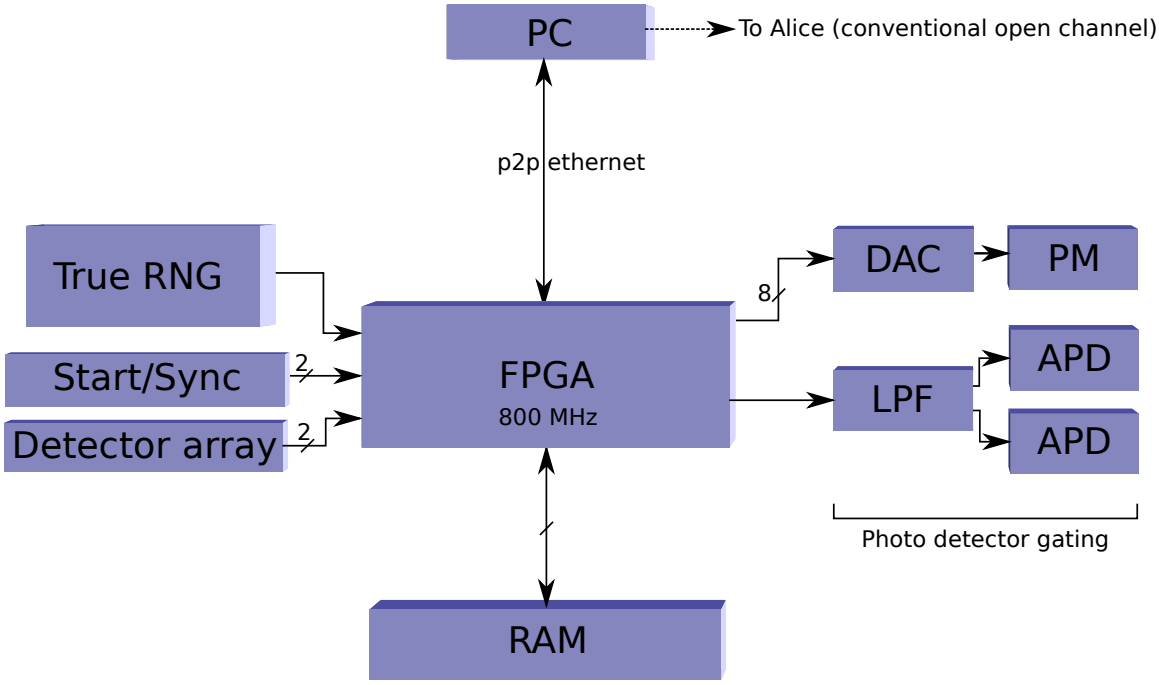


Figure 4.4: Simplified Bob side controller overview

RNG : Identical to Alice, though the requirements are lower (as noted above).

LPF : A low pass filter, to turn the digital square signal from the FPGA into a sinusoidal pulse. The input from the FPGA will be a low jitter 100 MHz signal, and the phase will be adjustable to synchronise them with incoming photons.[1]

APD : The gate of the avalanche photo diodes, which detects the incoming photons. Gating the APDs prevents them from triggering between pulses when no light should be arriving.

Start/Sync : The input corresponding to the same output in Alice. As synchronisation pulses come in, the FPGA must keep its clock in phase with them. This ensures there is no drift between the clock in Alice and Bob, so that the APDs will be gated at the correct time and the time stamps will match up during sifting. The frequency range in which the clock is allowed to synchronise is narrow, to prevent Eve from tampering with the multiplexed synchronisation signal.

Detector array : The incoming pulses amplified by the APDs.

DAC : Identical to Alice, though only one channel is needed.

PM : The phase modulator is used to choose one of the two possible bases to measure in.

[1] The 3SL150 FPGA does not support on-the-fly phase adjustment. An external solution may be necessary.

4.4 FPGA software design, Alice

4.4.1 Errors and test mode

Any part of the design where detectable errors may occur have error output flags. All these errors are passed to a dedicated error handling module (*error handler* in figure 4.5). This module has two purposes:

1. Signal the user. The LEDs on the FPGA board are by default all lit. When an error occurs, some LEDs are turned off. The pattern of lights signals what error occured. There is also a character display on the board, and the code may be extended to display error messages.

2. Abort the current key distribution on fatal errors. For example, when the RNG signals an error, its output can no longer be trusted, and the key is unusable.

All signals in the schematics prefixed with *status* are high under normal operation. If such a signal goes low, it signifies an error. Signals prefixed *statusF* are the same, but are also considered fatal. If such a signal goes low, the main state machine should abort any running communication.

Additionaly, it may be a good idea to add a module for analysing the raw data from the random number generator. There is a danger that a RNG should break silently. Running some statistical analysis will enable the system to abort should this happen. Many true RNGs already have this functionality built in, so this may not be necessary.

4.4.2 Timing calibration

Several subsystems need to be temporally adjusted in relation to each other. Due to differences in synthetisation, these adjustments may need to be redone between software revisions. To simplify this, several phase locked loops (PLLs) are used to generate separate clocks for each subsystem.

Most of the logic is clocked by one PLL (named simply *pll* in figure 4.5), including the clock output driving the DAC. This PLL also outputs a phase delayed copy dedicated to gating the DAC outputs, ensuring that the setup and hold requirements of the DAC has been met. The phase delay of this clock may be adjusted to delay the data compared to the DAC clock.

The DAC setup time is given as $t_{dbs} = 400\,\text{ps}$, and the hold time as $t_{dbh} = 1200\,\text{ps}$ [14]. At 100 MHz, we have a period $t_p = 10\,\text{ns}$. Thus, the clock-to-data delay is given by $\varphi = 1200\,\text{ps} + \frac{10000-1200-400}{2}\,\text{ps} = 5400\,\text{ps}$.

The laser is driven by a dedicated pll (*laser driver pll* in the figure), enabling adjustment of a laser pulse's position compared to the modulator inputs.

Finally, the random number subsystem (*randomiser*) is driven by a pll running at 800 MHz (*fastpll*). This enables the subsystem to generate 8 bits per clock cycle when running in test mode.

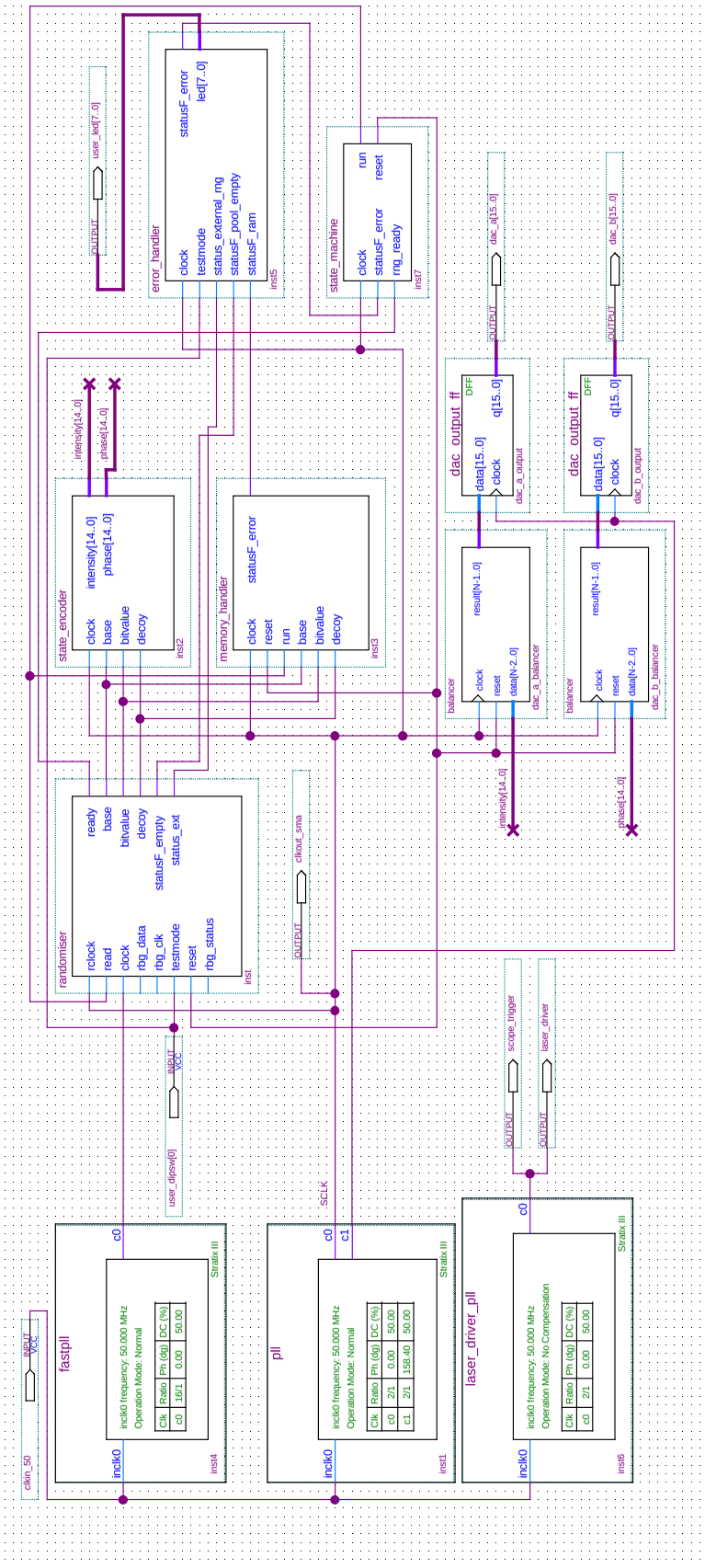


Figure 4.5: FPGA software overview

4.4.3 State encoder

The *state encoder* module takes the chosen base and bit value, and converts this in to a 15 bit unsigned value intended for the phase modulator. This part has not yet been implemented. The four different polarisations of light will correspond to four values that can be calibrated in this module.

It also sets one of two values intended for the intensity modulator, based on whether this is a decoy state or not. For decoy states, a high value (corresponding to the modulator's V_π) is chosen, thus maximising the laser pulse intensity. Otherwise, a low value is chosen, to let a low number of photons through.

4.4.4 Memory handler

The *memory handler* module receives the same input the state encoder does, in addition to a *run* and a *reset* signal. When the run signal is high, each output state is stored consecutively in RAM for later comparison with Bob.

4.4.5 Output balancer

The chosen output value so far for both modulators is a 15 bit unsigned integer. The value is then converted in the *balancer* module to a 16 bit signed integer. The balancer integrates its output values so far, flipping the input to negative if the current integrated value is positive. By balancing the values around zero like this, bias drift is prevented in the amplifiers connected to the modulators.

4.4.6 DAC output gating

The values piped to the digital to analog converter are signed 16 bit integers, encoded in 2's complement, and are gated out through a pair of flip flops (*dac output ff*, see 4.4.2).

4.4.7 Main state machine

The main FSM (finite state machine) will govern the process of key exchange by iterating through the steps. The states will be close to or identical to the ones listed in figure 4.2, most likely with some substeps.

This module will most likely not be written until the entire key exchange process has been decided upon, including communication steps with PC on each end, but has been inserted as a dummy module in the diagram.

4.4.8 PC communication module

This module is not in the diagram, but will need to be added when a detailed communication protocol has been agreed upon.

4.4.9 Synchroniser

Finally, the design will need a synchroniser module (also not shown) to pulse the synchronisation laser, though it is logical to deside upon Bob's synchronisation method first, as this will affect what pattern and period is needed.

4.5 FPGA software design, Bob

The software design in Bob has not yet been studied, though it will for the most part be a subset of Alice's. All the same modules will be used, and the main differences will be:

- Addition of photon detector gating.
- Receiving instead of sending synchronisation signals.
- Slightly modified main state machine.
- Only one balancer and output flip flop (as there is no intensity modulator in Bob).
- Modified PC communication.
- Slightly modified memory layout.

One way of synchronising based on received pulses is to run the synchronisation detector circuit at a higher frequency (e.g. 800 MHz), and generate the lower 100 MHz system clock from this. This allows adjustment of the system clock in 8 steps per period, and eliminates the need for an external clock generator.

A higher resolution solution for the APD gates will still need to be found, due to 3SL150's lack of live phase adjustments.

4.6 Digital to Analog Converters

The AD9746 is clocked by a 100 MHz 1.2 V toggling output of the FPGA, connected via coaxial cable on SMA connectors. The data itself is transmitted on a multi-wire planar cable via the HSMC interface of the FPGA development board (see appendix A). Every second wire is grounded, making the cable a waveguide and thus minimising noise errors in the transmission.

The input width of AD9647 is 14 bits. The development board however has a 16 bit input, where the 2 least significant bits are left unused. This makes the board compatible with the other DACs in the same family. To take full advantage of this, the output from the FPGA is a 16 bit value, and any of the boards in the family (AD974x, where $x \in \{1, 3, 5, 6, 7\}$) may be used with the current setup.

The transformers on the development board outputs have been removed to improve response times, resulting in two single ended outputs per channel [15]. At first, the positive output was used, but this resulted in non-symetric output with a DC bias (figure 4.6). This

problem is not present on the negative output. The cause for this is unknown, but must be a feature of the DAC.

Note that as the output is pseudo-random, and only one output per channel can be captured at a time, the to graphs do *not* show the same data.

The longitudinal grid lines in the negative output graph has been placed where the variance from intended level is at its smallest, and shows where the laser will be pulsed.

Also visible in figure 4.6 are transients which occur in the middle of each clock cycle, causing the levels to take twice as long to settle. This is one of the main reasons the system is set to 100 MHz during development. The effect is caused by an internal mechanism in the AD9647 intended to make the DAC better suited for frequency synthesis. This mechanism cannot be disabled, and causes the transients as a side effect.

4.7 Amplifiers

5 amplifiers were tested for usage with the modulators, by generating a high frequency (100 MHz) triangle wave and finding the input levels where the output became visibly distorted. Two of the amplifiers (EIN 503L and HP 8447E, figures 4.10 and 4.11) did not distort at levels available from the triangle wave generator, and a lower frequency sine wave had to be used.

Ideally, we would like 4.5 V input on the intensity modulator. As we can see from figures 4.7 to 4.11, the Mini-Circuits amplifiers and EIN 503L are powerful enough. However, only ZPUL-30P (figure 4.9) produced good square wave output, due to its frequency range's very low lower limit of 2.5 kHz (appendix D).

The phase modulator will need a higher input voltage, and it seems the Mini-Circuits ZHL-32A may be suitable for this.

4.7.1 ZPUL-30P input

The chosen amplifier for the intensity modulator has a gain of 29 dB. To get the desired output of 4.5 V, the input has to be 160 mV ($20 \log \frac{4.5}{0.16} \approx 29$).

The single ended output from AD9746 is 1 V, so this will have to be attenuated by $20 \log \frac{1}{0.160} \approx 16$ dB. Experiments show that we got a value closer to 4.5 V on the desired part of the amplifier output by using 17 dB attentuation on the DAC output, possibly because of the transient oscillation.

4.8 Random number generators

A random number generator (RNG), also called a deterministic random bitstream generator (DRBG), generates the bit values and chooses an encoding base. The user may swith between an externally connected RNG and an internal pseudo-RNG (PRNG), generating a random *looking* stream of bits. The internal PRNG is *not* cryptographically safe, and should only

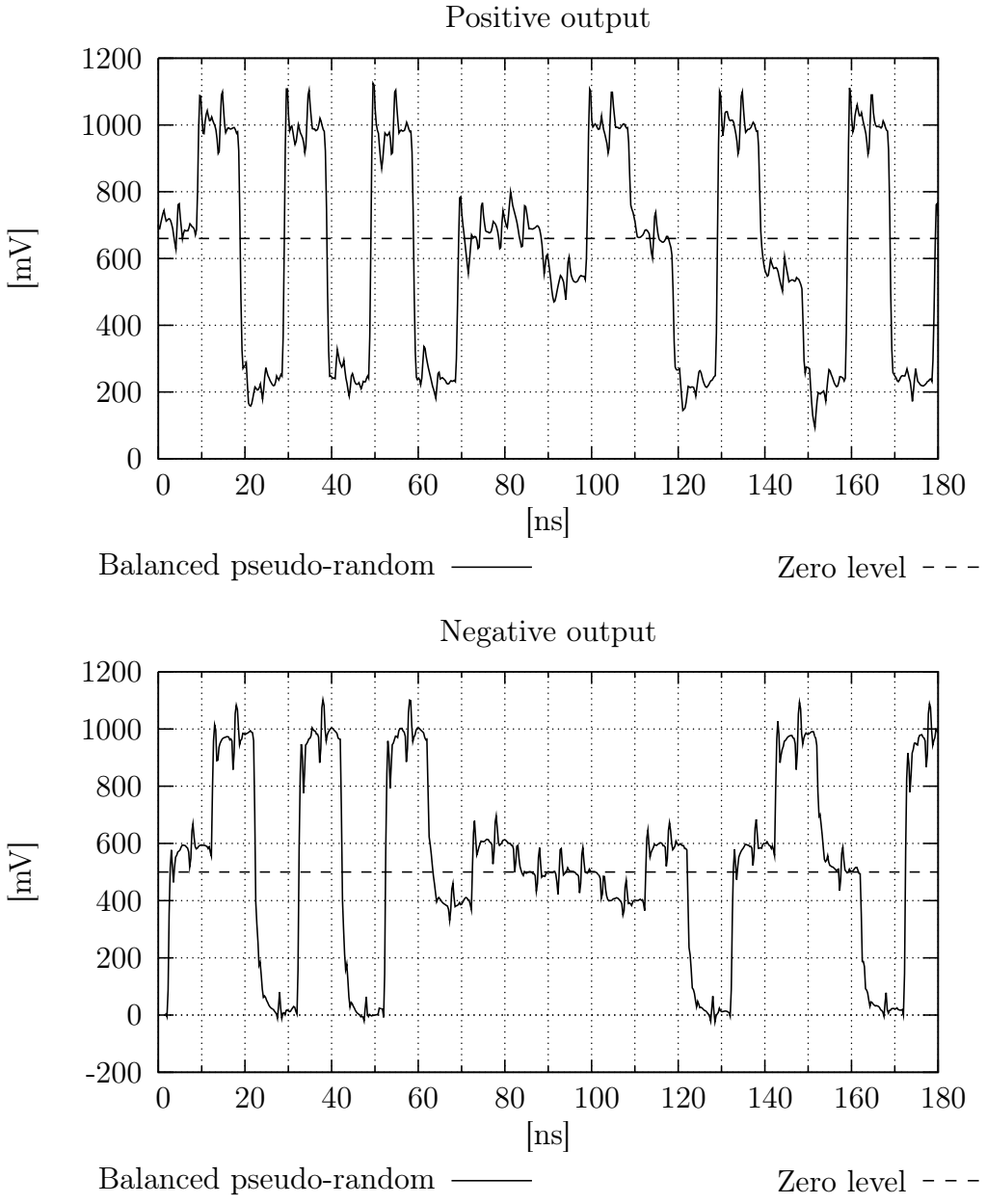


Figure 4.6: AD9746 single ended output at 100 MHz

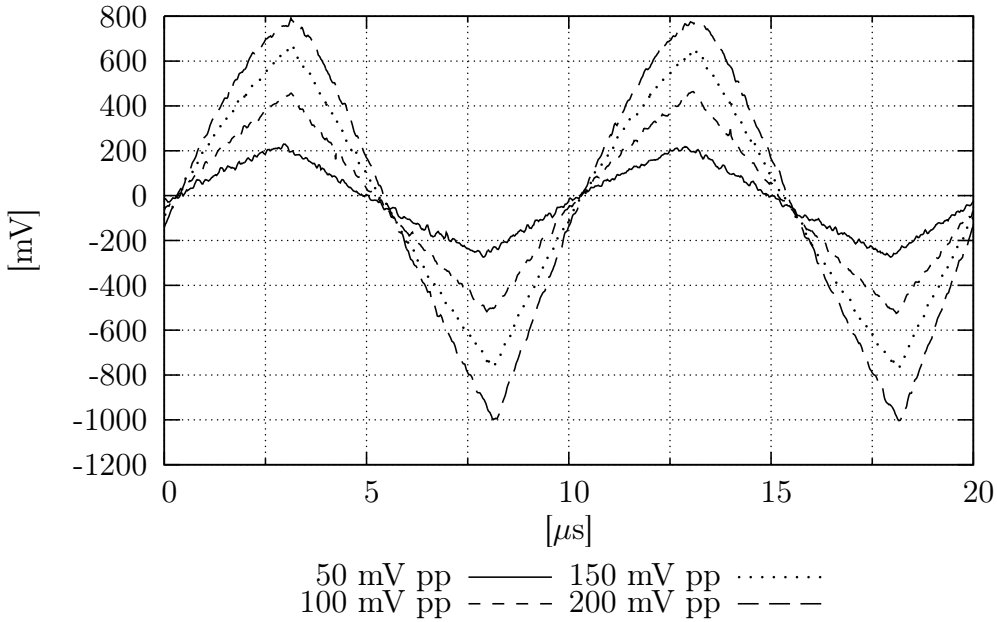


Figure 4.7: Sonoma Instrument 330

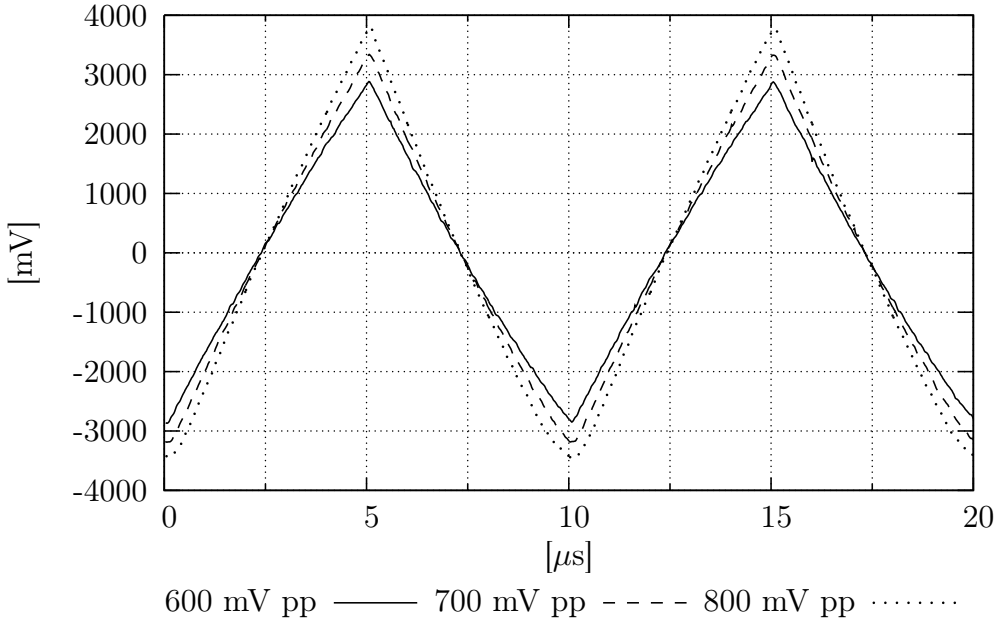


Figure 4.8: Mini-Circuits ZHL-32A (attenuated 10 dB)

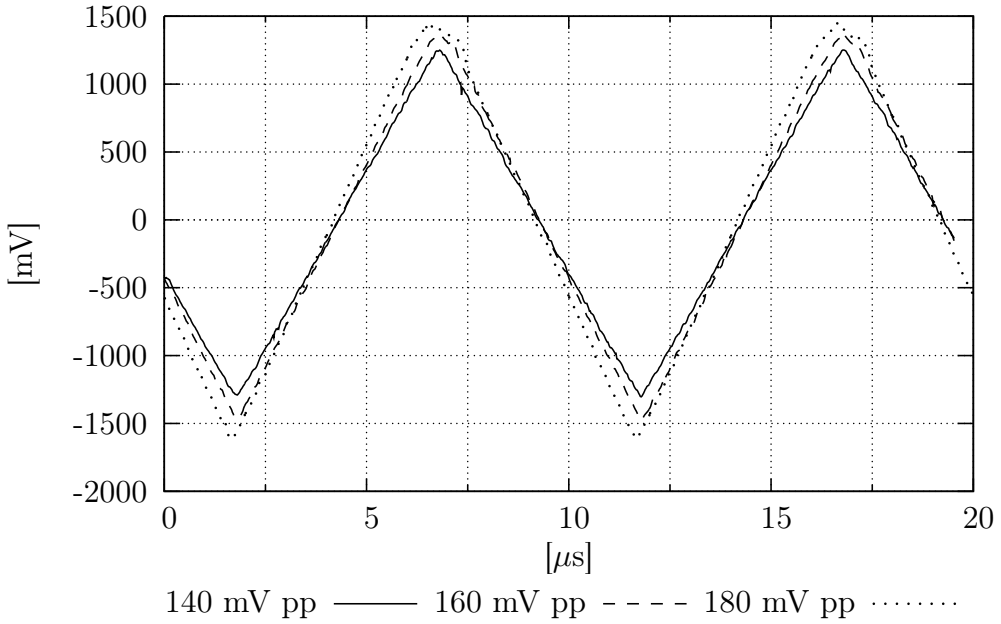


Figure 4.9: Mini-Circuits ZPUL-30P (attenuated 10 dB)

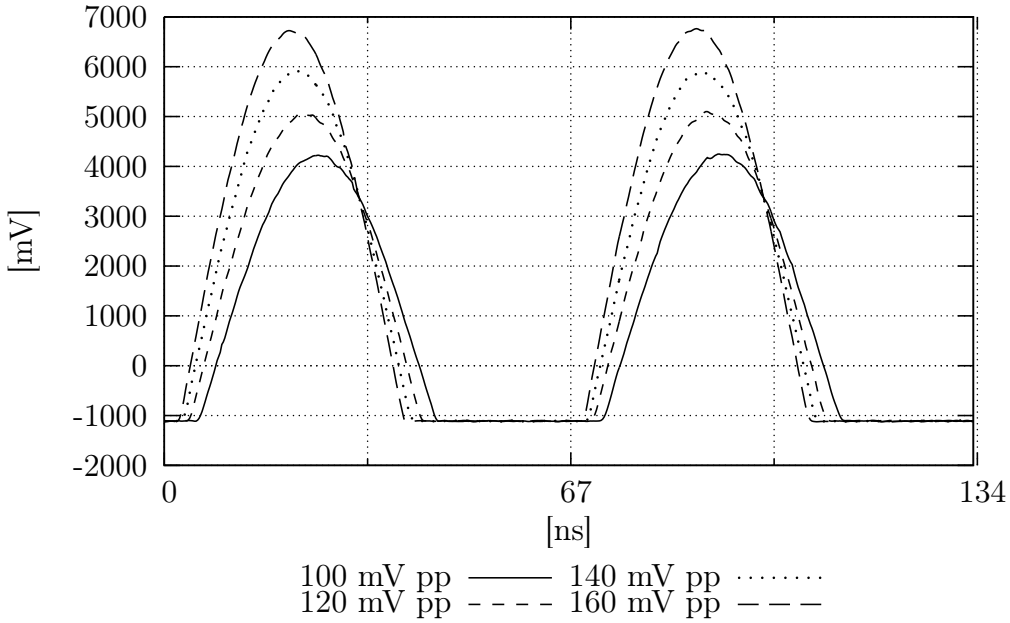


Figure 4.10: EIN 503L, clipped

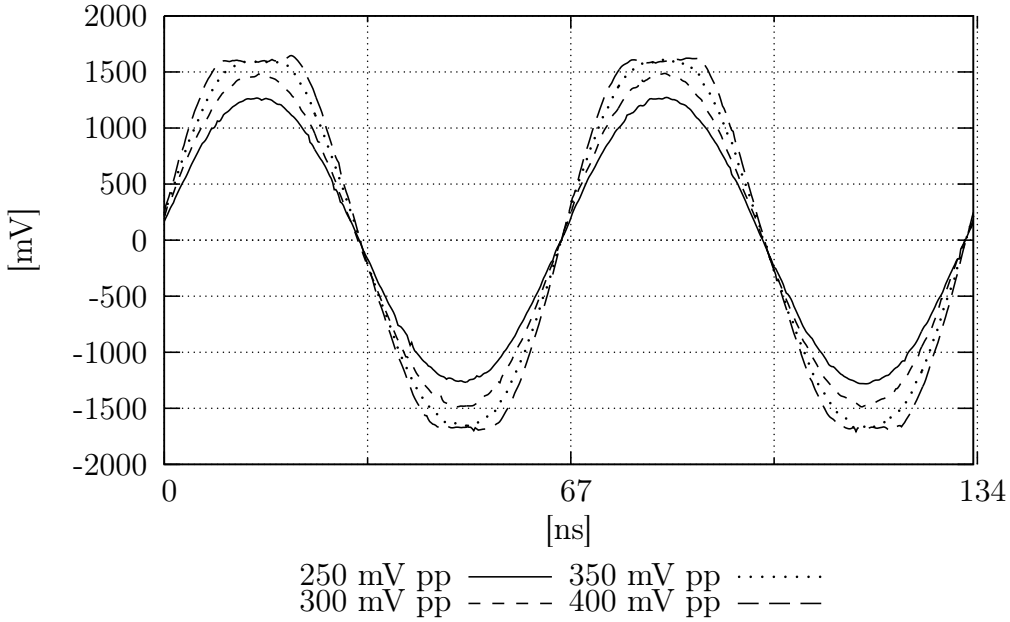


Figure 4.11: HP 8447E

be used for testing purposes. When the internal PRNG is enabled, the onboard LEDs will show an error pattern (see section 4.4.1).

All random data is gathered in a pool (*bitbucket* in figure 4.12). The internal or extrenal data line and clock is gated in based on whether the internal RNG is enabled or not. Since both the PRNG and any external RNG have a clock independent from the system clock, the pool is implemented with separate read and write clocks. The pool size is 131072 bits.

When data is available, it will be shifted into a holding buffer (*rbg output shift* in figure 4.12, code in appendix C) that is refilled every time it is read from outside the module. When the holding buffer is ready, a ready flag is set high for other modules to read.

Since the holding buffer runs on 800 MHz, and the outside system much slower, it has also been implemented with a separate read and write clock.

The flank detector (*negedge shifter*, *inst11* and *inst12*) detects when the module has been switched out of test mode. This will empty the pool, ensuring that test data will not be used further.

The system has not yet been tested with a true RNG.

4.8.1 RNG requirements

The completed system will have a customisable distribution for some of the random values, which will increase the random data requirements considerably. The number of bits needed depends on the lowest common demoninator of the probabilities.

For instance, if decoy states should occur for half of the pulses, you obviously need two

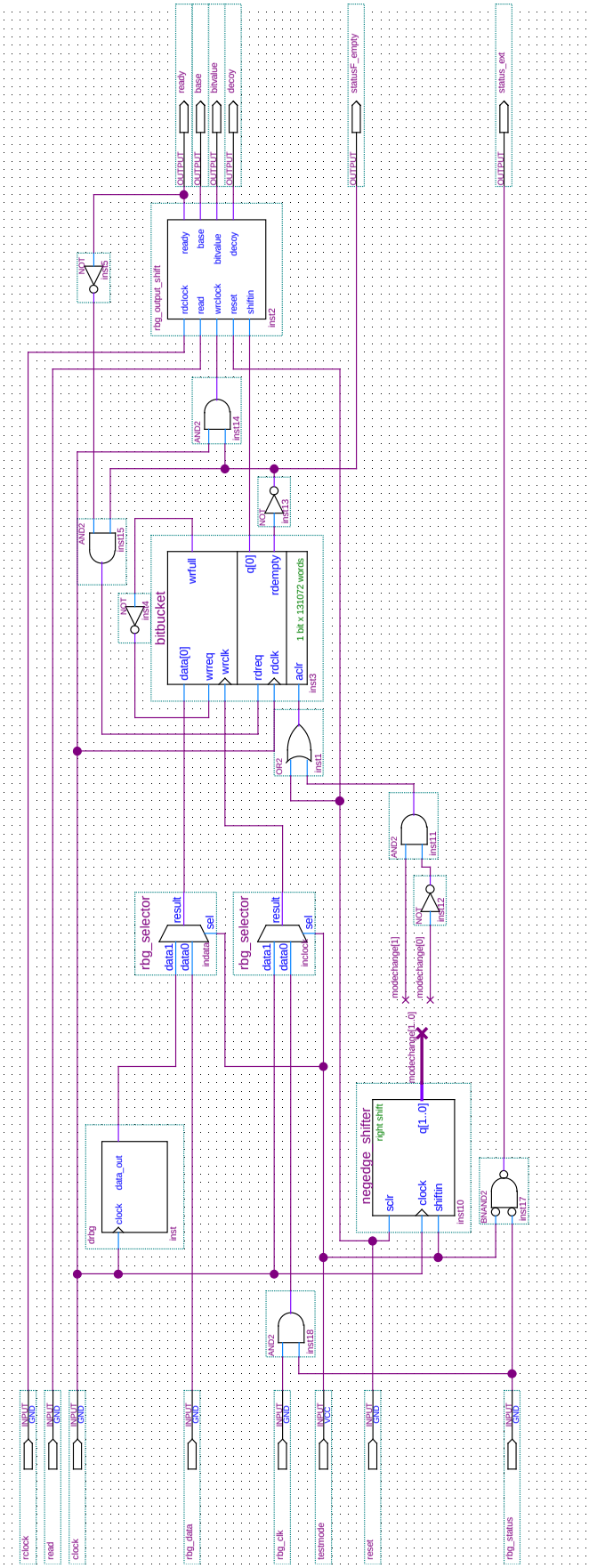


Figure 4.12: Random number generator subsystem

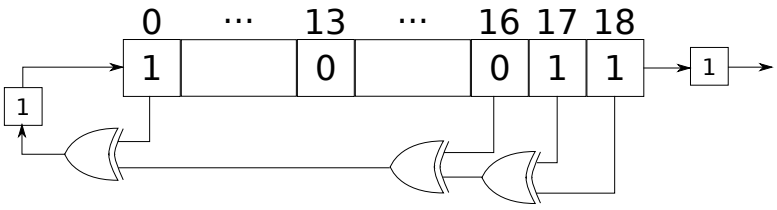


Figure 4.13: 19-bit linear feedback shift register

states (encoded in 1 bit). If however the two states should have a distribution such as $\frac{1}{7}$ and $\frac{6}{7}$, the lowest common demoninator is 7 (encodable in 3 bits). It will thus be necessary to generate at least 3 bits to get the desired distribution.

For the FPGA software, this means that the required number of generated bits depends on the resolution of the customisation. The smaller the probability adjustments can be, the higher the common demoninator will need to be.

This means that Alice's required bit rate will be significantly higher than Bob's, and will be further exacerbated if the distribution should be adjustable by the end user (as opposed to hard coded).

4.8.2 Internal PRNG

The internal random number generator is a simple 19-bit *linear feedback shift register* (LFSR). In such a generator, certain bits are tapped and a bitwise operation are performed on them (typically an *exclusive or*, *xor*). The resulting bit is shifted on to the end of the register. The bit that is shifted out is used as random data (see figure 4.13).

The PRNG is running on 800 MHz, the maximum the Stratix III FPGA is capable of. This is not enough to simulate distribution adjustments, but the PRNG can be duplicated if necessary.

By carefully choosing the operator and the bits to tap, one can create a maximum length LFSR. In such a register, all possible register values except one will occur once before the cycle restarts. In the implemented LFSR (and all LFSRs based on *xor*), the illegal value consists of all zeroes, as this would leave the register stuck. The code used can be seen in appendix B.

4.8.3 Physical RNG

The software assumes that a connected RNG has a clock output and an error output in addition to its data line. The clock output is used to shift the value on the data line into the pool (assuming the external RNG is selected).

As seen in figure 4.5, these three signals need to be routed from external pins (e.g. on the HSMC expansion board) to the *randomiser* sudsystem in the FPGA.

There is currently only room for one such RNG, but expanding the software to make room for more is trivial (for example by duplicating the *randomiser* software module).

Chapter 5

Final output

The resulting output from the chosen amplifier can be seen in figure 5.1 (attenuated due to the limited range of the oscilloscope), where output similar to the ones in figure 4.6 has been amplified.

The longitudinal grid lines have again been placed where the variance from intended level is at its smallest, and shows where the laser will be pulsed.

Appart from the undesired transients in the middle of each clock cycle, this is very close to the desired output. The level for each state falls within 0.10 V of the desired level on every clock cycle, corresponding to less than 0.32 V unattenuated.

This output is ready for use with the intensity modulator, though there was not enough time to perform such measurements.

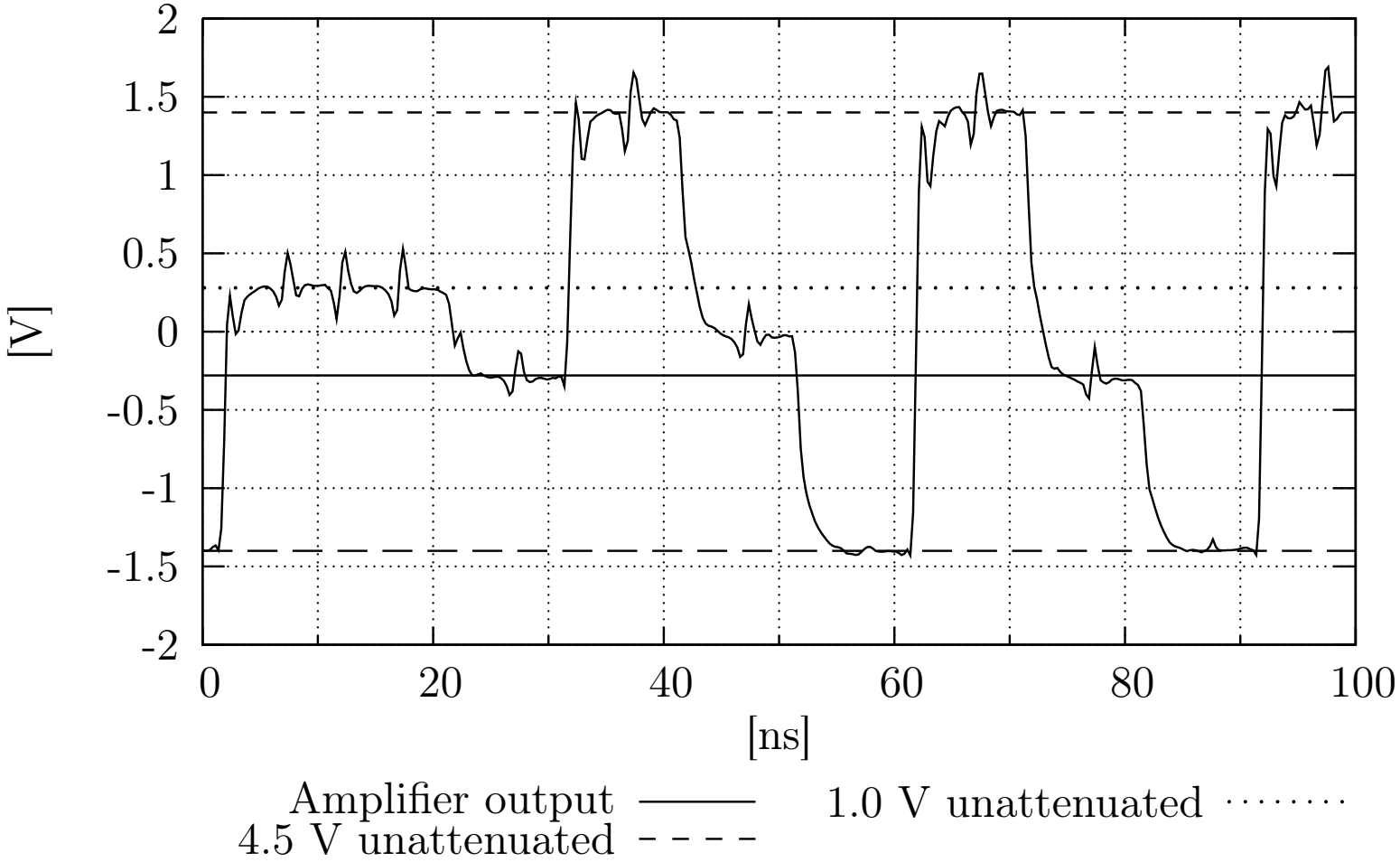


Figure 5.1: Final amplified signal (ZPUL-30p), attenuated 10 dB

Chapter 6

Conclusion

The goal of this thesis was two part. The first part, to set up the digital to analog converters and amplifiers and to demonstrate interference in the interferometer has been completed successfully, though only at a rate half of the target rate of 200 Mbit/s.

The second part, to implement the BB84 protocol for quantum key distribution as far as possible has not been fully achieved. The sender unit (*Alice*) is almost ready to generate all the necessary signals (without a customisable decoy state probability), but there has currently not been done any measurements on suitable amplifiers for the phase modulator.

For a full demonstration of BB84, it would also be necessary with a receiver unit (*Bob*).

The probability of generating a decoy state is currently locked to 12.5 %. For other distributions it may be necessary to add more random number generators, real or software.

6.1 Further work

The next logical step is to connect the amplifier output to the intensity modulator and calibrate the laser pulse timing. This relatively simple procedure should be enough to demonstrate a working decoy state generator, though there was not enough time to complete the measurements for this thesis.

The next logical step would be to finish the signal code and amplifier measurements for the phase modulator. At this stage, Alice will be a working sender unit for the BB84 protocol.

Next, a receiver unit must be created. Much of this work will already be done, as it has a lot in common with the sender. This will also include adding synchronisation code to both sender and receiver.

When the PC communication is implemented, it will be necessary with a governing state machine to guide the other software modules through each step of exchanging a complete key.

Bibliography

[1] C. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, pp. 656–715, ???? 1949. A footnote on the initial page says: "The material in this paper appeared in a confidential report, 'A Mathematical Theory of Cryptography', dated Sept. 1, 1946, which has now been declassified.".

[2] Wiesner, "Conjugate coding," *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, vol. 15, 1983.

[3] C. H. Bennett and G. Brassard, "An update on quantum cryptography," in *Advances in Cryptology: Proceedings of CRYPTO 84* (G. R. Blakley and D. Chaum, eds.), vol. 196 of *Lecture Notes in Computer Science*, pp. 475–480, Springer-Verlag, 1985, 19–22 Aug. 1984.

[4] A. Ekert, "Quantum cryptography based on bell's theorem," *Physical Review Letters*, vol. 67, pp. 661–663, Aug. 1991.

[5] Brassard and Salvail, "Secret key reconciliation by public discussion," in *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1993.

[6] C. H. Bennett, G. Brassard, and J.-M. Robert, "Privacy amplication by public discussion," *SIAM Journal on Computing*, vol. 17, pp. 210–229, 1988.

[7] Impagliazzo, Levin, and Luby, "Pseudo-random number generation from one-way functions," in *STOC: ACM Symposium on Theory of Computing (STOC)*, 1989.

[8] C. H. Bennett, G. Brasard, C. Crepeau, and U. M. Maurer, "Generalized privacy amplification," *IEEE Transactions on Information Theory*, vol. 41, pp. 1915–1923, Nov. 1995.

[9] J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1364–1396, 1999.

[10] G. Brassard, N. Lütkenhaus, T. Mor, and B. C. Sanders, "Limitations on practical quantum cryptography," *Phys. Rev. Lett.*, vol. 85, no. 1330, 2000.

[11] W. Y. Hwang, "Quantum key distribution with high loss: Toward global secure communication," *Phys. Rev. Lett.*, vol. 91, p. 057901, 2003.

[12] E. S. Simonsen, "Security of quantum key distribution source," Master's thesis, Norwegian University of Science and Technology, 2010.

[13] M. Ulianov, "Quantum key distribution system," Master's thesis, St. Petersburg State Polytechnical University, 2009.

[14] "Ad9741/ad9743/ad9745/ad9746/ad9747 data sheet (rev. 0)."

[15] "Ad9741/ad9743/ad9745/ad9746/ad9747 evaluation board schematic (rev. h)."

Appendix A

HSMC prototyping board pin mappings

P1D00 to P1D15 corresponds to odd numbered pins 9 to 39 on connector J9 of the AD9746 evaluation board. P2D00 to P2D15 corresponds to odd numbered pins 9 to 39 on connector J10.

DAC A	HSMC output	HSMC input	FPGA output
P1D00*	D61	132	AA1
P1D01*	D57	126	AB2
P1D02	D53	120	AC2
P1D03	D49	114	AE4
P1D04	D45	108	AE2
P1D05	D41	102	AF2
P1D06	D37	96	Y4
P1D07	D33	90	AG1
P1D08	D29	84	AF4
P1D09	D25	78	AH2
P1D10	D21	72	AJ2
P1D11	D17	66	AL2
P1D12	D13	60	AM2
P1D13	D9	54	AG4
P1D14	D5	48	AJ4
P1D15	D1	42	AJ9

* Not connected on the 14 bit AD9746

DAC B	HSMC output	HSMC input	FPGA output
P2D00*	D62	133	Y7
P2D01*	D58	127	AA6
P2D02	D54	121	Y5
P2D03	D50	115	AC7
P2D04	D46	109	AB5
P2D05	D42	103	AC5
P2D06	D38	97	W9
P2D07	D34	91	AD3
P2D08	D30	85	AE5
P2D09	D26	79	AD6
P2D10	D22	73	AF5
P2D11	D18	67	AE7
P2D12	D14	61	AH4
P2D13	D10	55	AC8
P2D14	D6	49	AB10
P2D15	D2	43	AL7
* Not connected on the 14 bit AD9746			
Laser driver:			
	CLKOUT0	39	AD28

Appendix B

19-bit linear feedback shift register

```
// DRBG: Deterministic Random Bit Generator
// This is intended as a cryptographically weak
// simulation of a physical random bit generator,
// implemented as a linear feedback shift register
// (LFSR). Unlike a physical random bit generator,
// it needs a clock input.
module drbg (clock, data_out);
  input  clock;
  output data_out;

  reg [18:0] r;

  assign data_out = r[18];

  always @ (posedge clock)
  begin
    if (r == 0)
      // Full length LFSRs will never move out of the 0 state.
      r <= 1;
    else
      r <= {r[0] ^ r[1] ^ r[2] ^ r[5], r[18:1]};
  end
endmodule
```

Appendix C

RNG subsystem output register

```
// Shift register for random bit generator output.
module rbg_output_shift (rdclock, read, wrclock, reset, shiftin,
                         ready, base, bitvalue, decoy);
  parameter WIDTH = 5;

  // For writing:
  input wrclock;     // Write clock.
  input shiftin;     // Bit value to shift in.
  reg prevrdtoggle; // Remember previous rdtoggle value.

  // For reading:
  input rdclock;     // Read clock
  input read;        // Signals that output was read this read cycle.
  reg rdtoggle;      // Is toggled on every read.

  // Other:
  input  reset;      // Synchronous reset.
  output reg ready; // Output is ready.

  // Actual shifted data.
  reg [WIDTH - 1:0] r;

  // Randomised data, wired from r[].
  output base;     assign base = r[0];
  output bitvalue; assign bitvalue = r[1];
  output decoy;    assign decoy = r[4] & r[3] & r[2];

  always @ (posedge wrclock)
  begin
    // Remember current rdtoggle.
    prevrdtoggle <= rdtoggle;
```

```
    if (reset)
    begin
      ready <= 0;
      r <= 1 << (WIDTH - 1);
    end
    else
    begin
      // Shift in unless already full.
      if (ready == 0)
      begin
        r <= {shiftin, r[WIDTH - 1:1]};

        // Output is ready when the first 1 is shifted out.
        if (r[0] == 1) ready <= 1;
      end

      // Valid output has been read.
      if (ready == 1 && rdtoggle != prevrdtoggle)
      begin
        ready <= 0;
        r <= 1 << (WIDTH - 1);
      end
    end
  end

  always @ (posedge rdclock)
  begin
    if (read == 1)
    begin
      // Data was read. Signal a new shift cycle.
      rdtoggle <= ~rdtoggle;
    end
  end
endmodule
```

Appendix D

Mini-Circuits ZPUL-30P data sheet

50Ω Non-Inverting 0.0025 to 700 MHz

CASE STYLE: S32

Connectors	Model	Price	Qty.
BNC	ZPUL-30P	$295.00 ea.	(1-9)

Features

- wide bandwidth 2.5 kHz to 700 MHz, useable to 1000 MHz
- excellent flatness, ±0.6 dB typ.
- can handle wide pulse width & (15µs typ.) with excellent rise/fall time (1.1 ns typ.)
- delay time, 1.5 ns typ.
- protected by US Patent, 6,943,629

Applications

- computers
- digital communication
- medical test set-ups

Pulse Amplifier Electrical Specifications

MODEL NO.	FREQUENCY (MHz)		GAIN (dB)		RISE/FALL TIME (ns)	PULSE WIDTH* (µs)	MAXIMUM POWER (dBm)		DYNAMIC RANGE		VSWR (:1) Typ.		DC POWER	
	f_L	f_U	Min.	Flatness Max.	Max.	Max.	Output (1 dB Compr.)	Input (no damage)	NF** (dB) Typ.	IP3 (dBm) Typ.	In	Out	Volt (V) Nom.	Current (mA) Max.
ZPUL-30P	0.0025	700	29	±1.0	1.5	6	+22***	+10	7.2	+34	2.0	2.0	24	400

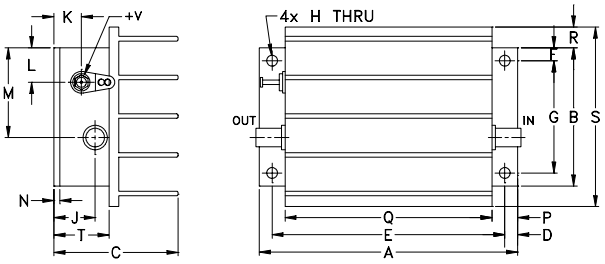
* Pulse width for less than 10% droop.
** Noise Figure tested above 10 MHz.
Open load is not recommended, potentially can cause damage.
With no load derate max input power by 20 dB
*** For 500-700 MHz, +20.5 dBm

Maximum Ratings

Operating Temperature	-20°C to 65°C
Storage Temperature	-55°C to 100°C
DC Voltage	+24.5V Max.

Permanent damage may occur if any of these limits are exceeded.

Outline Drawing



typical amplifier response to a pulse input